Atmel AVR32839: AVR UC3 Audio Player

32-bit Atmel Microcontroller

Application Note

Features

- **Music played over mass storage device**
  - **USB disk**
  - **SD card**
- **Software audio decoders**
- **FAT file system**
- **USB host mass storage class (MSC)**
- **Standalone - low memory footprint (code & RAM) and no operating system dependency**
- **Configurable audio output**
  - **External I^{2}S audio DAC**
  - **Internal audio bitstream DAC**
- **Local control with a keypad**

1. Introduction

This application note will help the reader to use the Atmel® AVR® UC3 Audio Player. The software implements a software audio decoder, a file system, a USB disk as well as SD/MMC card mass storage device support.

For more information about the Atmel AVR UC3 architecture, please refer to the appropriate documents available from http://www.atmel.com/avr.

2. Licenses

Some third party firmware modules are delivered under licensing. For details on the licenses, refer to section "Module licenses" on page 25.

3. Requirements

The software provided with this application note requires several components:

- A computer running Microsoft® Windows® 2000, Windows XP, Windows Vista®, Windows 7, or Linux®
- A development environment like:
  - Atmel AVR Studio® 5
  - Atmel AVR GNU Toolchain (GCC)
  - IAR Embedded Workbench® for AVR32
- An Atmel AVR JTAGICE mkII, JTAGICE 3, or AVR ONE! debugger

4. Atmel AVR UC3 Audio Player

4.1 Overview

The UC3 audio player is a generic audio player interface and is designed to support multiple audio formats like MP3, WMA, AAC.

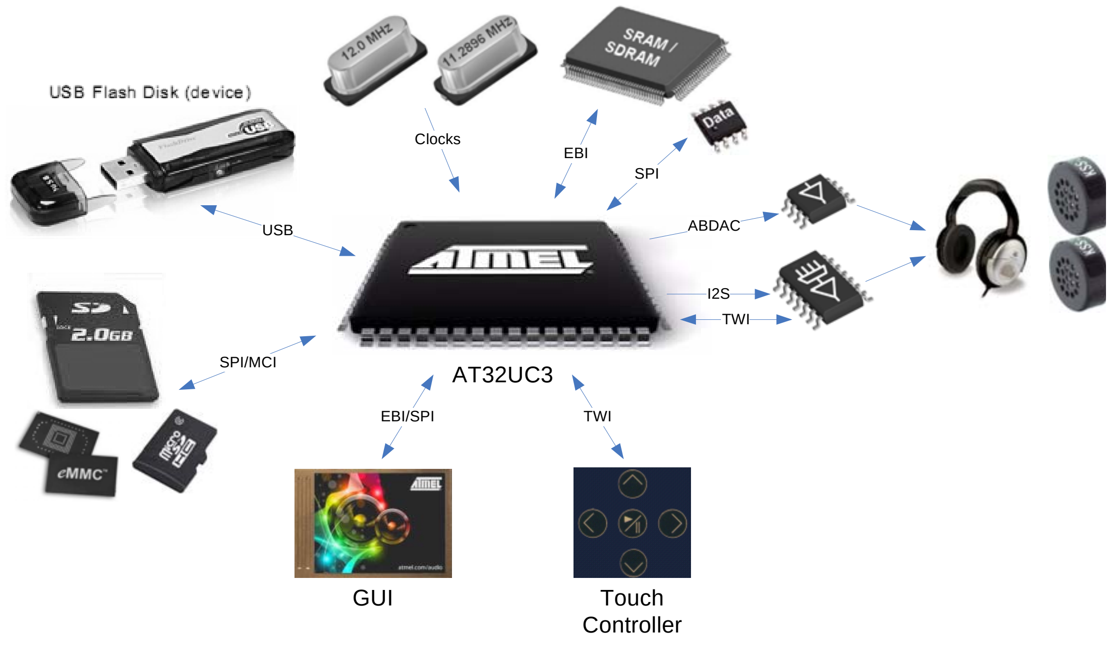
This document details the generic audio player implementation while specific audio codec implementation is detailed in the following separated application notes:

- Atmel AVR32840: AVR UC3 Audio Player - MAD® MP3 Decoder
- Atmel AVR32841: AVR UC3 Audio Player - Spirit MP3 Decoder
- Atmel AVR32842: AVR UC3 Audio Player - Microsoft WMA Decoder
- Atmel AVR32843: AVR UC3 Audio Player - Fraunhofer AAC Decoder

4.2 Block diagram

The following block diagram shows the UC3 interfacing a USB disk and SD card and outputting the audio stream from the storage device to an external DAC. The user can control the player using a keypad, running a customizable human-machine interface (HMI).

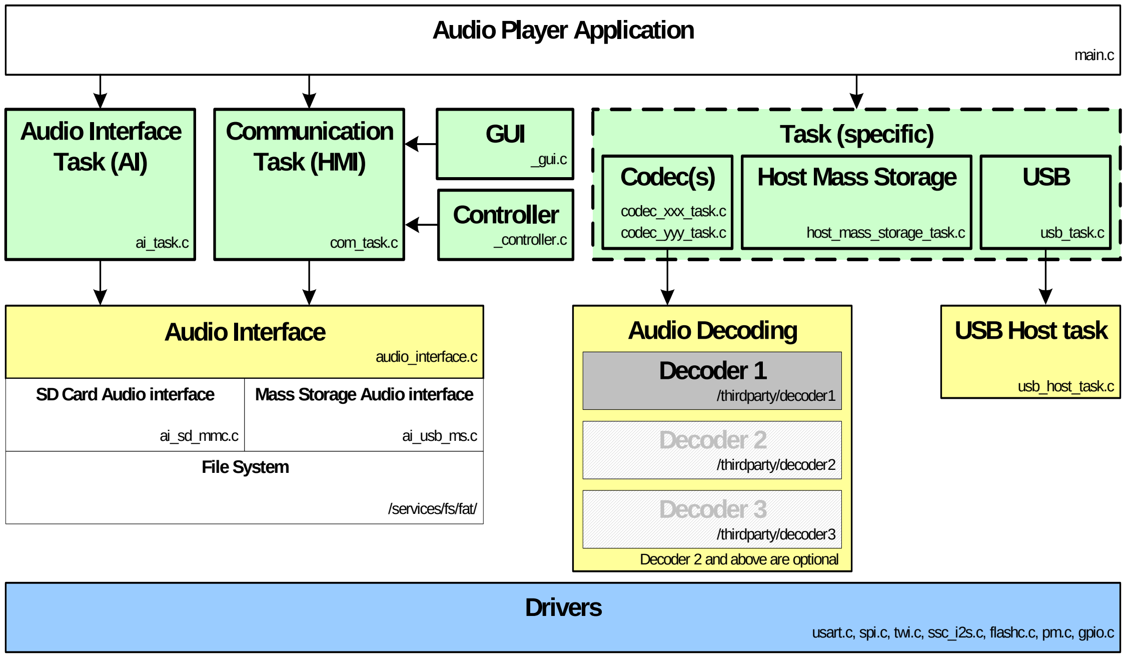
Figure 4-1. Block diagram.



4.3 Software architecture

Figure 4-2 shows the basic software architecture of the player.

Figure 4-2. Software architecture.



The application does not require any operating system to run. The main() function is in charge of calling the software "tasks" (using a scheduler) that make audio decoding, HMI, and USB management possible.

The main loop of the application is a simple free-running task scheduler:

```
while (true)
{
    ai_task();
    com_task();
    task();
}
```

- The audio interface task (ai_task) handles dynamic support for any new plugged device types. For example, if a new device using another class than the mass storage class is plugged in, and of course if this class is supported by the application, then this task will tell the audio interface to link the "Audio Interface" API to a low level API that supports this class. All the mechanism in place to dynamically link an API to this abstract API is located in the /avr32/services/audio/audio_player/audio_interface.[c,h] files
- The communication task (com_task) contains the HMI of the application. **This task holds the real intelligence of the user application and interfaces directly with the Audio Interface. This is the task that the user should modify for his own application**
- The last task (task) is a macro definition used for feature tasks. In the audio player, the modules involved will be the audio decoder(s), the USB stack and the USB host mass storage task:
  – USB task: this task handles the USB stack and events

– Host mass storage task:	this task checks for new devices connection and initialize them using the USB Mass Storage class
– Audio codec tasks:	these tasks manage the audio decoding

Here is an example of a player implementing an MP3 and WMA decoders.

```
#define task()              \
{                           \
    task_usb();             \
    task_usb_ms();          \
    task_mp3();             \
    task_wma();             \
}
```

It should be noted that the Atmel AVR UC3 Audio Player also supports digital music streaming from Apple® devices.

More information can be provided to MFi licensees.

5. Audio interface API

5.1 Overview

The Audio Interface (AI) manages disk navigation, audio navigation and audio control (see sections "Disk Navigator" on page 6, "Audio Navigator" on page 6 and "Audio control" on page 7). Thus, the user does not have to directly interface with the file system and audio control APIs. This greatly simplifies the software architecture.

The Audio Interface can be used in two different ways:

- Using "**asynchronous**" functions, which result/effect may not be produced in one single iteration. Using these functions usually leads to the use of state-machines in the user firmware (since one must wait for the completion of a command before launching a new one), and has the advantage of reducing the risks of audio underrun. Asynchronous functions always have the "ai_async" prefix. Note that only one asynchronous function can be used at a time
- Using "**synchronous**" functions, which are executed immediately. This drastically simplifies the user firmware architecture (no use of state-machines since the synchronous AI functions are immediately executed) but *may* produce audio underrun since the execution time of these functions may be too long. Synchronous functions just have the "ai_" prefix

All functions of the Audio Interface have a synchronous and asynchronous interface. For example, the command which returns the number of drives is:

- ai_**async**_nav_drive_nb() in the asynchronous interface
- ai_nav_drive_nb() in the synchronous interface

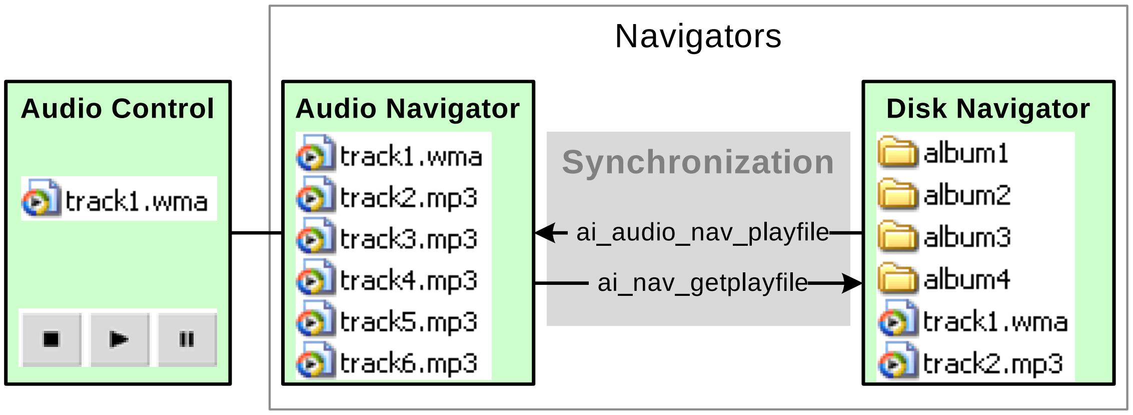
Using asynchronous functions shall be the preferred solution in order to avoid audio underruns.

5.2 Audio interface architecture

The Audio Interface commands managing the audio player are divided into three categories:

• **Disk navigation (Disk Nav)**	to browse into the tree architectures of the mass storage device
• **Audio navigation (Audio Nav)**	to manage a list of playable songs
• **Audio Control (Audio CTRL)**	to control the audio stream (play, pause, fast forward…)

Figure 5-1. Audio player navigators.



5.2.1 Disk Navigator

The “Disk Navigator” is similar to a file explorer and allow browsing into the tree architectures of the mass storage device. It is used to navigate in the tree architectures of the connected mass storage device. Depending on the player options (embedded codecs, palylists support…), it will hide all unwanted files, for example, whose extension differs from *.mp3, *.pls, *.smp or *.m3u…

The “Disk Navigator” is totally independent from the “Audio Navigator”.

The commands associated with this module are used to navigate into the disks/directories and to synchronize the selected file/folder with the “Audio Nav” module. This synchronization is made with these two commands **ai_nav_getplayfile** and **ai_audio_nav_playfile**. See section “Disk navigation” on page 7 for a complete list of commands.

5.2.2 Audio Navigator

The “Audio Navigator” deals with a list of playable song files. This list is defined once a **ai_audio_nav_playfile** command is executed. This command sets the list of files to be played according to the current selection in the “Disk Navigator” and to the “Audio Navigator” **ai_audio_nav_expmode_set** option.

When the **ai_audio_nav_playfile** command is executed:

- if the “Disk Navigator” is currently pointing on a playlist (for example, *.m3u), then only the files of this playlist will be included in the “Audio Navigator” file list
- if the “Disk Navigator” is currently pointing on a disk or a file, then the file list will depend on the audio explorator mode (set by the **ai_audio_nav_expmode_set** command):

Table 5-1. **ai_audio_nav_expmode_set** command behavior.

Explorer mode	Behavior
AUDIO_EXPLORER_MODE_DISKS	Builds a file list off all playable songs of all disks, and start playing from the selected file.
AUDIO_EXPLORER_MODE_DISK	Builds a file list off all playable songs of the current disk only, and start playing from the selected file.
AUDIO_EXPLORER_MODE_DIRONLY	Builds a file list off all playable songs that are contained in the current directory, and start playing from the selected file. Sub directories are ignored.
AUDIO_EXPLORER_MODE_DIRSUB	Builds a file list off all playable songs that are contained in the current directory and its sub-directories, and start playing from the selected file.

- if the “Disk Navigator” is currently pointing on a folder, the audio navigator will not enter into it. It will look for the first file that is in the current directory and build its file list according to the previous rules
- if the “Disk Navigator” is not pointing on any files or folders (which is the case after a mount or a goto), a directory or a playlist is selected, then it will select the first file of the file list

Note that the playing order can be changed at compilation time by enabling a define. This is fully explained in section “Source code architecture” on page 16.

Once the file list is set, two main commands are available to navigate into it:

- **ai_audio_nav_next** (select next file)
- **ai_audio_nav_previous** (select previous file)

Options can also be defined to the “Audio Navigator” (to change the repeat mode, enable/disable the shuffle mode).

To synchronize back the “Disk Navigator” with the “Audio Navigator”, that is, make the “Disk Navigator” selecting the file played by the “Audio Navigator”, the command **ai_nav_getplayfile** can be used. See section “Audio Navigation” on page 10 for a complete list of commands.

5.2.3 Audio control

This module controls the audio stream of the selected file (selected by the “Audio Navigator”). Commands like play, pause, stop, fast forward, fast rewind… are available. See section “Audio control” on page 12 for a complete list of commands.

5.3 Limitations

5.3.1 Speed

Speed navigation (such as file browsing) in directories may be affected if:

- A high-bitrate file is played at the same time
- Directories have many files
- The playlist includes many files

5.4 Disk navigation

The exploration is based on a selector displacement. The **file list** is the list of the files in the current directory according to the extension filter.

The “file list”:

- is updated when entering or exiting a directory or a disk
- starts with the directories then the files
- is sorted in the order it was created

Table 5-2. Disk Navigator commands.

Command name	Input	Output		Description
		Return	Extra result	
ai_get_product_id		product ID		Returns the product identifier (USB PID) of the connected device (if available).
ai_get_vendor_id		vendor ID		Returns the vendor identifier (USB VID) of the connected device (if available).
ai_get_serial_number		serial number length in bytes	serial number	Returns the serial number of the connected device (if available).
ai_nav_drive_nb		drives number		Returns the number of disks available.
ai_nav_drive_set	drive number	true or false		Selects the disk but does not mount it: (0 for drive 0, 1 for drive 1...). Returns false in case of error.
ai_nav_drive_get		drive number		Returns the selected disk number.
ai_nav_drive_mount		true or false		Mounts the selected disk. Returns false in case of error.
ai_nav_drive_total_space		drive capacity		Returns the total space, in bytes, available on the device.
ai_nav_drive_free_space		free drive space		Returns the free space, in bytes, available on the device.
ai_nav_dir_root		true or false		Initializes the file list on the root directory. Return false in case of error.
ai_nav_dir_cd		true or false		Enters in the current directory selected in file list. Return false in case of error.
ai_nav_dir_gotoparent		true or false		Exits current directory and goes to parent directory. Then selects the folder from which it just exits, rather than selecting the first file of the parent directory. This simplifies navigation since the user firmware does not have to memorize this information. Returns false in case of error.
ai_nav_file_isdir		true or false		Returns true if the selected file is a directory, otherwise returns false.
ai_nav_file_next		true or false		Goes to the next file in file list. Returns false in case of error.
ai_nav_file_previous		true or false		Goes to the previous file in file list. Returns false in case of error.
ai_nav_file_goto	file position in list	true or false		Goes to a position in file list (0 for position 0, 1 for position 1...). Returns false in case of error.
ai_nav_file_pos		file position		Returns the file position of the selected file in the current directory. Returns FS_NO_SEL if no file is selected.
ai_nav_file_nb		audio files number		Returns the number of audio files in the file list. Audio files are all the files which extensions match depending on the implemented audio codecs. There is a specific command for playlist files, see below.
ai_nav_dir_nb		directories number		Returns the number of directories in the file list.

Table 5-2. Disk Navigator commands.

Command name	Input	Output		Description
		Return	Extra result	
ai_nav_playlist_nb		playlists number		Returns the number of playlists in the file list. The playlists are all the files matching with the extension *.m3u, *.pls and *.smp.
ai_nav_dir_name		string length in bytes	UNICODE name	Returns the name of the current directory.
ai_nav_file_name		string length in bytes	UNICODE name	Returns the name of the selected file.
ai_nav_file_info_type		file type		Returns the type of the audio file selected.
ai_nav_file_info_version		version number		Returns the version of the metadata storage method used for the selected audio file if available, otherwise, returns 0.
ai_nav_file_info_title		string length in bytes	UNICODE title	Returns the title of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_artist		string length in bytes	UNICODE artist	Returns the artist's name of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_album		string length in bytes	UNICODE album	Returns the album's name of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_year		year		Returns the year of the selected audio file if available, otherwise, returns 0.
ai_nav_file_info_track		string length in bytes	UNICODE info	Returns the track information of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_genre		string length in bytes	UNICODE genre	Returns the genre of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_duration		track duration		Returns the total time of the selected audio file in milliseconds if available, otherwise, returns 0.
ai_nav_file_info_image	max image size	image size	- true of false - a pointer on the image data	Returns information and a pointer on a bitmap of the embedded cover art of the selected audio file. Return false in case of error.
ai_nav_getplayfile		true or false		Selects the file selected by the audio navigator. This command is the only link between these two navigators. Return false in case of error.

5.5 Audio Navigation

This navigator sets the list of files to be played. It can be seen as a «playlist». Before accessing this navigator, an **ai_audio_nav_playfile** command must be issued.

Table 5-3. Audio Navigator commands.

Command name	Input	Output		Description
		Return	Extra result	
ai_audio_nav_playfile		true or false		This command sets the audio file list according to the current mode of the audio navigator and plays (or sets to pause) the file selected in the disk navigator. In other words, it synchronizes the audio navigator with the disk navigator and plays (or sets to pause) the selected file from the disk navigator. Note that this command returns immediately and does not wait until the current track is completely played. This commands does not change the current options (repeat, random and expmode). It is the opposite of the command **ai_nav_getplayfile**. Return false in case of error.
ai_audio_context_save		context structure	- true or false - structure length in bytes	Gives complete audio context (player state, play time, repeat, random, file played, explorer mode).
ai_audio_context_restore	context structure	true or false		Restores an audio context (eventually restart playing).
ai_audio_nav_next		true or false		Jump to next audio file available in the list. The next song file is chosen according to the current options (repeat, random and mode).
ai_audio_nav_previous		true or false		Jumps to previous audio file available in the list. The next song file is chosen according to the current options (repeat, random and mode).
ai_audio_nav_eof_occur		true or false		This routine must be called once a track ends. It will choose, according to the options (repeat, random and mode), the next file to play.
ai_audio_nav_nb		audio files number present in the list		Returns the number of audio files present in the list.
ai_audio_nav_getpos		file position		Returns the file position of the selected file in the list.
ai_audio_nav_setpos	file position	true or false		Goes to a position in the list.
ai_audio_nav_repeat_get		ai_repeat_mode		Returns the current repeat mode (no repeat, repeat single, repeat folder, repeat all).
ai_audio_nav_repeat_set	ai_repeat_mode			Sets the current repeat mode (no repeat, repeat single, repeat folder, repeat all).
ai_audio_nav_shuffle_get		ai_shuffle_mode		Returns the current shuffle mode (no shuffle, shuffle folder, shuffle all).
ai_audio_nav_shuffle_set	ai_shuffle_mode			Sets the current shuffle mode (no shuffle, shuffle folder, shuffle all).
ai_audio_nav_expmode_get		ai_explorer_mode		Returns the current explorer mode (all disks, one disk, directory only, directory + sub directories).
ai_audio_nav_expmode_set	ai_explorer_mode			Sets the current explorer mode (all disks, one disk, directory only, directory + sub directories). This mode cannot be changed while an audio file is being played.

Table 5-3. Audio Navigator commands.

Command name	Input	Output		Description
		Return	Extra result	
ai_audio_nav_getname		string length in bytes	UNICODE name	Returns the name of selected file.
ai_audio_nav_file_info_type		file type		Returns the type of the audio file selected.
ai_audio_nav_file_info_version		version number		Returns the version of the metadata storage method used for the selected audio file if available, otherwise, returns 0.
ai_audio_nav_file_info_title		string length in bytes	UNICODE title	Returns the title of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_file_info_artist		String length in bytes	UNICODE artist	Returns the artist's name of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_file_info_album		string length in bytes	UNICODE album	Returns the album's name of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_file_info_year		year		Returns the year of the selected audio file if available, otherwise, returns 0.
ai_audio_nav_file_info_track		string length in bytes	UNICODE info	Returns the track information of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_file_info_genre		string length in bytes	UNICODE genre	Returns the genre of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_file_info_duration		track duration		Returns the total time of the selected audio file in milliseconds if available, otherwise, returns 0.
ai_audio_nav_file_info_image	max image size	image size	- true of false - a pointer on the image data	Returns information and a pointer on a bitmap of the embedded cover art of the selected audio file. Returns false in case of error.

5.5.1 Modes used by the interface

5.5.1.1 ai_repeat_mode

Defines the repeat modes:

- AUDIO_REPEAT_OFF: no repeat
- AUDIO_REPEAT_TRACK: repeat the current track
- AUDIO_REPEAT_FOLDER: repeat the current folder
- AUDIO_REPEAT_ALL: repeat the list of files

5.5.1.2 ai_shuffle_mode

Defines the shuffle playing mode:

- AUDIO_SHUFFLE_OFF: no shuffle
- AUDIO_SHUFFLE_FOLDER: shuffle into the current folder
- AUDIO_SHUFFLE_ALL: shuffle into the list of files

5.5.1.3 ai_explorer_mode

Defines the explorer mode (see section "Source code architecture" on page 16 for more information.

5.6 Audio control

The audio controller is used to control the audio stream of the audio file selected by the audio navigator.

Table 5-4. Audio control commands.

Command name	Input	Output		Description
		Return	Extra result	
ai_audio_ctrl_stop		true or false		Stops the audio.
ai_audio_ctrl_resume		true or false		Plays or resumes play after an **ai_audio_ctrl_pause** or an **ai_audio_ctrl_stop** command.
ai_audio_ctrl_pause		true or false		Pauses the audio.
ai_audio_ctrl_time		elapsed time		Returns the elapsed time of the audio track being played.
ai_audio_ctrl_status		status		Returns the status of the audio controller (stop, play, pause, a new audio file is being played, the current folder has changed).
ai_audio_ctrl_ffw	skip time	true or false		Fast forwards the audio until the skip time has been reached. Then, it will continue to play the rest of the audio file. The skip time passed in parameter is in second.
ai_audio_ctrl_frw	skip time	true or false		Fast rewinds the audio until the skip time has been reached. Then, it will set the audio player in play mode. The skip time passed in parameter is in second.
ai_audio_ctrl_start_ffw		true or false		Sets the audio player in fast forward mode. ***Function not implemented yet.***
ai_audio_ctrl_start_frw		true or false		Sets the audio player in fast rewind mode. ***Function not implemented yet.***
ai_audio_ctrl_stop_ffw_frw		true or false		Stops fast forwarding, rewinding and set the audio player in the previous mode (play or pause). ***Function not implemented yet.***

5.7 Using asynchronous or synchronous API

Using synchronous function is straightforward. Once finished, synchronous functions returns, with or without a result.

Using asynchronous function is more complicated: they may not produce the requested task in a single shot. Thus these functions need specific APIs to properly operate:

- `void ai_async_cmd_task(void)`:

  This function is the entry point of all asynchronous commands. It must be called to execute the current state of the internal state machine of the current asynchronous function.

- `bool is_ai_async_cmd_finished(void)`:

  This function returns true if the last command is finished.

- `uint8_t ai_async_cmd_out_status(void)`:

  Returns the status of the last executed command (CMD_DONE, CMD_EXECUTING or CMD_ERROR).

- `uint32_t ai_async_cmd_out_u32(void)`:

  If the last executed command should return a 32-bit result or less, this function will return this value.

- `uint64_t ai_async_cmd_out_u64(void)`:

  If the last executed command should return a 64-bit result, this function will return this value.

- `uint16_t ai_async_cmd_out_SizeArrayU8(void)`:

  If the last executed command should return an extra result (for example, a song name), this function returns the size in bytes of the extra result (no size limit).

- `char* ai_async_cmd_out_PtrArrayU8(void)`:

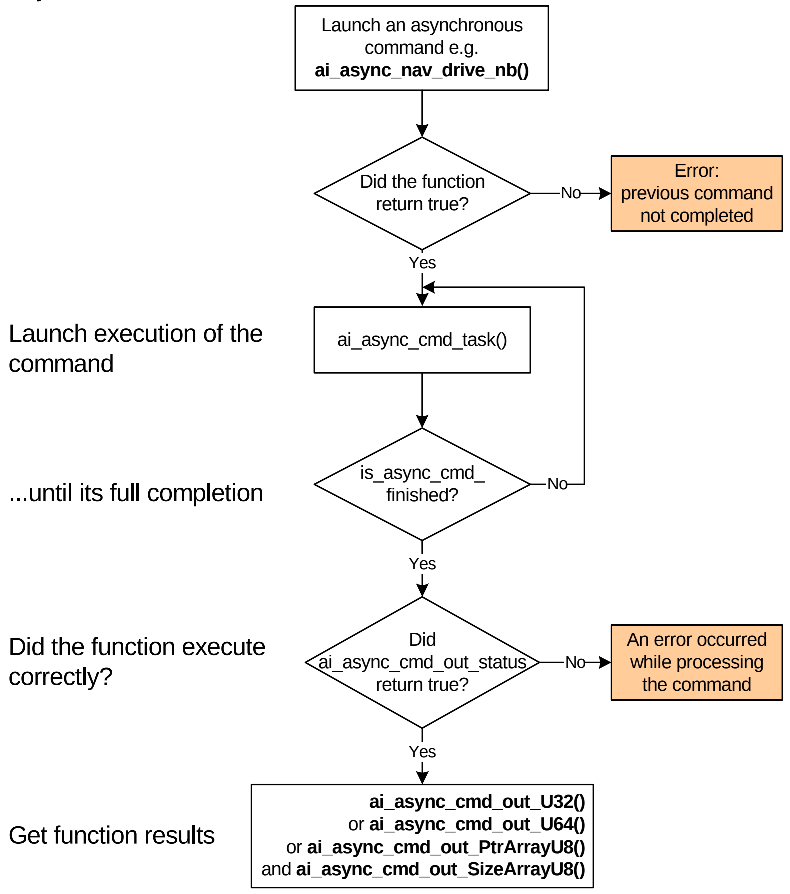
  Returns a pointer to the extra result (assuming that the last executed command returns an extra result). This pointer can be freed by the application with the ai_async_cmd_out_free_ArrayU8() function.

- `void ai_async_cmd_out_free_ArrayU8(void)`:

  This function may be called to free the allocated buffer which holds the extra-result. Note that the Audio Interface will automatically do this before executing a new command that need extra-results. This ensures that the application will not have memory leakage. Allowing the application calling this function will free the extra-results sooner and improve allocated memory usage.

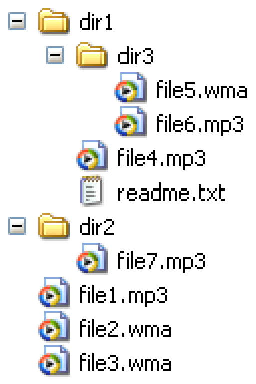
Figure 5-2 shows the flow of any asynchronous function usage.

Figure 5-2. Asynchronous function flow.


Launch an asynchronous command e.g. ai_async_nav_drive_nb()
Did the function return true?
No
Error: previous command not completed
Yes
Launch execution of the command
ai_async_cmd_task()
...until its full completion
is_async_cmd_finished?
No
Yes
Did the function execute correctly?
Did ai_async_cmd_out_status return true?
No
An error occurred while processing the command
Yes
Get function results
ai_async_cmd_out_U32() or ai_async_cmd_out_U64() or ai_async_cmd_out_PtrArrayU8() and ai_async_cmd_out_SizeArrayU8()

5.8 Examples

The following examples are using a disk with the following contents:



Let's take this disk as disk number 0 for the system.

5.8.1 Example 1 - Play "file1.mp3"

Table 5-5. Example: Play "file1.mp3".

Sequence step	Command name
0	**ai_nav_drive_nb**(): returns **1** disk
1	**ai_nav_drive_set**(0): selects the disk 0
2	**ai_nav_drive_mount**(): mounts the select disk 0
3	**ai_nav_file_goto**(0): goes to file position 0
4	**ai_nav_file_name**(): returns the name dir1
5	**ai_nav_file_isdir**(): returns true, the current file is a directory
6	**ai_nav_file_goto**(1): goes to file position 1
7	**ai_nav_file_name**(): returns the name dir2
8	**ai_nav_file_goto**(2): goes to file position 2
9	**ai_nav_file_name**(): returns the name file1.mp3
10	**ai_audio_nav_playfile**(): selects the file selected by the file navigator (file1.mp3)
11	**ai_audio_ctrl_resume**(): plays the selected file
12	*Wait for 10 seconds to listen to the beginning of the playback*
13	**ai_nav_file_goto**(3): goes to file position 3
14	**ai_nav_file_name**(): returns the name file2.wma

5.8.2 Example 2 - Browse while playing

Table 5-6. Example: Browse while playing.

Sequence step	Command name
0	**ai_nav_drive_nb**(): returns **1** disk
1	**ai_nav_drive_set**(0): selects the disk 0
2	**ai_nav_drive_mount**(): mounts the select disk 0
3	**ai_audio_nav_playfile**(): selects a file in the audio navigator. By default it will seek inside the directories to play the first file which is "file5.wma" in our case

Sequence step	Command name
4	**ai_audio_ctrl_resume**(): plays the selected file
5	**ai_nav_getplayfile**(): synchronizes the disk navigator with the audio navigator. Now the disk navigator is pointing on the “file5.wma” file
6	**ai_nav_file_nb**() + **ai_nav_dir_nb**() + **ai_nav_playlist_nb**(): gets the total number of entries (files + folder + playlist) in the current directory (dir3)
7	**ai_nav_file_goto**(0): goes to file position 0
8	**ai_nav_file_name**(): returns the name “file5.wma“. (Notice the difference with Example 1 - step #4)
9	**ai_nav_file_goto**(1): goes to file position 1
10	**ai_nav_file_name**(): returns the name “file6.mp3“
11	**ai_audio_ctrl_stop**(): stops the audio

6. Source code architecture

6.1 Package

The **audio_player-<board(s)>-<feature(s)>-<version>.zip** file contains projects for GCC, Atmel AVR Studio 5 and IAR™.

Default hardware configuration of the project is to run on Atmel AVR UC3 evaluation kits, although any board can be used as detailed in section "Board definition files" on page 18.

6.2 Documentation

For full source code documentation, please refer to the auto-generated Doxygen source code documentation found in:

– /avr32/applications/uc3-audio-player/readme.html

6.3 Projects / compiler

The **IAR** project is located here:

– /avr32/applications/uc3-audio-player/<part>_<board>_<feature(s)>/iar/

The **GCC** makefile is located here:

– /avr32/applications/uc3-audio-player/<part>_<board>_<feature(s)>/gcc/

The **Atmel AVR Studio 5** project solution package is located here.

– /avr32/applications/uc3-audio-player/<part>_<board>_<feature(s)>/as5_32/

6.4 Implementation details

The following sections describe the code implementation of the audio player running on the Atmel EVK1105. Other available packages are similar. Following information applies to all project configurations.

6.4.1 Main()

The main() function of the program is located in the file:

– /avr32/applications/uc3-audio-player/main.c

This function performs the following actions:

– Audio output initialization
  Refer to section "Audio rendering interface" on page 19.

– Clock configuration

– USB task call
  Refer to section "USB" on page 18.

– USB host mass-storage task call

– Communication task (HMI) call
  Refer to section "HMI communication task example" on page 17.

– Decoder tasks call
  Refer to section "Audio decoder" on page 17.

The /avr32/applications/uc3-audio-player/ contains other files:

- ./com_task.[c,h]: HMI core. Refer to Section "HMI communication task example" for more details.
  - ./user_interface/controller/joystick_controller.c: HMI using a joystick interface as a controller
  - ./user_interface/gui/et024006dhu_gui.c: HMI using a LCD screen as a display
- ./codec_<feature>_task.c: handle the audio codec decoding task
- ./conf/*.h: configuration files for audio, communication interface, memory and navigation explorer. Refer to section "Project configuration" on page 22 for more information on the configuration files

6.4.2 Audio decoder

The audio decoders are located in /thirdparty folder. Audio decoders are detailed in the companion application notes as referred to in section "Overview" on page 2.

6.4.3 Audio tags

ID3 is supported up to version 2.4. The ID3 reader source is located in /avr32/services/audio/mp3/id3.

6.4.4 Audio player API

The audio interface API is located in:

- /avr32/services/audio/audio_player/audio_interface.[c,h] files

Refer to section "Audio interface API" on page 5 for more details.

6.4.4.1 Mass storage audio interface

The mass storage audio interface can be found in /avr32/services/audio/audio_player/ai_usb_ms

- ./ai_usb_ms.[c,h]: mass storage audio interface
- ./ai_usb_ms_mp3_support.[c,h]: add support to the MP3 file format in the audio interface
- ./host_mass_storage_task.[c,h]: USB host mass storage task

6.4.4.2 SD card audio interface

The SD card audio interface can be found in /avr32/services/audio/audio_player/ai_sd_mmc

- ./ai_sd_mmc.c: SD card audio interface

6.4.5 HMI communication task example

The firmware implements an HMI example using a joystick and a LCD. The source code is located in the /avr32/applications/uc3-audio-player/ folder.

All the HMI is based on a pair of files, com_task.[c,h], which implements all mechanisms used to communicate between the internal APIs of the audio player and the user's HMI. An abstraction layer is used to easily attach all kinds of controller and graphical user interface to this communication task. It has been done to easily port the application to another board.

6.4.5.1 *Controller*

The controller is the module that makes the link between the driver and the abstraction layer used to ensure compatibility with the communication task. It is based on mainly two API groups: 'setup' and 'events'. All the APIs are defined in the file /avr32/applications/uc3-audio-player/user_interface/controller/controller.h.

The function *controller_init* is used to initialize the controller. It may be used or not, depending on the controller implementation.

All the other functions are boolean functions, returning **true** if an event has been raised or **false** otherwise. Their name is explicit and follows the following naming convention:

```
bool controller_<view>_<event>(void);
```

Where *<view>* corresponds to the current view when the event should be taken into account. If no value is set to this field, the event applies to all views.

The *<event>* is a short name describing the current event which the function applies to.

This example uses a joystick controller. The source code relative to this module is available in the /joystick_controller.c file.

6.4.5.2 *Graphical user interface*

This module is used to display to the user the high level audio player data. It acts as a display and when combined with the controller it provides full control of the application to the user.

This module is similar to the controller in terms of API groups.

The initialization function is called *gui_init* and takes into parameters all the system frequencies used to initialize the display.

The 'event' group (to keep the same architecture as the controller), is composed of only one function: *gui_update*. This function is called at every occurrence of the main loop and takes into parameters flags, describing which part of the view has to be updated. The resetting of these flags is let to the GUI module so that it can achieve the update of the display asynchronously and does not have to update every component at once.

The example provides a graphical LCD display module that implements both a navigation and a playback view. All the code is based on the files /et024006dhu_gui.c and /sdram_loader.c.

6.4.6 Atmel AT32UC3A drivers

The source code of the Atmel AVR32 UC3 drivers can be found in the /avr32/drivers/ folder.

6.4.7 USB

The USB low level driver is located in /avr32/drivers/usbb/_asf_v1/ folder.

The USB mass storage service is located in /avr32/services/usb/_asf_v1/class/mass_storage/ folder.

6.4.8 FAT file system

The FAT12/16/32 source code is located in /avr32/services/fs/fat/ folder.

6.4.9 Board definition files

The application is designed to run on Atmel evaluation kits. All projects are configured with the following define: BOARD=EVKxxxx. The EVKxxxx definition can be found in the /avr32/boards/evkxxxx/ folder.

6.4.9.1 Board customizing

- For **IAR** project, open the project options (Project -> Options), choose the "C/C++ Compiler", then "Preprocessor". Modify the 'BOARD=EVKxxxx' definition by 'BOARD=USER_BOARD'
- For **GCC**, open the 'config.mk' file (/avr32/applications/uc3-audio-player/<part>-<board>-<feature(s)>/gcc/) and change the DEFS definition '-D BOARD=EVKxxxx' by '-D BOARD=USER_BOARD'
- For **Atmel AVR Studio 5**, open the project properties (Project -> …Properties), select the "Toolchain" tab, then "Symbols" under "AVR32/GNU C Compiler" item. Modify the 'BOARD=EVKxxxx' definition by 'BOARD=USER_BOARD'

The HMI can be easily modified. Refer to section "HMI communication task example" on page 17 for more details.

6.4.10 Audio rendering interface

The audio DAC mixer source code is located in /avr32/services/audio/audio_mixer/audio_mixer.[c,h] files.

6.4.10.1 External audio DAC

The /avr32/components/audio/codec/tlv320aic23b/ folder contains the driver for the external DAC TLV320AIC23B.

The UC3 controls the TLV320AIC23B through a two-wire interface (TWI) in master mode.

The Atmel AVR UC3 SSC module generates I^{2}S frames using internal DMA (PDCA) to free up CPU cycles for audio decoding.

Each time a new song is played, the SSC module is configured corresponding to the sample rate of the new song. The SSC clocks are composed of a bit clock and a frame sync:

- The bit clock is the clock used to transmit a bit from the audio stream. For a 44.1KHz sample rate, the bit clock will be 44100 x 2 (stereo) x 16 (bits per channel), that is, 1.411MHz
- The Frame sync is equal to the sample rate frequency, that is, 44.1KHz taking the same example

To get accurate 44.1KHz audio frequency samples, an external 11.2896MHz oscillator is used as input to an internal PLL and source the CPU/HSB/PBA/PBB frequency with 62.0928MHz.

The TLV320AIC23B uses a master clock (MCLK) of 11.2896MHz, output by the UC3 through a generic clock. Then, the generic clock output (PA07) is connected to the MCLK of the TLV320AIC23B.

The SSC clock divider in CMR register is given by:

SSC.CMR.DIV = Round (F_{PBA} / (2 x ($F_{SampleRateSetPoint}$ x NumberChannel x BitsPerSamples))

The real frequency sample rate output by the SSC is given by:

$F_{ActualSampleRate}$= F_{PBA} / (2 x SSC.CMR.DIV x NumberChannel x BitsPerSamples)

Note: Note: NumberChannel = 2, BitsPerSamples = 16, F_{PBA} = 62.0928 MHz

The music interval in semitones is:

MusicInterval (semitones) = LOG(($F_{ActualSampleRate}$ / $F_{SampleRateSetPoint}$) / (2 $^{1/12}$))

Table 6-1. Samples rate with SSC module.

Sample rate set point [Hz]	Actual sample rate [Hz]	Relative Error [%]	Music Interval [semitones]
8000	8018	0.23	0.04
11025	11025	0.00	0.00
16000	15095	-0.59	-0.10
22050	22050	0.00	0.00
32000	32340	1.06	0.18
44100	44100	0.00	0.00
48000	48510	1.06	0.18

For further information, please refer to the AVR32788: AVR32 How to use the SSC in I^{2}S mode application note.

6.4.10.2 *Internal audio DAC*

The ABDAC driver is located in /drivers/abdac.

The external amplifier driver (TPA6130A2) is located in /components/audio/amp/tpa6130a2.

The Atmel AVR32 ABDAC module is using the internal DMA (PDCA) to free CPU cycles for audio decoding.

To get accurate audio frequency samples, the two external oscillators 12MHz (OSC1) and 11.2896MHz (OSC0) are used to source (directly or through a PLL) the ABDAC generic clock.

The PLL0 output frequency is 62.0928MHz. The PLL1 output frequency is 48MHz (used for USB).

When used, the ABDAC generic clock divider is given by:

ABDAC generic clock divider = Round ($F_{GCLKInput}$ / (2 x 256 x ($F_{SampleRateSetPoint}$)) -1).

Refer to /DRIVERS/ABDAC/abdac.c for the configuration of the ABDAC generic clock.

Table 6-2. Samples rate with ABDAC module.

Sample rate set point [Hz]	ABDAC generic clock input frequency	Actual sample rate [Hz]	Relative error [%]	Music interval [semitones]
8000	PLL0	8085	1.06	0.18
11025	OSC1	11025	0.00	0.00
16000	PLL1	15625	-2.34	-0.41
22050	OSC1	22050	0.00	0.00
32000	PLL1	31250	-2.34	-0.41
44100	OSC1	44100	0.00	0.00
48000	PLL1	46875	-2.34	-0.41

6.4.11 SDRAM loader

Graphical data like images consume a lot of space in RAM and flash which could be used by the application instead. Because of that, the images used by the audio player application GUI are stored in Atmel DataFlash® and are loaded to SDRAM upon startup. The reason for having the data in the SDRAM, rather than in flash, is the speed improvement that leads to faster updates of the display with new content.

The SDRAM loader module uses a memory manager to manage the SDRAM memory space. This memory manager can also be used in the application to allocate memory from SDRAM instead of the internal SRAM.

The following sections give a short introduction to the parts involved in the SDRAM loader module.

The SDRAM loader source code is located in /avr32/applications/uc3-audio-player/user_interface/gui/sdram_loader.[c,h] files.

6.4.11.1 *Atmel DataFlash*

The DataFlash memory is formatted with a FAT16 file system and currently contains graphical data that is used by the audio player application in the GUI implementation. The graphical data is stored as BMP files and the used format is RGB565 which can be used directly on the display when swapped from little endian to big endian. Other BMP formats are not supported.

The FAT file system makes it easy for developers to upgrade the content with new files or even use it for other applications like a web-server. An upgrade of the content can be done by using a mass storage example application.

The BMP pictures used in the application are stored in the /avr32/applications/uc3-audio-player/pictures/ folder.

To customize the GUI, the bitmap files can be updated and must be saved to the “Windows bitmap image” format using a “16-bit R5 G6 B5” encoding. This can be done using “GIMP”, the GNU Image Manipulation Program (see http://www.gimp.org).

6.4.11.2 *Loading process*

The SDRAM loader consists of two files, sdram_loader.c and sdram_loader.h, in the /avr32/applications/uc3-audio-player/user_interface/gui/ folder. In these files the images are specified that should be loaded to SDRAM and how they should be converted. A configuration that loads three images to SDRAM could look like this:

```
typedef struct {
  const wchar_t *name;
  void *start_addr;
} ram_file_t;
…
#define STARTUP_IMAGE       0
#define DISK_NAV_IMAGE      1
#define AUDIO_PLAYER_IMAGE  2
#define NUMBER_OF_IMAGES    13

ram_file_t ram_files[NUMBER_OF_IMAGES] = {
        { .name = L"/AVR32_start_320x240_RGB565.bmp"},
        { .name = L"/disk_nav_320x240_RGB565.bmp"},
        { .name = L"/audio_player.bmp"}
};
```

The ram_files array is used throughout the GUI to get access to the image data. In order to load other data than RGB565 BMP data to SDRAM the module needs to be modified.

The SDRAM loader module is called once during the initialization of the graphical user interface. When called it initializes the SDRAM interface and reads the raw image data from specified BMP files into SDRAM. To get the raw image data the BMP header must be read to get the image size and the offset of the data in the file. The copy process does also a conversion from the little endian to the big endian data ordering and because of that the final image data can be dumped directly into the display buffer.

A single call to the SDRAM module is enough to do the initialization and load process:

```
void load_sdram_data(int hsb_hz);
```

The sole parameter is the HSB frequency in hertz which is needed to initialize the SDRAM timings. The above function is called starting from main in the following order:

```
main() -> com_task() -> gui_init() -> sdram_load_data()
```

6.4.11.3 SDRAM memory management

The images could be placed at specified locations in SDRAM and thus would make a memory management unneeded, but on the other hand it is often better to do memory management to remove the task of keeping track of the data in memory from the developer.

The audio player uses a separate memory manager to manage the SDRAM memory (this memory manager can also replace the default memory manager in the Newlib library if needed).

The source files of the memory manager are located in /thirdparty/dlmalloc/ folder while the configuration is in /config/conf_dlmalloc.h file.

The memory manager is initialized in the SDRAM loader module and it is configured to use the whole SDRAM memory. After the initialization memory can be allocated from the SDRAM with the mspace_malloc call. Memory from the internal SRAM can be allocated with the default malloc call.

6.4.12 JPEG decoder

The JPEG decoder is used for the audio cover art file support.

The JPEG decoder library is located in /thirdparty/ijg/libs/ folder while the configuration file is in /config/conf_jpeg_decoder.h file. This is a layer that handles the data input and output of the JPEG library. It also handles the error handling to jump out of the library in case of a critical error. Since the JPEG decoding is done in the SDRAM, the memory management back-end of the library is located here too, which normally would be included in the library itself. The memory management back-end is modified to use the SDRAM with the DLMALLOC memory manager (see section "SDRAM memory management") instead of the standard malloc implementation.

6.5 Project configuration

The configuration files are located in the /config/ folder.

There are two different configuration files. The one related to the audio player itself: conf_audio_player.h, is mostly a high level configuration file and can be customized to support certain configurations or enable/disable audio player features. The rest of the configuration files should be modified to change, separately, audio player's module configurations.

Configuration files are not linked to ‘IAR’, ‘GCC’ or ‘AVR Studio 5’ projects. The user can alter any of them, then rebuild the entire project in order to reflect the new configuration.

6.5.1 High level configuration file: conf_audio_player.h.

6.5.1.1 Audio DAC selection

- **DEFAULT_DACS**, specifies the default audio DAC used for the audio output. Three values are possible: AUDIO_MIXER_DAC_AIC23B for the I^2S interface, AUDIO_MIXER_DAC_ABDAC for the internal DAC (ABDAC module), and AUDIO_MIXER_DAC_PWM_DAC to use PWM channels (external low-pass filter is required)

6.5.1.2 Sampling bufferization

- **USE_AUDIO_PLAYER_BUFFERIZATION**, set to ‘true’ to support navigation while playing feature. This uses extra memory (internal or external) for buffering decoded samples in order to prevent audio blips while the user navigates in the disk architecture for example. The memory address and size can be configured in the conf_buff_player.h file as described below. *Note that using the audio sample bufferization will not ensure that every audio blip will be covered. It will always depends on the speed of the mass storage device connected, its file system, the requested operations. The default buffer size is set to 128KB*

6.5.1.3 Features implementation

A set of defines that can be modified to enable or disable audio player features.

- **SUPPORT_<CODEC>**, set to ‘true’ to support the corresponding audio codec. Note that at least one codec must be enabled and that the number of supported codec depends on the UC3 Flash and RAM memory size
- **SUPPORT_PLAYLISTS**, set to ‘true’ to support playlists
- **SUPPORT_EMBEDDED_COVER_ARTS**, set to ‘true’ to support cover art display. Note that the use of the embedded cover arts requires RAM in order to decode embedded JPEG pictures. Therefore it needs an external memory (SRAM or SDRAM) to handle this feature

6.5.2 Low level configuration files: /config/*.h.

6.5.2.1 conf_access.h

This file contains the possible external configuration of the memory access control. It configures the abstract layer between the memory and the application and specifies the commands used in order to access the memory. For example, this file will define the functions to be called for a SD/MMC memory access.

6.5.2.2 conf_audio_interface.h

A set of configuration flags to enable/disable internal features of the audio player.

6.5.2.3 conf_audio_mixer.h

Configures all parameters relative to the audio DACs. This file is made to support multiple configurations and can be easily upgraded to handle new DACs.

6.5.2.4 conf_buff_player.h

Defines the starting address and the size of the memory used to buffer audio samples (if the feature is enabled).

6.5.2.5 conf_dmalloc.h

Configuration of malloc/free functions.

6.5.2.6 conf_explorer.h

It defines the configuration used by the FAT file system. The configuration is also applied to the playlist handler and the file navigation. The main parameters are:

- **NAV_AUTO_FILE_IN_FIRST**, must be define in order to select and play files of a folder before the subdirectories
- **FS_NAV_AUTOMATIC_NBFILE**, this flag can be set to DISABLE in order to speed up the response of the ai_audio_nav_playfile command. On the other hand, the three commands ai_audio_nav_getpos, ai_audio_nav_getpos and ai_audio_nav_nb will not be available anymore. It will also affect the use of the explorer modes, if different from “all disks” and “one disk”

6.5.2.7 conf_jpeg_decoder

Configuration of the JPEG decoder.

6.5.2.8 conf_pwm_dac.h

Configuration of the PWM DAC (which PWM channel is used, which pins are concerned).

6.5.2.9 conf_tlv320aic23b.h

Configuration of the external I^{2}S DAC (which pins are used and which configuration interface).

6.5.2.10 conf_usb.h

Configuration file used for the USB.

6.5.2.11 conf_version.h

Internal version of the firmware.

7. Module licenses

Following modules requires end-user licensing.

7.1 Memory manager

The memory manager (dlmalloc) license is located in /thirdparty/dlmalloc/license.txt.

7.2 JPEG decoder

The IJG JPEG decoder license is located in /thirdparty/ijg/license.txt.

7.3 Audio decoders

Audio decoder licenses are detailed in the companion application notes as referred to in section "Overview" on page 2.

8. FAQ

Q: What is the maximum number of playlist links supported?

A: The file system supports up to 65535 links inside a playlist.

Q: What are the supported text formats?

A: The file system supports the ASCII, UTF8 and UNICODE (UTF16LE & UTF16BE) text formats.

Q: How are the directories and files sorted inside the disk?

A: The logical structure is the same as an explorer view. Directories and files are sorted using their creation order.

Q: Which file systems are supported?

A: FAT 12/16/32.

Q: What is the maximum number of files supported in a directory?

A: There is no limitation in the firmware for the supported number of files and directories. The only limitation is due to the FAT file system:

- for FAT12/16 root directory only: up to 256 files (short names),
- for FAT12/16/32 up to 65535 files (short names) per directory.

Q: What is the minimum RAM requirement to run this application?

A: The default application is using external SDRAM to support all features. This application can run with 64K of RAM (internal size on the Atmel AT32UC3A0512) by disabling the audio buffer and the cover art support. Audio blips will be present if the user tries to navigate in the disk while a track is played. The SDRAM loader also uses external SDRAM memory for graphical data.

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
16F, Shin Osaki Kangyo Bldg.
1-6-4 Osaki Shinagawa-ku
Tokyo 104-0032
JAPAN
Tel: (+81)(3) 3-6417-0300
Fax: (+81)(3) 3-6417-0370