AVR32709: AVR32 UC3 Audio Decoder Over USB on AT32UC3A0512 or AT32UC3A1512

32-bit AVR® Microcontroller

Application Note

Features

- **Software MP3 Decoder**
- **FAT File System**
- **Music played over USB (USB Host mass storage class)**
- **Standalone - Low Memory Footprint (Code & RAM)**
- **Audio output over I2S using SSC controller**
- **Local Control with Joystick**

1. Introduction

This application note will help the reader to use the AVR32® UC3 Audio Decoder over USB software. This software includes a software MP3 decoder, a file system, and a USB Host mass storage class support.

The solution presented here in this application is based on Atmel AVR32 UC3A0/A1 microcontroller.

For more information about the AVR32 architecture, please refer to the appropriate documents available from http://www.atmel.com/avr32.

2. License

The MP3 decoder MAD is distributed under the terms of the GPL. Its redistribution is not generally restricted, as long as the terms of the GPL are followed. This means MAD can be incorporated into other software as long as that software is also distributed under the GPL. (Should this be undesirable, alternate arrangements may be possible by contacting Underbit.)

For more information about codec licensing, please read the application note AVR32722: How to licence audio and video codecs.

3. Requirements

The software provided with this application note requires several components:

- A computer running Microsoft® Windows® 2000/XP/Vista or Linux®
- AVR32Studio and the GNU toolchain (GCC) or IAR Embedded Workbench for AVR32 compiler.
- A JTAGICE mkII or AVROne! debugger

7817A–AVR32–08/08

4. Theory of Operation

4.1 Overview

Today, embedded MP3 decoders are everywhere for consumers listening to audio content on mobile devices.

4.1.1 MP3

MPEG-1 Audio Layer 3, more commonly referred to as MP3, is a digital audio encoding format using a form of lossy data compression. Several bit rates are specified in the MPEG-1 Layer 3 standard: 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256 and 320 Kbit/s, and the available sampling frequencies are 32, 44.1 and 48 KHz. A sample rate of 44.1 KHz is almost always used. 128 Kbit/s bitrate files are slowly being replaced with higher bitrates like 192 Kbit/s, with some being encoded up to MP3's maximum of 320 Kbit/s.

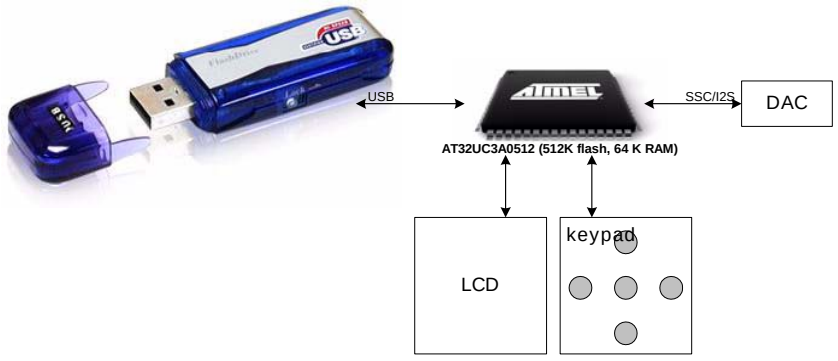
A tag in a compressed audio file is a section of the file that contains metadata such as the title, artist, album, track number or other information about the file's contents.

The chosen MP3 decoder here is MAD (libmad), a high-quality MPEG audio decoder. It currently supports MPEG-1 and the MPEG-2 extension to Lower Sampling Frequencies, as well as the so-called MPEG 2.5 format. All three audio layers (Layer I, Layer II, and Layer III a.k.a. MP3) are fully implemented. MAD does not yet support MPEG-2 multichannel audio (although it should be backward compatible with such streams).

4.2 Block diagram

The following block diagram describes the UC3 interfacing the USB stick and output the audio stream from the key to the external DAC. The user can control the player using a keypad, running a customisable Human-Machine Interface (HMI).

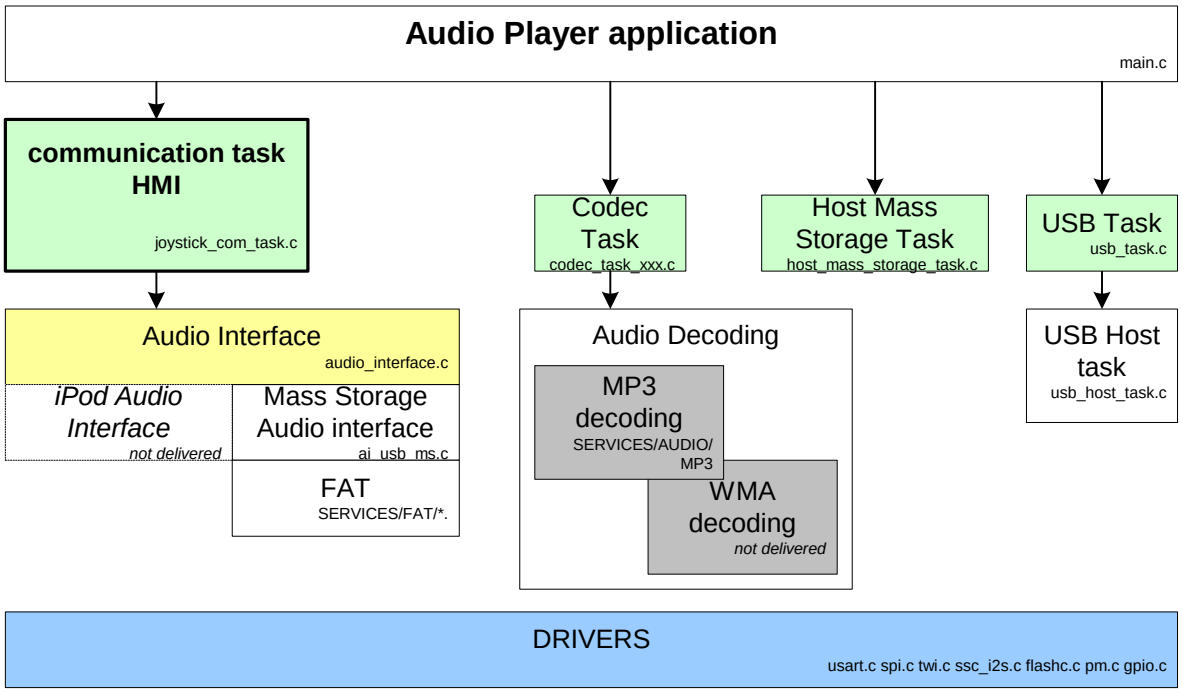
Figure 4-1. Block diagram



4.3 Software architecture

The following figure shows the basic software architecture of the application:

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. There are 4 tasks:

- The communication task (shown in bold on the graph) 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 Codec Task: This task is in charge of the Audio Decoding management.
- The USB task: This task handles the USB stack and events.
- The Host Mass Storage task: This task will check for new devices connection and initialize them using the USB Mass Storage class.

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

```
while (TRUE)
  {
    usb_task();
    host_mass_storage_task();
    com_task();
    codec_task();
  }
```

Note that the Audio Interface can also support iPod® audio decoding through the USB Audio class. Please refer to the «AVR32730: AVR32 UC3 iPod Support Over USB on AT32UC3A0512» application note for more information.

5. Audio Interface API

In the following, we will consider that the audio player API can support multiple formats like WMA or MP3.

5.1 Overview

The Audio Interface (AI) manages disk navigation, audio navigation and audio control (see below). 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 2 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.
- 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.

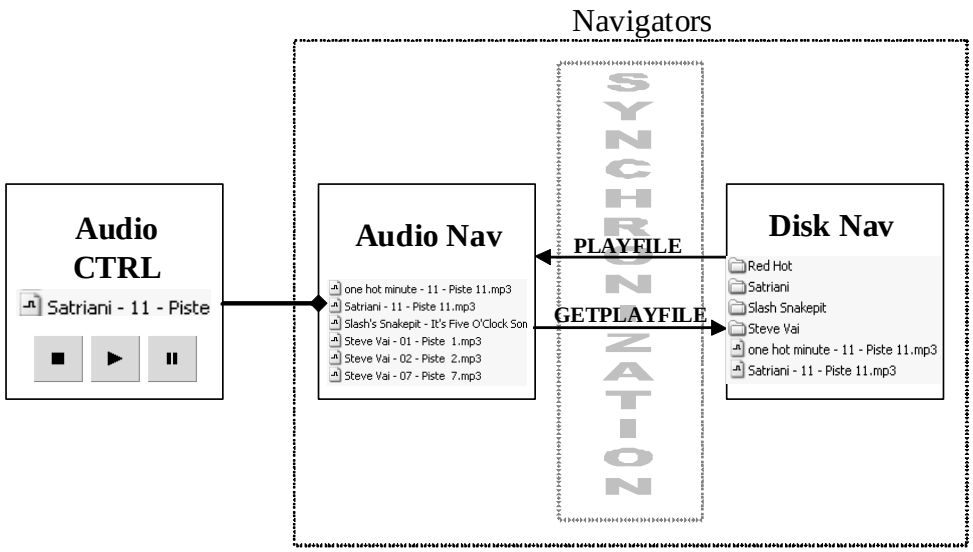
From now on, until the end of the document, we will always use the synchronous name of the AI functions.

5.2 Audio Interface Architecture

The AI commands to interface the audio player are divided into three categories:

- **Disk navigation (Disk Nav)** to browse into the tree architectures of the USB 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 Overview



5.2.1 Disk Navigator: “browse into the tree architectures of the mass storage device”

The “Disk Navigator” is similar to a file explorer. It is used to navigate in the tree architectures of the connected USB device. It will hide all files whose extension differs from *.mp3, *.wma or *.m3u. Note that the file extension filtering is selectable: the user tells to the Audio Interface which kind of files it shall manage. Thus, new file format support (e.g. WAV, AAC, or other) is open.

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 “Disk Navigation” on page 8. for a complete list of commands.

5.2.2 Audio Navigator: “manage a list of playable songs”

The “Audio Navigator” deals with a list of 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 (*.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 5.9.3 “Example 3 - Change the playing order” on page 15.

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

- **ai_audio_nav_next** (select next file) and
- **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”, i.e. make the “Disk Navigator” selecting the file played by the “Audio Navigator”, the command **ai_nav_getplayfile** can be used. See “Audio Navigation” on page 10. for a complete list of commands.

5.2.3 Audio Control: “control the audio stream (play/pause/fast forward/...)”

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 “Audio Control” on page 12. for a complete list of commands.

5.3 Features

5.3.1 Playlist

- Support playlist up to 65535 files.
- Support ASCII, UTF8 and UNICODE (UTF16LE & UTF16BE) text format.
- The navigation in the playlist is possible only in the “Audio Navigator”.

5.3.2 Navigation

- Navigation is done using the file creation order, not the alphabetic order.

5.3.3 File system

- FAT 12/16/32.
- 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.

5.4 Limitations

5.4.1 RAM

- The default software configuration of the project is designed to run on the UC3A0512 (64K of RAM).
- Depending on the RAM settings, the audio player may not be able to provide a track or metadata info (e.g. author) while playing another track. This may happen for example if the HEAP, which is used to supply buffers during audio decoding, becomes too small. This prevents the audio interface from allocating a new buffer which will contain the wanted metadata information.

5.4.2 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.

5.5 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 (. mp3, . wma, .m3u)

The “file list”:

- is updated when you exit or enter a directory or a disk.
- starts with the directories then the files.
- is sorted in the creation order.

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		Length of the serial number in bytes	Serial number	Returns the serial number of the connected device (if available).
ai_nav_drive_nb		Number of drive		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		Capacity of the drive		Returns the total space, in bytes, available on the device.
ai_nav_drive_free_space		Free space on the drive		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_goto	Position in file 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.

Table 5-2. Disk Navigator Commands

Command Name	Input	Output		Description
		Return	Extra result	
ai_nav_file_nb		Number of audio files		Returns the number of audio files in the file list. Audio files are all the files which extensions matches *.mp3 or *.wma. There is a specific command for playlist files, see below.
ai_nav_dir_nb		Number of directories		Returns the number of directories in the file list.
ai_nav_playlist_nb		Number of playlists		Returns the number of playlists in the file list. The playlists are all the files matching with the extension *.m3u.
ai_nav_dir_name		Length of the string in bytes	UNICODE name	Returns the name of the current directory.
ai_nav_file_name		Length of the string in bytes	UNICODE name	Returns the name of the selected file.
ai_nav_file_info_type		File type		Returns the type of file selected (MP3, WMA, ...).
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		Length of the string in bytes	UNICODE title	Returns the title of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_artist		Length of the string 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		Length of the string 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		Length of the string 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		Length of the string in bytes	UNICODE genre	Returns the genre of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_duration		Duration of the track		Returns the total time of the selected audio file in milliseconds if available, otherwise, returns 0.
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.

5.6 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 the file selected in the disk navigator. In other words, it synchronizes the audio navigator with the disk navigator and plays the selected file from the disk navigator. This commands does not change the current options (repeat/random/expmode). It is the opposite of the command **ai_nav_getplayfile**. Return false in case of error.
ai_audio_context_save		Structure Context	- True or false - The length of the structure in bytes	Gives complete audio context (player state, play time, repeat, random, file played, explorer mode).
ai_audio_context_restore	Structure Context	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/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/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/expmode), the next file to play.
ai_audio_nav_nb		Number of audio files 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		Length of the string in bytes	UNICODE name	Returns the name of selected file.
ai_audio_nav_file_info_type		File type		Returns the type of file selected (MP3, WMA, ...).
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		Length of the string 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		Length of the string 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		Length of the string 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		Length of the string 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		Length of the string 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		Duration of the track		Returns the total time of the selected audio file in milliseconds if available, otherwise, returns 0.

5.6.1 Types used by the interface

5.6.1.1 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.

5.6.1.2 Ai_shuffle_mode

Defines the shuffle/random mode:

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

5.6.1.3 Ai_explorer_mode

Defines the explorer mode (see Section 5.9.3 "Example 3 - Change the playing order" on page 15 for more information.

5.7 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 into fast forward mode. ***Function not implemented yet.***
ai_audio_ctrl_start_frw		True or false		Sets the audio player into 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 into the previous mode (play or pause). ***Function not implemented yet.***

5.8 Using Asynchronous or Synchronous API

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

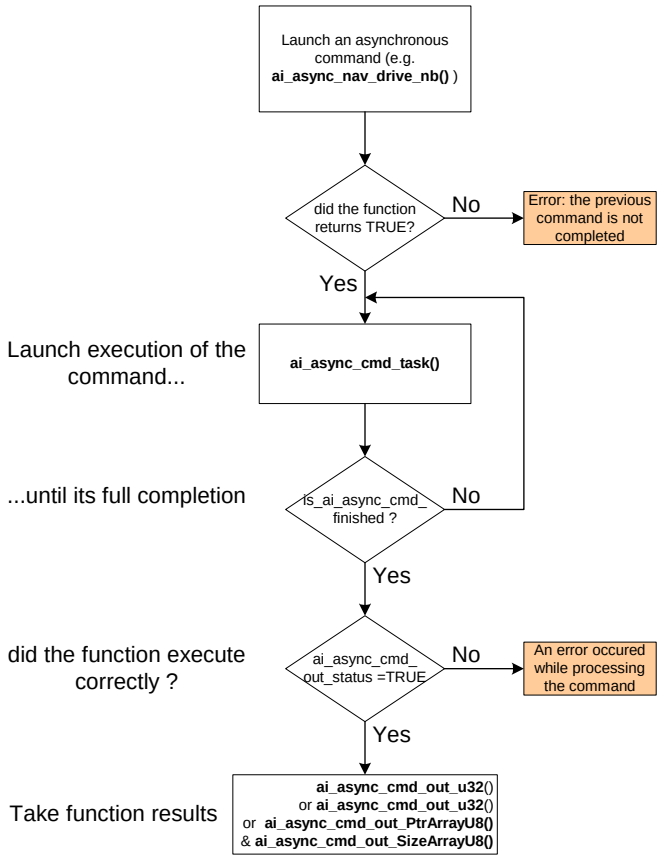
Using asynchronous function is more complicated: they *may* not produce the requested task in a single shot. Thus these functions need some other functions to properly operate:

- void ai_async_cmd_task(void): Executes an asynchronous command. Should be called after each asynchronous command when the application has free time.
- Bool is_ai_async_cmd_finished (void): This function returns TRUE if the last command is finished.
- U8 ai_async_cmd_out_status(void): Returns the status of the last executed command (CMD_DONE, CMD_EXECUTING or CMD_ERROR).

- U32 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.
- U64 ai_async_cmd_out_u64(void): if the last executed command should return a 64-bit result, this function will return this value.
- U16 ai_async_cmd_out_SizeArrayU8 (void): if the last executed command should return an extra result (e.g. a song name), this function returns the size in bytes of the extra result (no size limit).
- U8* 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.

The following picture shows the flow of the asynchronous function use.

Figure 5-2. Asynchronous function flow



5.9 Examples

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

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

5.9.1 Example 1 - Play “file1.mp3”

Table 5-5. Example: play “file1.mp3”

Command Order	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**(): plays the selected file file1.mp3
11	**ai_nav_file_goto**(3): goes to file position 3
12	**ai_nav_file_name**(): returns the name file2.wma

5.9.2 Example 2 - Play while browsing

Table 5-6. Example: play while browsing

Command Order	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**(): plays a file. By default it will seek inside the directories to play the first file which is “file5.wma” in our case.
4	**ai_nav_getplayfile**(): synchronizes the disk navigator with the audio navigator. Now the disk navigator is pointing on the “file5.wma” file.
5	**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).
6	**ai_nav_file_goto**(0): goes to file position 0
7	**ai_nav_file_name**(): returns the name “file5.wma“. (Notice the difference with Figure 5.9.1 - step #4)
8	**ai_nav_file_goto**(1): goes to file position 1
9	**ai_nav_file_name**(): returns the name “file6.mp3“
10	**ai_audio_ctrl_stop**(): stops the audio

5.9.3 Example 3 - Change the playing order

The playing order can be changed at compilation time by enabling the NAV_AUTO_FILE_IN_FIRST define (Section 6.5 “Project Configuration” on page 19).

Table 5-7. Playfile sequence

Command Order	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**(): plays a file.

If the NAV_AUTO_FILE_IN_FIRST define is **not** set, the sequence will play audio files in the following order:

Table 5-8. Playfile sequence with NAV_AUTO_FILE_IN_FIRST **undefined**

Order	File name	Parent directory path
0	file5.wma	/dir1/dir3/
1	file6.mp3	/dir1/dir3/
2	file4.mp3	/dir1/
3	file7.mp3	/dir2/
4	file1.mp3	/
5	file2.wma	/
6	file3.wma	/

Otherwise, if this define is set, the sequence will play audio files starting with files on the root:

Table 5-9. Playfile sequence with NAV_AUTO_FILE_IN_FIRST **defined**

Order	File name	Parent directory path
0	file1.mp3	/
1	file2.wma	/
2	file3.wma	/
3	file4.mp3	/dir1/
4	file5.wma	/dir1/dir3/
5	file6.mp3	/dir1/dir3/
6	file7.mp3	/dir2/

6. Source Code Architecture

6.1 Package

The EVK1105-AUDIO-PLAYER-MASS-STORAGE-MP3-X.Y.Z.zip contains projects for UC3A0512 RevH or later:

- EVK1105-AUDIO-PLAYER-MASS-STORAGE-X.Y.Z

Default hardware configuration of the project is to run on the EVK1105 board, although any board can be used (refer to section 6.4.7 “Board Definition Files” on page 19).

6.2 Documentation

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

- src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/readme.html

6.3 Projects / Compiler

The IAR™ project is located here:
- src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS_STORAGE/AT32UC3A0512_MP3/IAR/

The GCC makefile is located here:
- src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS_STORAGE/AT32UC3A0512_MP3/GCC/

The Avr32Studio project is located in the root-dir of the package:
- ./

6.4 Implementation Details

6.4.1 Main()

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

- src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/main.c

This function will:

- Initialize audio output - refer to section 6.4.8 “Audio Rendering Interface” on page 19
- Do the clock configuration
- Call the USB task -refer to section 6.4.5.1 “USB” on page 18
- Call the USB host Mass-Storage task. This task will check for new devices connection and initialize them using the USB Mass Storage protocol.
- Call the communication task (HMI) - refer to 6.4.4 “HMI Communication Task Example” on page 18.
- Call the decoder task to perform MP3 and/ or WMA decoding - refer to sections 6.4.2 “MP3” on page 18

The src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/ contains the following files:

- audio_mixer.c,h: audio mixer for DAC output.
- main.c: contains the main() function.
- host_mass_storage_task.c,h: USB host mass storage task.
- joystick_com_task.c,h: HMI with simple joystick interface.
- /CONF/*.h: configuration file for audio, communication interface, memory and navigation explorer. Please refer to section 6.5 “Project Configuration” on page 19 for more information on the configuration files.

6.4.2 MP3

The MP3 source files are located in:

- src/SERVICES/AUDIO/MP3/LIBMAD/: AVR32 port of LibMAD MP3 decoder

A library form of the decoder is provided in /UTILS/LIBS/LIBMAD/AT32UC/.

ID3 is supported up to version 2.4. The ID3 reader source is located in:

- src/SERVICES/AUDIO/MP3/ID3/reader_id3.c,h

6.4.3 Audio Player API

The Audio Interface API is located in:

- src/SERVICES/AUDIO/AUDIO_PLAYER/audio_interface.h

The Mass Storage Audio Interface can be found in:

- src/SERVICES/AUDIO/AUDIO_PLAYER/AI_USB_MS/
  - ai_usb_ms.c,h: Mass Storage Audio interface.
  - host_mass_storage_task.c,h: USB host mass storage task.

Refer to Section 5. “Audio Interface API” on page 4 for more details.

6.4.4 HMI Communication Task Example

The included firmware implements an HMI example using a keypad and a SPI-driven LCD:

- src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/joystick_com_task.c,h: HMI with simple keypad interface

6.4.5 AT32UC3A Drivers

The example firmware uses the AVR32 UC3 driver library available in

- src/UTILS/LIBS/DRIVERS/AT32UC3A/.

6.4.5.1 USB

The USB low level driver is located in:

- src/DRIVERS/USB/

The USB mass storage service is located in:

- src/SERVICES/USB/CLASS/MASS_STORAGE/

6.4.6 FAT File System

The FAT12/16/32 files is located in the directory src/SERVICES/FAT/.

6.4.7 Board Definition Files

The application is designed to run on the EVK1105. All projects are configured with the following define: BOARD=EVK1105. The EVK1105 definition can be found in the src/BOARDS/EVK1105 directory.

6.4.7.1 Board customization

For IAR project, open the project options (Project -> Options), choose the «C/C++ Compiler», then «Preprocessor». Modify the BOARD=EVK1105 definition by BOARD=USER_BOARD.

For GCC, just modify in the config.mk file (src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/AT32UC3A0512_MP3/GCC/) the DEFS definition with -D BOARD=USER_BOARD.

For Avr32Studio, open the project properties (Project -> Properties), go in the «C/C++ build», then «Settings», «tool settings» and «Symbols». Modify the BOARD=EVK1105 definition by BOARD=USER_BOARD.

6.4.8 Audio Rendering Interface

The src/COMPONENTS/AUDIO/CODEC/TLV320AIC23B/ directory contains the driver for the external DAC TLV320AIC23B.

Audio output is using the SSC module to generate I2S frames using internal DMA to free CPU cycles for audio decoding.

6.5 Project Configuration

The project configuration files can be found in the src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/CONF/ directory directory.

Configuration files are not linked to IAR, GCC or Avr32Studio projects. The user can alter any of them, then rebuild the entire project in order to reflect the new configuration.

- 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.
- 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.
- conf_audio_player.h: this file is a set of defines that configure the general features of the application. Following are the main parameters:
  - *DEFAULT_COM_TASK, defines the peripheral used to communicate to the audio player. The current value is JOYSTICK_COM_TASK. The joystick_com_task is in charge of the Human Machine Interface.*
  - *DEFAULT_DACS, specifies the default audio DAC used for the audio output. Two values are possible: AUDIO_MIXER_DAC_AIC23B for the I2S interface and AUDIO_MIXER_DAC_PWM_DAC to use PWM channels (external low-pass filter is required). The default configuration uses the external DAC (tlv320aic23b) mounted on the EVK1105 board. Note that the ABDAC is not supported in the current release of the application.*

- 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 play first the files at the root of a directory instead of the one inside 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".*
- conf_pwm_dac.h: configuration of the PWM DAC (which PWM channel is used, which pins are concerned).
- conf_tlv320aic23b.h: configuration of the I2S DAC (which pins are used and which configuration interface).
- conf_usb.h: configuration file used for the USB.

6.6 Compiling the application

The following steps show you how to build the embedded firmware according to your environment

6.6.1 If you are using AVR32Studio

- Launch avr32Studio
- Create a new AVR32 C project («File» -> «new» -> «AVR32 C Project»).
- Fill-in the dialogue box with project name, set target MCU to UC3A0512 and press finish.
- Choose Import archive file («File» -> «import»…), press the "next" button.
- Select the EVK1105-AUDIO-PLAYER-MASS-STORAGE-X.Y.Z.zip archive file with the browse button. Select «into folder», check «Overwrite existing resources without warning» and press the "finish" button.
- The project is now available in the given project name.
- Press the build button
- Load the Code: Please refer to the application note AVR32015: AVR32 Studio getting started

6.6.2 If you are using GCC with the AVR32 GNU Toolchain

- - Open a shell, go to the src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/AT32UC3A0512_MP3/GCC/ directory and type:
  make rebuild program run

6.6.3 If you are using IAR Embedded Workbench® for Atmel AVR32

- - Open IAR and load the associated IAR project of this application (located in the directory src/APPLICATIONS/EVK1105-AUDIO-PLAYER-MASS-STORAGE/AT32UC3A0512_MP3/IAR/).
- - Press the "Debug" button at the top right of the IAR interface.
  The project should compile. Then the generated binary file is downloaded to the microcontroller to finally switch to the debug mode.
- - Click on the "Go" button in the "Debug" menu or press F5.

Headquarters

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

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr32@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature