Atmel AVR32716: AVR UC3 USB Audio Class

Features

- **Standalone USB device audio class processing application**
- **USB audio stream re-synchronization**
- **Standalone - low memory footprint (code and RAM) and no operating system dependency**
- **Audio output over I^2S using synchronous serial controller (SSC) or internal audio bitstream DAC (ABDAC)**
- **USB "Stereo with Microphone" USB device with HID control with a keypad**
- **Source code for GCC and IAR™ compilers**

1. Introduction

This application is a "Stereo headset with Microphone" USB device which demonstrates the usage of the Atmel® AVR® UC3 USB AUDIO class through a real-time audio application.

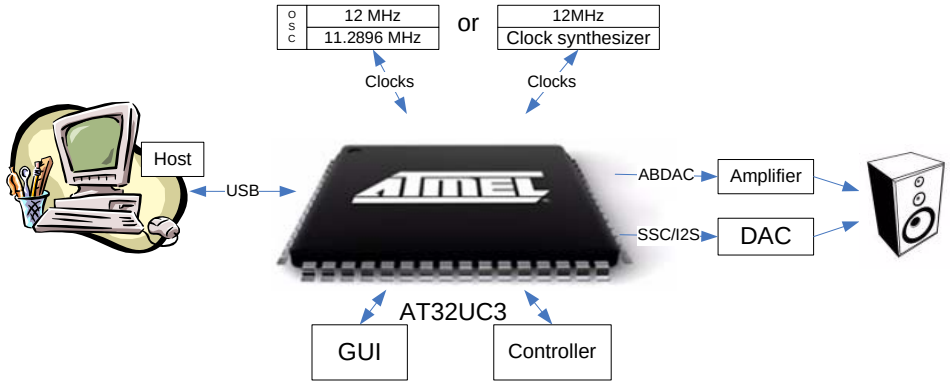
This application is designed to work with the Atmel EVK1105 evaluation kit. The application source code is provided for GCC and IAR compilers.

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

To be able to use this document efficiently, the reader should be experienced in using USB protocols and classes.

This application is based on the Atmel AVR UC3 Software Framework. The source code is available in the AVR UC3 Software Framework.

Figure 1-1. Block diagram.



2. 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®
- Atmel AVR32 Studio and the GNU toolchain (GCC) or IAR Embedded Workbench® for Atmel AVR UC3 compiler
- ATmel AVR JTAGICE mkII or Atmel AVR ONE! debugger

32-bit Atmel Microcontroller

Application Note

3. Related parts

Even if the application note is dedicated to the Atmel EVK1105 (Atmel AT32UC3A0512), this documentation also applies to the following Atmel AT32UC3 parts:

- AT32UC3A0xxx
- AT32UC3A1xxx
- AT32UC3A3xxx
- AT32UC3B0xxx
- AT32UC3B1xxx

4. Related items

The software provided with this application note requires several components:

  - **32-bit AVR GNU Toolchain**: AVR UC3 GNU Toolchain is a set of standalone command line programs used to create applications for AVR UC3 microcontrollers (compiler, assembler, linker, debugger).
    http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4118
- or
  - **IAR Embedded Workbench for Atmel AVR UC3**: IAR Embedded Workbench provides a suite of AVR UC3 development tools for embedded systems (compiler, assembler, linker, debugger).
    http://www.iar.com/
- **Atmel EVK1105**: The EVK1105 is an evaluation kit and development system for the AT32UC3A0512 microcontroller.
  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4428
- **Atmel AVR UC3 Software Framework**: This framework provides software drivers and libraries to build any application for AVR UC3.
  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4192
- **USB AUDIO Stereo Headset with microphone software**: The software on which this documentation is based. It is available in the AVR UC3 Software Framework.
  http://www.atmel.com
- **Atmel AT32UC3A0512 datasheet**:
  http://www.atmel.com/dyn/products/product_card.asp?part_id=4117
- **Atmel AVR32788: 32-bit AVR How to use the SSC in I²S mode**:This application note describes how the I²S protocol is handled on AVR UC3 devices and gives important information about how to get the best configuration for different sample rates.
  http://www.atmel.com/dyn/resources/prod_documents/doc32127.pdf
- **AVR32769: How to compile the standalone AVR UC3 Software Framework in AVR32 Studio V2**: This application note describes how an Atmel AVR32 Studio project can be easily created:
  http://www.atmel.com/dyn/resources/prod_documents/doc32115.pdf

5. Theory of operation

5.1 Overview

The USB is very well suited for transport of audio (voice and sound). In addition, the USB has more than enough bandwidth for sound, even high-quality audio. Many applications related to voice telephony, audio playback, and recording can take advantage of the USB.

This application note demonstrates, through a stand alone application, the usage of the Atmel AVR UC3 USB audio class. The application operates in the USB device mode, enumerating as a “stereo headset with microphone”. It uses the USB Audio and HID (consumer) classes.

An essential issue in audio is synchronization of the data streams. Indeed, the smallest artifacts are easily detected by the human ear. Therefore, a robust synchronization scheme on isochronous transfers needs to be added in order to have a high quality sound output.

The USB specification describes several type of audio synchronization endpoint:

- Asynchronous: Asynchronous isochronous audio endpoints produce or consume data at a rate that is locked either to a clock external to the USB or to a free-running internal clock. These endpoints cannot be synchronized to a start of frame (SOF) or to any other clock in the USB domain
- Synchronous: The clock system of synchronous isochronous audio endpoints can be controlled externally through SOF synchronization. Such an endpoint must do one of the following:
    - Slave its sample clock to the 1ms SOF tick
    - Control the rate of USB SOF generation so that its data rate becomes automatically locked to SOF
- Adaptive: Adaptive isochronous audio endpoints are able to source or sink data at any rate within their operating range. This implies that these endpoints must run an internal process that allows them to match their natural data rate to the data rate that is imposed at their interface

The 32-bit Atmel AVR UC3 Flash Microcontroller family does not allow the use of *synchronous* stream. Indeed, the product is not able to slave the DAC clock to the 1ms SOF tick received in the USB device mode.

The current audio class application presented here is thus using *adaptive* synchronization type.

5.1.1 Benefits of the adaptive synchronization type

Thanks to the adaptive synchronization, an audio application can support several sampling frequencies (for example, 32, 44.1 and 48kHz). All the supported frequencies of the speaker and the microphone will be declared to the host during the USB enumeration of the device. Then, depending on the sampling frequency of the “file” being played on the PC, the USB will ask the device to change the sampling frequency accordingly (for example, switching from 44.1 to 48kHz) before starting the playback.

Note that you can, if you want, support only one sampling frequency for the speaker output into the final application. In that case, the operating system is in charge of resampling, if needed, the audio stream to that single frequency.

This is what the current application is doing: it supports only the 44.1kHz sampling frequency (due to the crystals configuration of the Atmel EVK1105). The PC will anyway be able to play 32

or 48kHz files. It will resample them to 44.1kHz before sending the digital audio stream through the USB.

5.1.2 Synchronization methods

The USB specification describes three different type of synchronization endpoints, as said earlier in that document. Therefore, there are several methods that are usually implemented in final consumer products:

- **Method A**
  Insertion / removal of audio samples. This method is well suited for low cost designs. It consists in adding or removing samples on-the-fly in order to prevent overrun or underrun situations. This method will give good results as long as the incoming USB audio sampling frequency is very close to the DAC input clock frequency. However, if the difference between the two frequencies is too high, this will add audio distortion.
  This is one of the three methods covered by this application note.
- **Method B**
  Use of an external clock synthesizer to dynamically adapt the external DAC frequency. This method gives the best results. It consists in slightly increasing or decreasing the sample frequency of the DAC by few PPM in order to avoid overrun/underrun. This is undetectable by the human ear and guaranty a perfect 1 to 1 copy of the digital audio stream.
  This is the second method covered by this application note, achieved by adding an external component (for example, the Cirrus Logic CS 2200). This solution gives the best performances.
- **Method C**
  Adaptive resampling. It consists in resampling the incoming stream at a frequency which may slightly change over the time, in order to avoid underrun or overrun situation. This method needs CPU resources and resampling adds distortion to the audio signal (less than the method A though).
  This is the last method covered by this application note.
- **Method D**
  Use of a feedback endpoint. This method, fully described in the USB specification, consists in giving to the Host an accurate indicator of the required stream rate (according to the slave). The feedback endpoint seems to not be fully supported on Windows XP.

5.1.3 Summary

The pros. and cons. of each synchronization methods is given in Table 5-1 on page 5.

.

Table 5-1. Pros. and cons. of each synchronization method.

Method	Pros.	Cons.
Method A: **Insertion/removal of audio samples**	Very simple. Good audio results for low-cost design: audio distortion almost not audible by the human hear. For example, add/remove samples every 208ms for 100ppm diff. Few CPU and RAM resources	Audio distortion audible on pure sinus wave. 12MHz and 11.2896MHz crystal configuration: only 44.1kHz support.
Method B: **Use of an external clock synthesizer**	Very simple. Few CPU and RAM resources. Perfect audio quality: direct 1:1 audio copy from the USB to the DAC. No distortion / no data processing Support any sampling frequencies.	BOM cost is increased.
Method C: **Adaptive resampling**	Good audio results, but still audio distortion due to the resampling.	Needs CPU & RAM resources (see Section 9.3.3 Method C: Adaptive resampling (page 21)) Audio resampling implies loss of quality compared to method B.
Method D: **Use of a feedback endpoint** ***(under analysis)***	Described in the USB 2.0 standard. Very simple. Few CPU and RAM resources. Should give very good audio quality *(needs to be confirmed)*	Seems not natively supported by Windows XP operating system.

5.1.4 HID control of the application

The application note also supports the control of applications such as Windows Media player through the use of an HID endpoint. During the USB enumeration, the device declares an HID “consumer” interface which handles typical remote control (play/pause/next/previous/volume up and down).

It is thus possible to change the track (next/previous) by respectively using the right and left key of the Atmel EVK1105. The master volume can be controlled too.

5.1.5 Audio synchronization methods used in this application

As said earlier, the application is re-synchronizing the USB audio stream thanks to three possible methods. As a consequence, the application is splitted into three different projects:

1. The first project is using the method A and is located here:
   /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_ADD_DEL_SAMPLES/
2. The second project is using the method B (through the use of an external clock synthesizer - the Cirrus Logic CS2200-CP). The project is located here:
   /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_CLOCK_SYNTHESIZER/
3. The third project is using the method C and is located here:
   /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_ADAPTIVE_SRC/

6. Implementation details

We will describe here how are working these three synchronization methods.

6.1 Method A: Add/remove of audio samples

6.1.1 Hardware consideration

The Atmel EVK1105 does not need any hardware patch or modification to run in this configuration.

6.1.2 Supported sampling frequencies

The EVK1105 has a 11.2896MHz crystal that is well suited for 44.1kHz sampling frequencies. However, this crystal can not be used to get accurate 32kHz or 48kHz frequencies. As a consequence, during the USB enumeration process, the UC3 will only present to the host one sampling frequency: 44.1kHz. Any other sampling frequencies played from a PC/MAC will be re-sampled at 44.1kHz (for example a 48kHz file played on a PC will be re-sampled at 44.1kHz).

6.1.3 Clock consideration

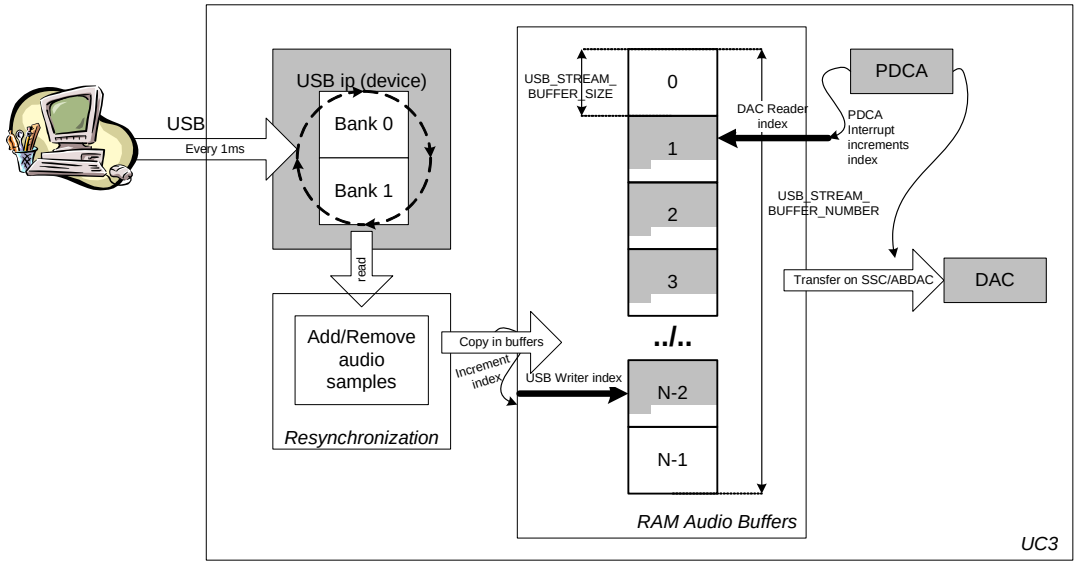
The CPU frequency is configured to 62.09MHz (11.2896 x 5.5).

Note that with this method, the oversampling frequency sent to the external DAC (when using the SSC I^2S) is fixed and exactly 11.2896MHz (256 x 44100).

6.1.4 Algorithm

The basics of the synchronization is simple: the software uses audio buffers in the internal RAM and constantly monitors the reader (the DAC) and writer (the USB) indexes. First, the system waits until half of the total number of buffers contains audio samples before starting the playback.

Figure 6-1. Buffering scheme.



Then, the playback can start and the system monitors the read and write indexes. As soon as these two pointers becomes too far or too close, the software will add (copy) or remove samples on a regular basis (one stereo sample at a time), in order to avoid audible effects.

The period (in micro-second) of adding/removing samples is given by:

$$\frac{1000000}{|Fusb - Fdac|}$$

For example, a 100ppm difference between to the two frequencies will lead to an audio sample addition/removal every 208ms.

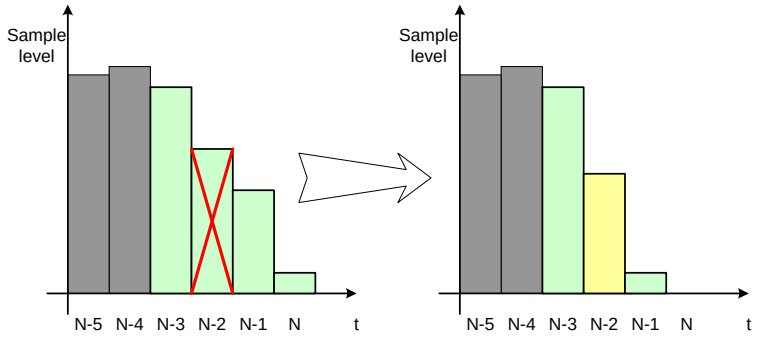
The period of the buffer indexes monitoring is configurable and should not be too fast, otherwise the control loop will be unstable. It can not be too slow either, otherwise the corrections will not prevent overrun or underrun.

Considering that N represents the last sample of a received packet, the sample removal is done using the following scheme:

- Sample N-2 is rebuilt and equal to the average of N, N-1, N-2 and N-3
- Sample N is moved at N-1
- The packet size is reduced of one sample

This is shown in Figure 6-2.

Figure 6-2. Audio sample removal.

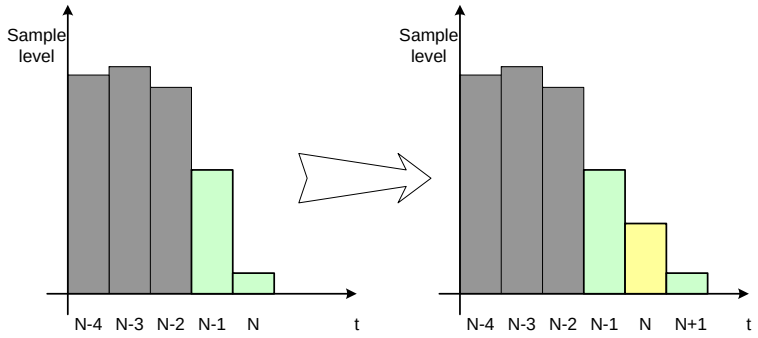


The sample insertion is done using the following scheme:

- Sample N is moved at N+1
- Sample N is rebuilt and equal to the average of N-1 and N+1
- The packet size is increased of one sample

This is shown in Figure 6-3.

Figure 6-3. Audio sample insertion.

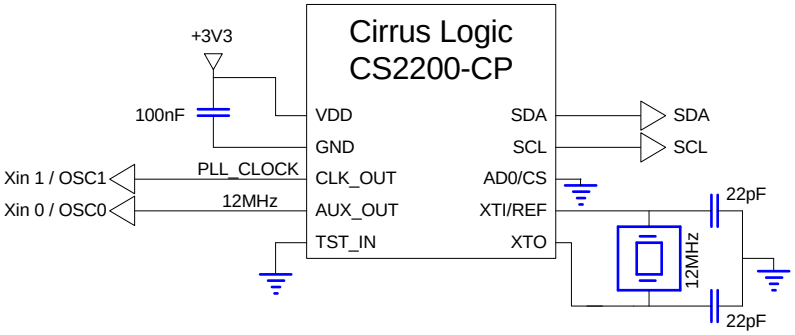


6.2 Method B: Use of an external clock synthesizer

6.2.1 Hardware consideration

The Atmel EVK1105 does not have any external clock synthesizer onboard. Thus, the kit must be modified in order to use this solution. The 12MHz and 11.2896MHz crystals must be removed, and the clock synthesizer must be connected as described in Figure 6-4.

Figure 6-4. Cirrus Logic CS2200-CP connection.



The CS2200-CP is an extremely versatile system clocking device that utilizes a programmable phase lock loop. The CS2200-CP is based on an analog PLL architecture comprised of a Delta-Sigma Fractional-N Frequency Synthesizer. This architecture allows for frequency synthesis and clock generation from a stable reference clock.

This component will produce a low-jitter clock (typical 70ps) and a very small frequency resolution (±0.5ppm).

6.2.2 Supported sampling frequencies

Thanks to the CS2200-CP, the application is able to manage three sampling frequencies: 32kHz, 44.1kHz and 48kHz. This means that, during the USB enumeration process, the UC3 will declare these three frequencies to the USB host. Any other sampling frequencies played from a PC/MAC will be re-sampled into one of the supported frequencies (for example, a 22.05kHz file played on a PC will be re-sampled at 48kHz).

6.2.3 Clock consideration

The CPU frequency is not fixed and will vary according to the sampling frequency of the file being played.

For a 32kHz file, the CS2200-CP will be configured to output a 8.192MHz (32000 x 256) master clock and the CPU frequency will be 40.9MHz (8.192 x 5).

For a 44.1kHz file, the CS2200-CP will be configured to output a 11.2896MHz (44100 x 256) master clock and the CPU frequency will be 56MHz (11.2896 x 5).

For a 48kHz file, the CS2200-CP will be configured to output a 12.288MHz (48000 x 256) master clock and the CPU frequency will be 61MHz (12.288 x 5).

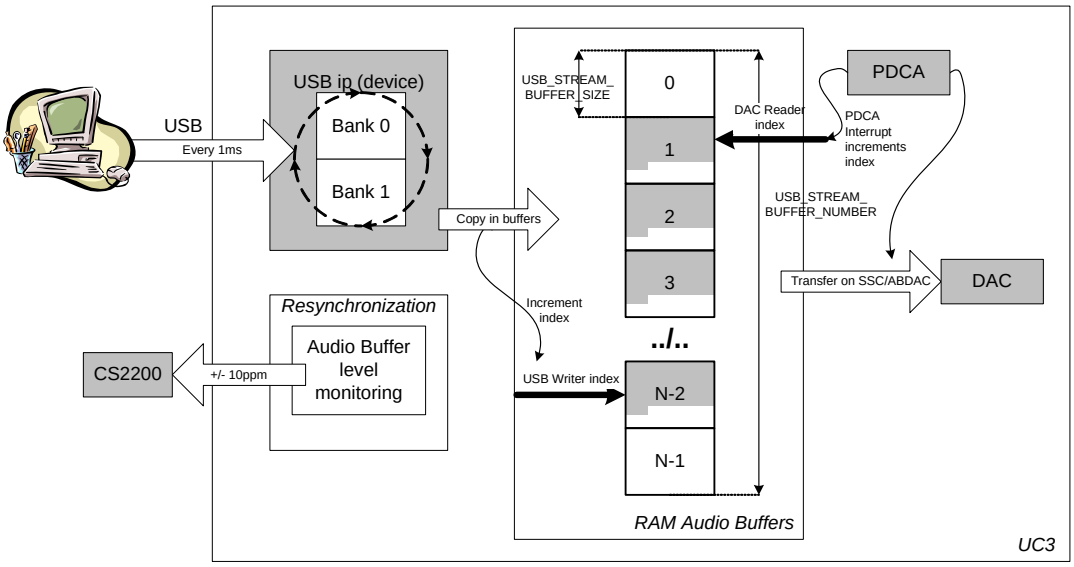
Note that with this method, the oversampling frequency sent to the external DAC (when using the SSC I^2S) is exactly 256 x Fs.

6.2.4 Algorithm

Here is the algorithm used by this synchronization method:

- The application bufferises the audio stream in a ring buffer in the internal RAM. This ring buffer represents 16 buffers (USB_STREAM_BUFFER_NUMBER) of 1ms of data, that is, 16ms
- The application waits that at least eight buffers are full before launching the playback
- Each packet received from the USB is put in a buffer. This increments a writer index
- Buffer are sent to the SSC by the PDCA. This increments a reader index
- The application has been designed to support an initial frequency difference of 500ppm. This may happen when you are connected on a PC. The USB specification allows a possible maximum frequency deviation of 500ppm
- For that, we add ±10ppm every 320ms:
    - Buffering represents 16ms of audio. Playback starts with half buffer: 8ms
    - When input and output clock differ by 500ppm, the buffer will over/underflow after 8ms/500ppm = 16sec
    - We make correction by steps of 10ppm, we need 50 steps to correct the 500ppm difference
    - These 50 corrections must be made within 16s. This means every 16/50 = 320ms

Figure 6-5. Buffering scheme.



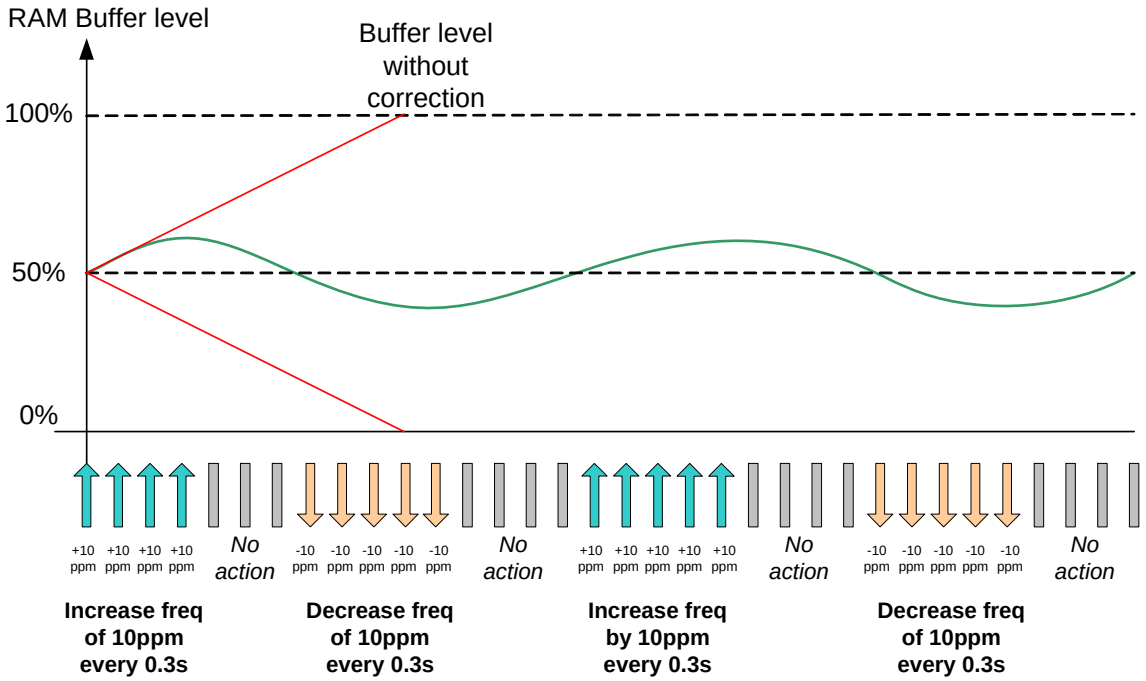
- The application periodically monitors (every 320ms) the reader and writer indexes and take actions using the following algorithm:

```
If buffer_level >50% and if trend is positive
        Then increment CS2200 out clock by 10ppm
Else if buffer_level <50% and if trend is negative
        Then decrement CS2200 out clock by 10ppm
```

The *trend* is the tendency of the buffer level, that is, the *increase* or the *decrease* direction.

This is illustrated in Figure 6-6 on page 10.

Figure 6-6. RAM buffer level with the CS2200-CP clock synthesizer.



Thanks to this method, there is no data distortion, no data loss. The frequency deviation is not audible by a human hear. This method is very well suited for high end consumer audio products.

6.3 Method C: Adaptive resampling (or sampling rate conversion)

6.3.1 Hardware consideration

The Atmel EVK1105 does not need any hardware patch or modification to run in this configuration.

6.3.2 Supported sampling frequencies

The EVK1105 has a 11.2896MHz crystal that is well suited for 44.1kHz sampling frequencies. However, this crystal can not be used to get accurate 32kHz or 48kHz frequencies. As a consequence, during the USB enumeration process, the UC3 will only present to the host one sampling frequency: 44.1kHz. Any other sampling frequencies played from a PC/MAC will be re-sampled at 44.1kHz (for example, a 48kHz file played on a PC will be re-sampled at 44.1kHz).

6.3.3 Clock consideration

The CPU frequency is configured to 62.09MHz (11.2896 x 5.5).

Note that with this method, the oversampling frequency sent to the external DAC (when using the SSC I^2S) is fixed and exactly 11.2896MHz (256 x 44100).

6.3.4 Algorithm

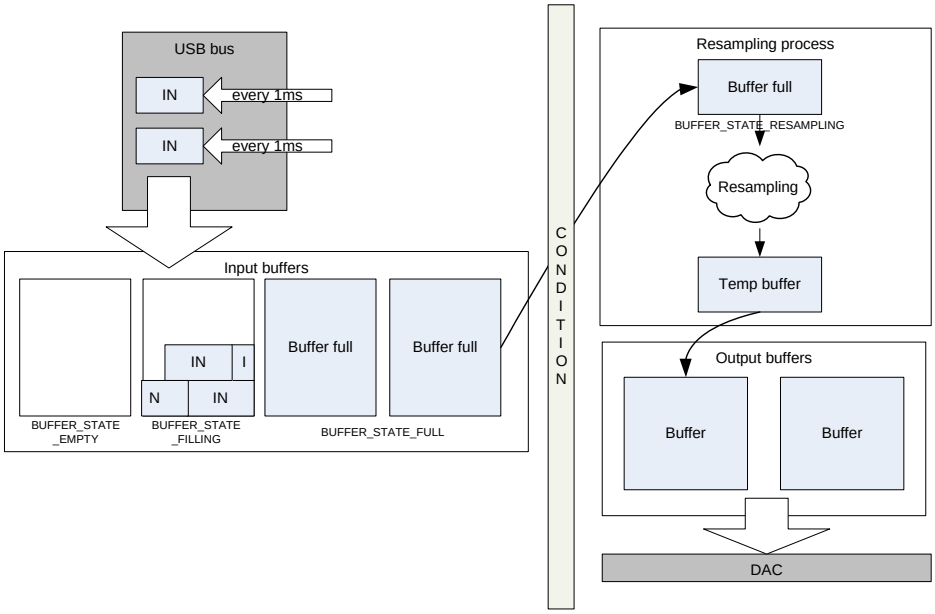
Here is the algorithm used by this synchronization method:

The data path is composed of a number of “input” audio buffers and “output” audio buffers.The input buffers are circular buffers. Each buffer follows the following process during one loop:

- BUFFER_STATE_EMPTY: the buffer is empty, no valid data are inside, it is waiting to be used

- BUFFER_STATE_FILLING: the buffer is being filled by data directly taken from the USB bus (IN requests). Only one buffer can be in this state at a time
- BUFFER_STATE_FULL: the buffer is full of valid data that are waiting to be processed
- BUFFER_STATE_RESAMPLING: the buffer is being re-sampled and the output is stored into the output buffers through a temporary buffer

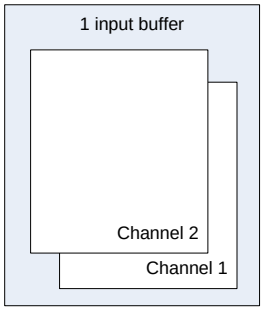
Figure 6-7. Buffering scheme.



After this final state, the buffer goes back to the BUFFER_STATE_EMPTY and renews the procedure.

All buffers are subdivided into channels. If the AUDIO_STREAM_MAX_NB_CHANNELS define is equal to two, then each buffer will be subdivided into two channels.

Figure 6-8. Audio buffer composition.



This buffer has two channels, left (1) and right (2). The size of EACH channel is equal to AUDIO_STREAM_MAX_USB_PACKET_SIZE.

Therefore, the buffer itself has a size of AUDIO_STREAM_MAX_USB_PACKET_SIZE x 2

All buffers have a fixed size which is equal to AUDIO_STREAM_MAX_USB_PACKET_SIZE * AUDIO_STREAM_MAX_NB_CHANNELS (= 360 x 2 = 720 bytes by default).

At startup, to pre-fill the input buffer, the SYNCHRONIZATION flag is initially raised. When the number of full input buffers is enough, this flag is cleared which validate one item of the "condition" (see Section Section 6.3.4.1 “Condition” on page 12).

6.3.4.1 Condition

The condition is validated if all the following statements are true:

- The DAC is ready to receive a new buffer
- The SYNCHRONIZATION flag is not raised

6.3.4.2 Resampling process

This process is linear and so is processed in one time. The current full buffer will change to the state "re-sampling". During this time, the samples of each channel of the current buffer are re-sampled and reformatted using a temporary buffer before being stored inside the free output buffer.

The temporary buffer has a fixed size. Its size is calculated from the output of a resampling buffer. For example, if the USB streaming outputs a 22050Hz sound, and we want to output on the DAC a 44100Hz sound. The ratio is two, therefore the temporary buffer will be equal to twice the size of one channel of the input buffer. By default, one channel of the input buffer is 360, therefore, the temporary buffer will be 720bytes.

6.3.4.3 Output buffers

The output buffers can be directly used by the DAC with no need of recopying the buffer. Therefore, if used with a PDCA, the output buffers should be of a number of two, one used for the load buffer and one for the reload buffer.

Their size is also fixed and is equal to the size of the temporary buffer (see Section “Resampling process”) multiplied by the number of channels.

6.3.4.4 Over/Underrun conditions handling

The overrun or underrun conditions are taken care by over or under re-sampling the stream on the fly.

For each frequency supported, three re-sampling frequencies are defined:

- The normal output frequency defining the sampling frequency at which the stream should be played when no over/underrun conditions are met: 44.1kHz for example
- The under-sampling frequency, which will be used once the stream is facing an overrun condition. We need to produce fewer samples to slow-down the audio stream. The under-sampling frequency can be 44kHz for example
- The over-sampling frequency, which will be used once the stream is facing an underrun condition. We need to produce more samples to speed-up the audiostream. The over-sampling frequency can be 44.2kHz for example

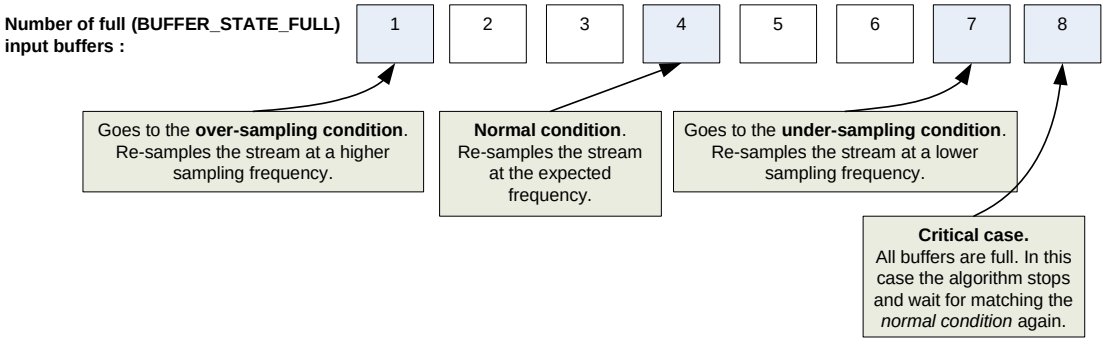
To determine an over or under running condition, the algorithm will check the input buffer statuses. If there are not enough buffers full, then, it assumes to be in an underrun condition; the same applies for the overrun condition.

Then, the algorithm will automatically switch from the normal output frequency to the over/under-sampling frequency to recover from the unexpected frequency excursion.

Therefore, these conditions depend on the number of input buffer chosen. By default, the software is configured so that the overrun condition will triggered if only one full buffer is left while the underrun condition triggers if (NB_INPUT_BUFFER - 1) are full.

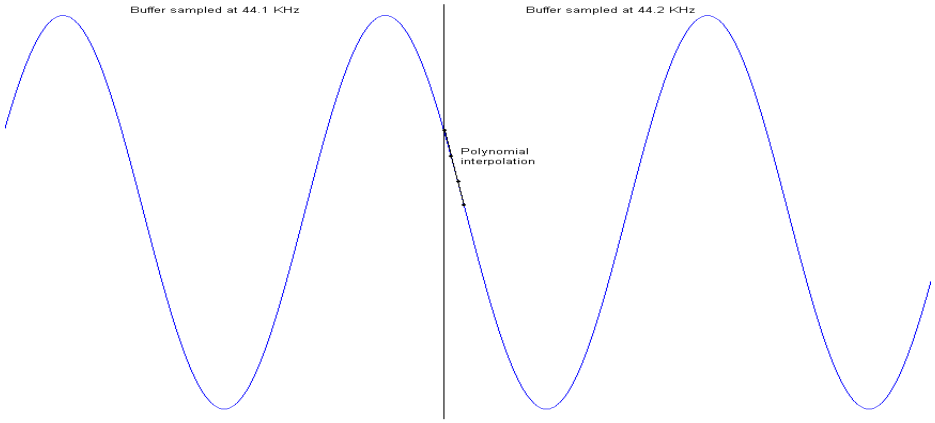
In this setup (see Figure 6-9 on page 13), the optimal number of input buffers is eight.

Figure 6-9. Overrun/underrun condition.



To ensure a perfect transition between two re-sampled buffers at a different frequency, a polynomial interpolation is done in addition to the re-sampling for few samples. This is taken care by the DSP library.

Figure 6-10. Polynomial interpolation from the DSP library.



6.3.4.5 Frequency response

The default configuration of the resampling module is:

- 6th order re-sampling order
- filter type used is a low-pass windowed sinc

The frequency response (-3db) is equal to:

```
Fc = min(Fsin, Fsout) / 2
```

with F_{SIN}: input sampling frequency; F_{SOUT}: output sampling frequency.

For example, when resampling from 32kHz to 48kHz, the frequency response will be 16kHz.

6.4 Special consideration for the SSC module

You may have noticed that the CPU frequency is changed dynamically depending on the sampling frequency of the audio stream. This is because the SSC module derives its clock internally only from the peripheral bus. When you want to change the SSC frequency, the whole system clock needs to be changed.

There is an alternative which gives the possibility to isolate the SSC module from the system clock, by using a loopback of a generic clock on the SSC RX pin. But in this case, the microphone feature can not be used.

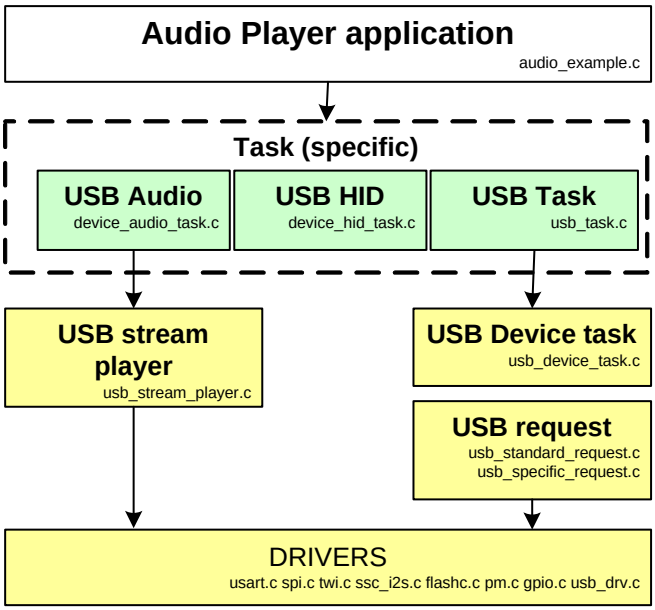
This approach is fully described in the AVR32788: 32-bit AVR How to use the SSC in I²S mode application note.

We strongly encourage the reader to take a look into this document in order to get familiar with the UC3 SSC module.

7. Software architecture

Figure 7-1 shows the basic software architecture of the application.

Figure 7-1. 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 streaming, HMI, and USB management possible. There are three tasks:

- The USB task: This task handles the USB stack and events
- The USB Audio task: this task manages the USB audio stream
- The USB HID task: This task is in charge of the HID management, which consists in sending reports depending on the keyboard events (next/previous, ...)

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

```
while (TRUE)
{
   usb_task();
   device_audio_task();
   device_hid_task();
}
```

8. Source code architecture

8.1 Package

The EVK1105-USB-AUDIO-x-y-z.zip contains projects for GCC (or Atmel AVR32 Studio) and IAR.

Default hardware configuration of the project is to run on Atmel AVR UC3 evaluation kits, although any board can be used (refer to Section 8.4.3 “Board definition files” on page 16).

Note that this application is also included into the Atmel AVR UC3 Software Framework.

8.2 Documentation

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

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/readme.html

8.3 Projects / compiler

The IAR projects are located here:

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_ADD_DEL_SAMPLES/<part>-<board>/IAR/

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_CLOCK_SYNTHESIZER/<part>-<board>/IAR/

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_ADAPTIVE_SRC/<part>-<board>/IAR/

The GCC makefiles are located here:

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_ADD_DEL_SAMPLES/<part>-<board>/GCC/

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_CLOCK_SYNTHESIZER/<part>-<board>/GCC/

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/AUDIO_EXAMPLE_ADAPTIVE_SRC/<part>-<board>/GCC/

An Atmel AVR32 Studio project can be easily created by following the steps from the “AVR32769: How to Compile the standalone AVR UC3 Software Framework in AVR32 Studio V2” application note.

8.4 Implementation details

The following describes the code implementation of the USB audio player running on the Atmel EVK1105. Other available packages are similar so you will find useful information here that applies to every project configurations.

8.4.1 Main()

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

- /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/audio_example.c

This function will:

- Initialize audio output - refer to Section 8.4.4 “Audio rendering interface” on page 16
- Do the clock configuration
- Call the USB task. Refer to Section 8.4.2.1 “USB” on page 16
- Call the USB audio task. This task will handle the audio stream
- Call the USB HID task. This task will handle the HID report

The /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/ folder contains the following files:

- ./audio_example.c/.h: contains the main() function
- ./avr32_logo.c/.h: the AVR UC3 bitmap image
- ./device_audio_task.c/.h: contains the task that manages the audio stream from/to the USB
- ./device_hid_task.c/.h: contains the task that manages the HID
- ./microphone_samples.c/.h: contains pre-built microphone samples
- ./CONF/*.h: configuration file for audio, display and USB. Please refer to Section 8.5 “Project configuration” on page 18 for more information on the configuration files

8.4.2 AT32UC3A drivers

The firmware uses the Atmel AVR UC3 driver library available in:

- /UTILS/LIBS/DRIVERS/AT32UC3A/

The source code can be found into /DRIVERS.

8.4.2.1 USB

The USB low level driver is located in:

- /DRIVERS/USBB/

8.4.3 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 /BOARDS/EVKxxxx directory.

8.4.3.1 Board customization

- 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’, just modify in the ‘config.mk’ file (/SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/<part>-<board>/GCC/) the DEFS definition with ‘-D BOARD=USER_BOARD’

8.4.4 Audio rendering interface

The audio DAC mixer source code is located in:

- /SERVICES/AUDIO/AUDIO_MIXER/audio_mixer.c,h.

8.4.4.1 I^2S using SSC module

The /COMPONENTS/AUDIO/CODEC/TLV320AIC23B/ directory contains two drivers: a stand-alone release for the external DAC TLV320AIC23B, and another with the CS2200 support.

The Atmel AVR UC3 communicates with the TLV320AIC23B with the Two Wire Interface (TWI). The UC3 is the TWI master.

The UC3 SSC module generates I^2S frames using internal DMA (PDCA) to free CPU cycles for the application.

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, outputted 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:

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

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

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

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

The music interval in semitones is:

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

Table 8-1. Samples rate with SSC module (using FOSC0=12MHz and FOSC1=11.2896MHz).

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: 32-bit AVR How to use the SSC in I^2S mode" application note.

8.4.4.2 ABDAC

The ABDAC driver is located in /DRIVERS/ABDAC.

The /COMPONENTS/AUDIO/AMP/TPA6130A/ directory contains two drivers: a standalone driver for the external amplifier driver (TPA6130A), and a second version with the CS2200 support.

The Atmel AVR UC3 ABDAC module is using the internal DMA (PDCA) to free CPU cycles for the application.

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 8-2. Samples rate with ABDAC module (using FOSC0=12MHz and FOSC1=11.2896 MHz).

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

8.5 Project configuration

The configuration files are located under the /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/CONF/ directory and should be modified to change, separately, audio player's module configurations.

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.

8.5.1 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.

8.5.2 conf_conf_et024006dhu.h

Configures all parameters relative to the LCD display.

8.5.3 conf_qt60168.h

Configuration of the Atmel QTouch® 60168 component used on the kit.

8.5.4 conf_tlv320aic23b.h

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

8.5.5 conf_tpa6130.h

Configuration of the external audio amplifier that is connected to the ABDAC (which pins are used and which configuration interface).

8.5.6 conf_usb.h

Configuration file used for the USB.

Some parameters are used for the configuration of the audio stream:

- USB_STREAM_BUFFER_SIZE: is the size (in bytes) of an audio buffer
- USB_STREAM_BUFFER_NUMBER: is the number of audio buffers
- USB_STREAM_RESYNC_PPM_STEPS: is the step (in PPM) used for the resynchronization.
- TIMER_USB_STREAM_MONITOR is the monitoring period (in ms) of the buffer indexes

8.6 Compiling the application

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

8.6.1 If you are using Atmel AVR32 Studio

- The application is supported by the Software Framework plugin. Open it (“Framework” then “Select Drivers/Components/Services”) and simply follows the instructions.

8.6.2 If you are using the standalone GCC with the Atmel AVR UC3 GNU Toolchain

- Open a shell, go to the /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/<part>-<board>/GCC/ directory and type:
  make rebuild program run

8.6.3 If you are using IAR Embedded Workbench for Atmel AVR UC3

- Open IAR and load the associated IAR project of this application (located in the directory /SERVICES/USB/CLASS/AUDIO/EXAMPLES/EXAMPLE_DEVICE/<part>-<board>/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

8.7 Running the application

Connect the Atmel EVK1105 board using a mini-B plug to a PC. The firmware will enumerate in the USB device mode, as a stereo headset with microphone.

The PC will automatically recognize the new audio device, meaning that previous PC speaker configuration will no longer work. To listen audio, just open your favorite audio application (for example, Windows Media Player) and play music. The sound will be output on the stereo 3.5mm mini jack connector.

In order to listen to the microphone sound, follow the following instruction:

- Launch the Sound Recorder Windows application and start recording:

- Speak clearly in front of the microphone. You can see the data stream activity in the Sound Recorder application

9. Memory footprint

Table 9-1 summarizes the memory footprint with various configuration.

9.1 GCC

The GCC version used is version 4.3.2.

GCC options are -O3 -fsection-anchors -ffunction-sections options.

Linker options are -Wl,--direct-data,--gc-sections.

9.2 IAR

The IAR version used is version 3.20A.

IAR options are “High” (balanced).

9.3 Summary

Here is the memory used by the application under both compilers, for all projects.

9.3.1 Method A: Add/remove samples

Table 9-1. Memory footprint under GCC and IAR. Unit is in Kbyte [kB].

Designation	Internal flash	Internal RAM	Internal flash	Internal RAM
	GCC		IAR	
Flash, including:	83		60	
• 32-bit AVR logo	20.3		20.3	
• Fonts (8x8, 8x16)	2.3		2.3	
Initialized data		1.4		0.4
Uninitialized data		0.7		0,7
Stack		4		4
HEAP used		5.1		5
Total	**83**	**11.2**	**60**	**10.1**

9.3.2 Method B: Clock synthesizer

Table 9-2. Memory footprint under GCC and IAR. Unit is in Kbyte [kB].

Designation	Internal flash	Internal RAM	Internal flash	Internal RAM
	GCC		IAR	
Flash, including:	85		62	
• 32-bit AVR logo	20.3		20.3	
• Fonts (8x8, 8x16)	2.3		2.3	
Initialized data		1.4		0.4
Uninitialized data		0.7		0,7
Stack		4		4
HEAP used		5.1		5
Total	**85**	**11.2**	**62**	**10.1**

9.3.3 Method C: Adaptive resampling

Table 9-3. Memory footprint under GCC and IAR. Unit is in Kbyte [kB].

Designation	Internal flash	Internal RAM	Internal flash	Internal RAM
	GCC		IAR	
Flash, including:	91		67	
• 32-bit AVR logo	20.3		20.3	
• Fonts (8x8, 8x16)	2.3		2.3	
Initialized data		1.4		0.4
Uninitialized data		0.7		0,7
Stack		4		4
HEAP used		26		26
Total	**91**	**32.1**	**67**	**31.1**

10. FAQ

Q: How can I have the best audio quality (that is, without distortion)?

A: The best audio quality is achieved thanks to the use of an external clock synthesizer (for example, a CL CS2200-CP). In this situation, you have a perfect 1:1 audio stream transfer from the PC/MAC to the DAC, without any data loss. This method is used in high end consumer audio products.

Q: What about using a USB feedback endpoint for the synchronization?

This synchronization method is under study.

Q: Which USB audio class does this application note support?

The current software is using the USB audio class version 1. This ensures the best support of the application across the various Windows and MAC operating systems.

Q: Can I avoid to dynamically change the CPU frequency according to the played sampling frequency (SSC I^2S)?

The SSC module derives its clock internally only from the peripheral bus. When you want to change the SSC frequency, the whole system clock needs to be changed. There is a solution listed in Section 6.4 “Special consideration for the SSC module” on page 13.

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) 3-6417-0300
Fax: (+81) 3-6417-0370