ahi_sub

COLLABORATORS			
	TITLE : ahi_sub		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 26, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ahi_sub

1.1 ahi_sub.guide

TABLE OF CONTENTS

1.2 [driver].audio/--background--

[driver].audio/--background--

OVERVIEW

DRIVER VERSIONS

The lowest supported driver version is 2. If you use any feature introduced in later versions of AHI, you should set the driver version to the same version as the features were introduced with. Example: You use PreTimer() and PostTimer(), and since these calls were added in V4 of ahi.device, your driver's version should be 4, too.

AUDIO ID NUMBERS

Just some notes about selecting ID numbers for different modes: It is up to the driver programmer to chose which modes should be available to the user. Take care when selecting.

```
       The upper word is the hardware ID, and can only be allocated by
       Martin Blom <lcs@lysator.liu.se>. The lower word is free, but in
       order to allow enhancements, please only use bit 0 to 3 for modes!
       If your driver supports multiple sound cards, use bit 12-15 to
       select card (first one is 0). If your sound card has multiple
       AD/DA converters, you can use bit 8-11 to select them (the first
       should be 0).

       Set the remaining bits to zero.

       Use AHI:Developer/Support/ScanAudioModes to have a look at the modes
       currently available. Use AHI:Developer/Support/sift to make sure your
       mode descriptor file is a legal IFF file.

       I do reserve the right to change the rules if I find them incorrect!
```

1.3 [driver].audio/AHIsub_#?

```
[driver].audio/AHIsub_#?

NAME
       AHIsub_SetEffect -- Set effect.
       AHIsub_SetFreq -- Set frequency.
       AHIsub_SetSound -- Set sound.
       AHIsub_SetVol -- Set volume and stereo panning.
       AHIsub_LoadSound -- Prepare a sound for playback.
       AHIsub_UnloadSound -- Discard a sound.


SYNOPSIS
       See functions in 'ahi.device'.


IMPLEMENTATION
       If AHIsub_AllocAudio() did not return with bit AHISB_MIXING set,
       all user calls to these function will be routed to the driver.

       If AHIsub_AllocAudio() did return with bit AHISB_MIXING set, the
       calls will first be routed to the driver, and only handled by
       'ahi.device' if the driver returned AHIS_UNKNOWN. This way it is
       possible to add effects that the sound card handles on its own, like
       filter and echo effects.

       For what each function does, see the autodocs for 'ahi.device'.


INPUTS
       See functions in 'ahi.device'.


NOTES
       See functions in 'ahi.device'.
```

```
SEE ALSO
       ahi.device/AHI_SetEffect(), ahi.device/AHI_SetFreq(),
       ahi.device/AHI_SetSound(), ahi.device/AHI_SetVol(),
       ahi.device/AHI_LoadSound(), ahi.device/AHI_UnloadSound()
```

1.4 [driver].audio/AHIsub_AllocAudio

```
[driver].audio/AHIsub_AllocAudio

NAME
       AHIsub_AllocAudio -- Allocates and initializes the audio hardware.


SYNOPSIS
       result = AHIsub_AllocAudio( tags, audioctrl);
       D0                          A1    A2

       ULONG AHIsub_AllocAudio( struct TagItem *, struct AHIAudioCtrlDrv * );


IMPLEMENTATION
       Allocate and initialize the audio hardware. Decide if and how you
       wish to use the mixing routines provided by 'ahi.device', by looking
       in the AHIAudioCtrlDrv structure and parsing the tag list for tags
       you support.

       1) Use mixing routines with timing:
           You will need to be able to play any number of samples from
           about 80 up to 65535 with low overhead.
           · Update AudioCtrl->ahiac_MixFreq to nearest value that your
             hardware supports.
           · Return AHISF_MIXING|AHISF_TIMING.
       2) Use mixing routines without timing:
           If the hardware can't play samples with any length, use this
           alternative and provide timing yourself. The buffer must
           take less than about 20 ms to play, preferable less than 10!
           · Update AudioCtrl->ahiac_MixFreq to nearest value that your
             hardware supports.
           · Store the number of samples to mix each pass in
             AudioCtrl->ahiac_BuffSamples.
           · Return AHISF_MIXING
           Alternatively, you can use the first method and call the
           mixing hook several times in a row to fill up a buffer.
           In that case, AHIsub_GetAttr(AHIDB_MaxPlaySamples) should
           return the size of the buffer plus AudioCtrl->ahiac_MaxBuffSamples.
           If the buffer is so large that it takes more than (approx.) 10 ms to
           play it for high sample frequencies, AHIsub_GetAttr(AHIDB_Realtime)
           should return FALSE.
       3) Don't use mixing routines:
           If your hardware can handle everything without using the CPU to
           mix the channels, you tell 'ahi.device' this by not setting
           either the AHISB_MIXING or the AHISB_TIMING bit.
```

```
       If you can handle stereo output from the mixing routines, also set
       bit AHISB_KNOWSTEREO.

       If you can handle hifi (32 bit) output from the mixing routines,
       set bit AHISB_KNOWHIFI.

       If this driver can be used to record samples, set bit AHISB_CANRECORD,
       too (regardless if you use the mixing routines in AHI or not).

       If the sound card has hardware to do DSP effects, you can set the
       AHISB_CANPOSTPROCESS bit. The output from the mixing routines will
       then be two separate buffers, one wet and one dry. You should then
       apply the Fx on the wet buffer, and post-mix the two buffers before
       you send the samples to the DAC. (V4)


INPUTS
       tags - pointer to a taglist.
       audioctrl - pointer to an AHIAudioCtrlDrv structure.


TAGS
       The tags are from the audio database (AHIDB_#? in <devices/ahi.h>),
       NOT the tag list the user called ahi.device/AHI_AllocAudio() with.


RESULT
       Flags, defined in <libraries/ahi_sub.h>.


EXAMPLE


NOTES
       You don't have to clean up on failure, AHIsub_FreeAudio() will
       always be called.


BUGS


SEE ALSO
       AHIsub_FreeAudio(), AHIsub_Start()
```

1.5 [driver].audio/AHIsub_Disable

```
[driver].audio/AHIsub_Disable

NAME
       AHIsub_Disable -- Temporary turn off audio interrupt/task


SYNOPSIS
       AHIsub_Disable( audioctrl );
```

```
                      A2

       void AHIsub_Disable( struct AHIAudioCtrlDrv * );


IMPLEMENTATION
       If you are lazy, then call exec.library/Disable().
       If you are smart, only disable your own interrupt or task.


INPUTS
       audioctrl - pointer to an AHIAudioCtrlDrv structure.


NOTES
       This call should be guaranteed to preserve all registers.
       This call nests.


SEE ALSO
       AHIsub_Enable(), exec.library/Disable()
```

1.6 [driver].audio/AHIsub_Enable

```
[driver].audio/AHIsub_Enable

NAME
       AHIsub_Enable -- Turn on audio interrupt/task


SYNOPSIS
       AHIsub_Enable( audioctrl );
                      A2

       void AHIsub_Enable( struct AHIAudioCtrlDrv * );


IMPLEMENTATION
       If you are lazy, then call exec.library/Enable().
       If you are smart, only enable your own interrupt or task.


INPUTS
       audioctrl - pointer to an AHIAudioCtrlDrv structure.


NOTES
       This call should be guaranteed to preserve all registers.
       This call nests.


SEE ALSO
       AHIsub_Disable(), exec.library/Enable()
```

1.7 [driver].audio/AHIsub_FreeAudio

```
[driver].audio/AHIsub_FreeAudio

NAME
       AHIsub_FreeAudio -- Deallocates the audio hardware.


SYNOPSIS
       AHIsub_FreeAudio( audioctrl );
                         A2

       void AHIsub_FreeAudio( struct AHIAudioCtrlDrv * );


IMPLEMENTATION
       Deallocate the audio hardware and other resources allocated in
       AHIsub_AllocAudio(). AHIsub_Stop() will always be called by
       'ahi.device' before this call is made.


INPUTS
       audioctrl - pointer to an AHIAudioCtrlDrv structure.


NOTES
       It must be safe to call this routine even if AHIsub_AllocAudio()
       was never called, failed or called more than once.


SEE ALSO
       AHIsub_AllocAudio()
```

1.8 [driver].audio/AHIsub_GetAttr

```
[driver].audio/AHIsub_GetAttr

NAME
       AHIsub_GetAttr -- Returns information about audio modes or driver


SYNOPSIS
       AHIsub_GetAttr( attribute, argument, default, taglist, audioctrl );
       D0              D0         D1        D2       A1       A2

       LONG AHIsub_GetAttr( ULONG, LONG, LONG, struct TagItem *,
                            struct AHIAudioCtrlDrv * );
```

```
IMPLEMENTATION
       Return the attribute based on a tag list and an AHIAudioCtrlDrv
       structure, which are the same that will be passed to
       AHIsub_AllocAudio() by 'ahi.device'. If the attribute is
       unknown to you, return the default.


INPUTS
       attribute - Is really a Tag and can be one of the following:
           AHIDB_Bits - Return how many output bits the tag list will
               result in.
           AHIDB_MaxChannels - Return the resulting number of channels.
           AHIDB_Frequencies - Return how many mixing/sampling frequencies
               you support
           AHIDB_Frequency - Return the argument:th frequency
               Example: You support 3 frequencies 32, 44.1 and 48 kHz.
                   If argument is 1, return 44100.
           AHIDB_Index - Return the index which gives the frequency closest
               to argument.
               Example: You support 3 frequencies 32, 44.1 and 48 kHz.
                   If argument is 40000, return 1 (=> 44100).
           AHIDB_Author - Return pointer to name of driver author:
               "Martin 'Leviticus' Blom"
           AHIDB_Copyright - Return pointer to copyright notice, including
               the '©' character: "© 1996 Martin Blom" or "Public Domain"
           AHIDB_Version - Return pointer version string, normal Amiga
               format: "paula 1.5 (18.2.96)\r\n"
           AHIDB_Annotation - Return pointer to an annotation string, which
               can be several lines.
           AHIDB_Record - Are you a sampler, too? Return TRUE or FALSE.
           AHIDB_FullDuplex - Return TRUE or FALSE.
           AHIDB_Realtime - Return TRUE or FALSE.
           AHIDB_MaxPlaySamples - Normally, return the default. See
               AHIsub_AllocAudio(), section 2.
           AHIDB_MaxRecordSamples - Return the size of the buffer you fill
               when recording.

           The following are associated with AHIsub_HardwareControl() and are
           new for V2.
           AHIDB_MinMonitorVolume
           AHIDB_MaxMonitorVolume - Return the lower/upper limit for
               AHIC_MonitorVolume. If unsupported but always 1.0, return
               1.0 for both.
           AHIDB_MinInputGain
           AHIDB_MaxInputGain - Return the lower/upper limit for
               AHIC_InputGain. If unsupported but always 1.0, return 1.0 for
               both.
           AHIDB_MinOutputVolume
           AHIDB_MaxOutputVolume - Return the lower/upper limit for
               AHIC_OutputVolume.
           AHIDB_Inputs - Return how many inputs you have.
           AHIDB_Input - Return a short string describing the argument:th
               input. Number 0 should be the default one. Example strings
               can be "Line 1", "Mic", "Optical" or whatever.
           AHIDB_Outputs - Return how many outputs you have.
           AHIDB_Output - Return a short string describing the argument:th
               output. Number 0 should be the default one. Example strings
```

```
            can be "Line 1", "Headphone", "Optical" or whatever.
        argument - extra info for some attributes.
        default - What you should return for unknown attributes.
        taglist - Pointer to a tag list that eventually will be fed to
            AHIsub_AllocAudio(), or NULL.
        audioctrl - Pointer to an AHIAudioCtrlDrv structure that eventually
            will be fed to AHIsub_AllocAudio(), or NULL.


NOTES


SEE ALSO
        AHIsub_AllocAudio(), AHIsub_HardwareControl(),
        ahi.device/AHI_GetAudioAttrsA()
```

1.9 [driver].audio/AHIsub_HardwareControl

```
[driver].audio/AHIsub_HardwareControl

NAME
        AHIsub_HardwareControl -- Modify sound card settings


SYNOPSIS
        AHIsub_HardwareControl( attribute,  argument, audioctrl );
        D0                      D0          D1        A2

        LONG AHIsub_HardwareControl( ULONG, LONG, struct AHIAudioCtrlDrv * );


IMPLEMENTATION
        Set or return the state of a particular hardware component. AHI uses
        AHIsub_GetAttr() to supply the user with limits and what tags are
        available.


INPUTS
        attribute - Is really a Tag and can be one of the following:
            AHIC_MonitorVolume - Set the input monitor volume to argument.
            AHIC_MonitorVolume_Query - Return the current input monitor
                volume (argument is ignored).

            AHIC_InputGain - Set the input gain to argument. (V2)
            AHIC_InputGain_Query (V2)

            AHIC_OutputVolume - Set the output volume to argument. (V2)
            AHIC_OutputVolume_Query (V2)

            AHIC_Input - Use the argument:th input source (default is 0). (V2)
            AHIC_Input_Query (V2)

            AHIC_Output - Use the argument:th output destination (default
                is 0). (V2)
```

```
           AHIC_Output_Query (V2)

       argument - What value attribute should be set to.
       audioctrl - Pointer to an AHIAudioCtrlDrv structure.


RESULT
       Return the state of selected attribute. If you were asked to set
       something, return TRUE. If attribute is unknown to you or unsupported,
       return FALSE.


NOTES
       This call must be safe from interrupts.


SEE ALSO
       ahi.device/AHI_ControlAudioA(), AHIsub_GetAttr()
```

1.10 [driver].audio/AHIsub_Start

```
[driver].audio/AHIsub_Start

NAME
       AHIsub_Start -- Starts playback or recording


SYNOPSIS
       error = AHIsub_Start( flags, audioctrl );
       D0                    D0     A2

       ULONG AHIsub_Start(ULONG, struct AHIAudioCtrlDrv * );


IMPLEMENTATION
       What to do depends what you returned in AHIsub_AllocAudio().

     * First, assume bit AHISB_PLAY in flags is set. This means that you
       should begin playback.

     - AHIsub_AllocAudio() returned AHISF_MIXING|AHISF_TIMING:

       A) Allocate a mixing buffer of ahiac_BuffSize bytes. The buffer must
          be long aligned!
       B) Create/start an interrupt or task that will do 1-6 over and over
          again until AHIsub_Stop() is called. Note that it is not a good
          idea to do the actual mixing and conversion in a real hardware
          interrupt. Signal a task or create a Software Interrupt to do
          the number crunching.

       1) Call the user Hook ahiac_PlayerFunc with the following parameters:
                  A0 - (struct Hook *)
                  A2 - (struct AHIAudioCtrlDrv *)
                  A1 - Set to NULL.
```

2) [Call the ahiac_PreTimer function. If it returns TRUE (Z will be cleared so you don't have to test d0), skip step 3 and 4. This is used to avoid overloading the CPU. This step is optional. A2 is assumed to point to struct AHIAudioCtrlDrv. All registers except d0 are preserved. (V4)]

3) Call the mixing Hook (ahiac_MixerFunc) with the following parameters:

```
        A0 - (struct Hook *)           - The Hook itself
        A2 - (struct AHIAudioCtrlDrv *)
        A1 - (WORD *[])                - The mixing buffer.
```

   Note that ahiac_MixerFunc preserves ALL registers.
   The user Hook ahiac_SoundFunc will be called by the mixing routine when a sample have been processed, so you don't have to worry about that.
   How the buffer will be filled is indicated by ahiac_Flags. It is always filled with signed 16-bit (32 bit if AHIACB_HIFI in in ahiac_Flags is set) words, even if playback is 8 bit. If AHIDBB_STEREO is set (in ahiac_Flags), data for left and right channel are interleaved:

```
 1st sample left channel,
 1st sample right channel,
 2nd sample left channel,
 ...,
 ahiac_BuffSamples:th sample left channel,
 ahiac_BuffSamples:th sample right channel.
```

   If AHIDBB_STEREO is cleared, the mono data is stored:

```
 1st sample,
 2nd sample,
 ...,
 ahiac_BuffSamples:th sample.
```

   Note that neither AHIACB_STEREO nor AHIACB_HIFI will be set if you didn't report that you understand these flags when AHI_AllocAudio() was called.

   For AHI V2, the type of buffer is also available in ahiac_BuffType. It is suggested that you use this value instead. ahiac_BuffType can be one of AHIST_M16S, AHIST_S16S, AHIST_M32S and AHIST_S32S.

4) Convert the buffer if needed and feed it to the audio hardware. Note that you may have to clear CPU caches if you are using DMA to play the buffer, and the buffer is not allocated in non-cachable RAM.

5) [Call the ahiac_PostTimer function. A2 is assumed to point to struct AHIAudioCtrlDrv. All registers are preserved. (V4)]

6) Wait until the whole buffer has been played, then repeat.

Use double buffering if possible!

You may DECREASE ahiac_BuffSamples slightly, for example to force an even number of samples to be mixed. By doing this you will make ahiac_PlayerFunc to be called at wrong frequency so be careful! Even if ahiac_BuffSamples is defined ULONG, it will never be greater than 65535.

ahiac_BuffSize is the largest size of the mixing buffer that will be needed until AHIsub_Stop() is called.

ahiac_MaxBuffSamples is the maximum number of samples that will be mixed (until AHIsub_Stop() is called). You can use this value if you need to allocate DMA buffers.

ahiac_MinBuffSamples is the minimum number of samples that will be mixed. Most drivers will ignore it.

If AHIsub_AllocAudio() returned with the AHISB_CANPOSTPROCESS bit set, ahiac_BuffSize is large enough to hold two buffers. The mixing buffer will be filled with the wet buffer first, immediately followed by the dry buffer. I.e., ahiac_BuffSamples sample frames wet data, then ahiac_BuffSamples sample frames dry data. The DSP fx should only be applied to the wet buffer, and the two buffers should then be added together. (V4)

- If AHIsub_AllocAudio() returned AHISF_MIXING, do as described above, except calling ahiac_PlayerFunc. ahiac_PlayerFunc should be called ahiac_PlayerFreq times per second, clocked by timers on your sound card or by using 'realtime.library'. No other Amiga resources may be used for timing (like direct CIA timers).
  ahiac_MinBuffSamples and ahiac_MaxBuffSamples are undefined if AHIsub_AllocAudio() returned AHISF_MIXING (AHISB_TIMING bit not set).

- If AHIsub_AllocAudio() returned with neither the AHISB_MIXING nor the AHISB_TIMING bit set, then just start playback. Don't forget to call ahiac_PlayerFunc ahiac_PlayerFreq times per second. Only your own timing hardware or 'realtime.library' may be used. Note that ahiac_MixerFunc, ahiac_BuffSamples, ahiac_MinBuffSamples, ahiac_MaxBuffSamples and ahiac_BuffSize are undefined. ahiac_MixFreq is the frequency the user wants to use for recording, if you support that.

* Second, assume bit AHISB_RECORD in flags is set. This means that you should start to sample. Create a interrupt or task that does the following:

  Allocate a buffer (you chose size, but try to keep it reasonable small to avoid delays - it is suggested that RecordFunc is called at least 4 times/second for the lowers sampling rate, and more often for higher rates), and fill it with the sampled data. The buffer must be long aligned, and it's size must be evenly divisible by four. The format should always be AHIST_S16S (even with 8 bit mono samplers), which means:
      1st sample left channel,
      1st sample right channel (same as prev. if mono),
      2nd sample left channel,
      ... etc.
  Each sample is a signed word (WORD). The sample rate should be equal to the mixing rate.

  Call the ahiac_SamplerFunc Hook with the following parameters:
      A0 - (struct Hook *)   - The Hook itself
      A2 - (struct AHIAudioCtrlDrv *)

```
            A1 - (struct AHIRecordMessage *)
        The message should be filled as follows:
            ahirm_Type - Set to AHIST_S16S.
            ahirm_Buffer - A pointer to the filled buffer.
            ahirm_Samples - How many sample frames stored.
        You must not destroy the buffer until next time the Hook is called.

        Repeat until AHIsub_Stop() is called.

      * Note that both bits may be set when this function is called.


INPUTS
        flags - See <libraries/ahi_sub.h>.
        audioctrl - pointer to an AHIAudioCtrlDrv structure.


RESULT
        Returns AHIE_OK if successful, else an error code as defined
        in <devices/ahi.h>. AHIsub_Stop() will always be called, even
        if this call failed.


NOTES
        The driver must be able to handle multiple calls to this routine
        without preceding calls to AHIsub_Stop().


SEE ALSO
        AHIsub_Update(), AHIsub_Stop()
```

1.11 [driver].audio/AHIsub_Stop

```
[driver].audio/AHIsub_Stop

NAME
        AHIsub_Stop -- Stops playback.


SYNOPSIS
        AHIsub_Stop( flags, audioctrl );
                     D0     A2

        void AHIsub_Stop( ULONG, struct AHIAudioCtrlDrv * );


IMPLEMENTATION
        Stop playback and/or recording, remove all resources allocated by
        AHIsub_Start().


INPUTS
        flags - See <libraries/ahi_sub.h>.
        audioctrl - pointer to an AHIAudioCtrlDrv structure.
```

```
NOTES
       It must be safe to call this routine even if AHIsub_Start() was never
       called, failed or called more than once.


SEE ALSO
       AHIsub_Start()
```

1.12 [driver].audio/AHIsub_Update

```
[driver].audio/AHIsub_Update

NAME
       AHIsub_Update -- Update some variables


SYNOPSIS
       AHIsub_Update( flags, audioctrl );
                      D0     A2

       void AHIsub_Update(ULONG, struct AHIAudioCtrlDrv * );


IMPLEMENTATION
       All you have to do is to update some variables:
       Mixing & timing: ahiac_PlayerFunc, ahiac_MixerFunc, ahiac_SamplerFunc,
       ahiac_BuffSamples (and perhaps ahiac_PlayerFreq if you use it).
       Mixing only: ahiac_PlayerFunc, ahiac_MixerFunc, ahiac_SamplerFunc and
           ahiac_PlayerFreq.
       Nothing: ahiac_PlayerFunc, ahiac_SamplerFunc and ahiac_PlayerFreq.


INPUTS
       flags - Currently no flags defined.
       audioctrl - pointer to an AHIAudioCtrlDrv structure.


RESULT


NOTES
       This call must be safe from interrupts.


SEE ALSO
       AHIsub_Start()
```