Freescale Semiconductor, Inc.
Reference Manual

Document Number: USBSPRDRM
Rev. 0, 03/2016

USB Speakers Reference Manual

1. Introduction

This document explains the software architecture and implementation of the USB Speakers reference design.

The content of this document is oriented to any person interested in understanding the functionality of the USB Speakers reference design with basic knowledge of USB protocol and Inter-IC Sound (I^2S).

Contents

2. Product Brief

The USB Speakers reference design demonstrates the implementation of an audio application whose audio input is via USB and whose output is via a headphone amplifier. It also has the capability to control the audio volume. It features the KL46Z MCU which includes an ARM® Cortex® M0+ processor and a Dialog Semiconductor DA7212 ultra low-power audio codec.

2.1. Get to know the KL46Z

The FRDM-KL46Z is an ultra-low-cost development platform enabled by the Kinetis L series KL4x MCU family built on the ARM Cortex M0+ processor. This device has a maximum operation frequency of 48 MHz, 256 KB of flash, 32 KB RAM, a full-speed USB controller, segment LCD controller, and many analog and digital peripherals.

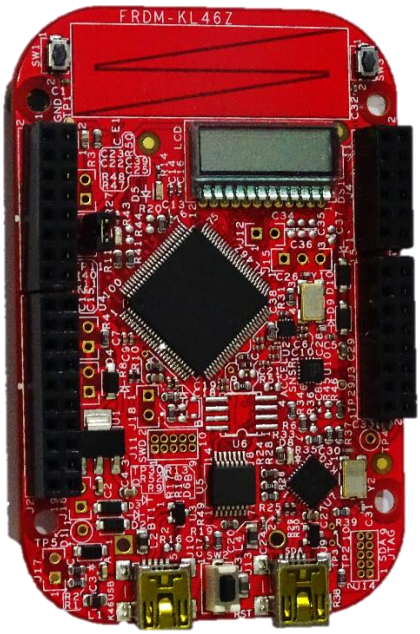

Figure 1. FRDM-KL46Z board

The FRDM-KL46Z features the NXP open standard embedded serial and debug adapter known as OpenSDA. This circuit offers several options for serial communications, flash programming, and run-control debugging.

2.2. Get to know the Dialog Semiconductor DA7212

The Dialog Semiconductor DA7212 ultra low-power audio codec is an ‘audio shield’ that features a two-channel audio codec with capless headphone drive and 3.5 mm stereo AUX input connector, Interchangeable GND and MIC routing to the smartphone jack allows multiple headsets to be supported, compatible with the *NXP Freedom Development Platform*.

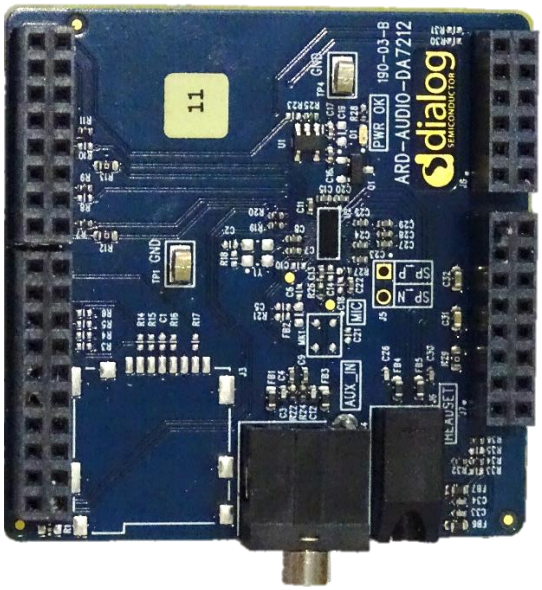

Figure 2. ARD-AUDIO-7212

DA7212 is ideal for standalone audio/video digital processors, controlled via the MCU platform processor over the I^2C serial communication interface. Digital audio is transmitted and received over the I^2S interface.

3. Functional Description

This chapter describes the device behavior across all its functional phases. It comprises all the required information to fully understand the application functionality. A general diagram is shown in *Figure 3*.

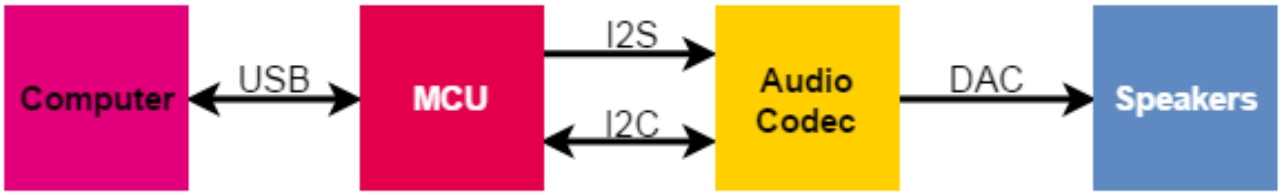

Figure 3. General diagram of the application.

The SW follows the layered scheme shown in *Figure 4*.

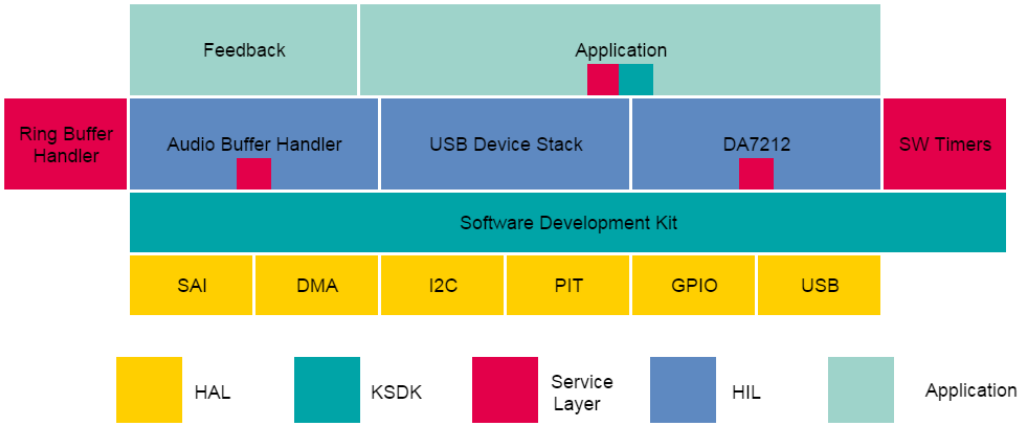

Figure 4. USB Speakers layered scheme

3.1. DA7212 audio codec

To configure the audio codec with I^2C, a driver using a finite state machine (FSM) was implemented. In this way all the register sequence initialization are written with a non-blocking communication. The states diagram of the FSM is shown in *Figure 5*.

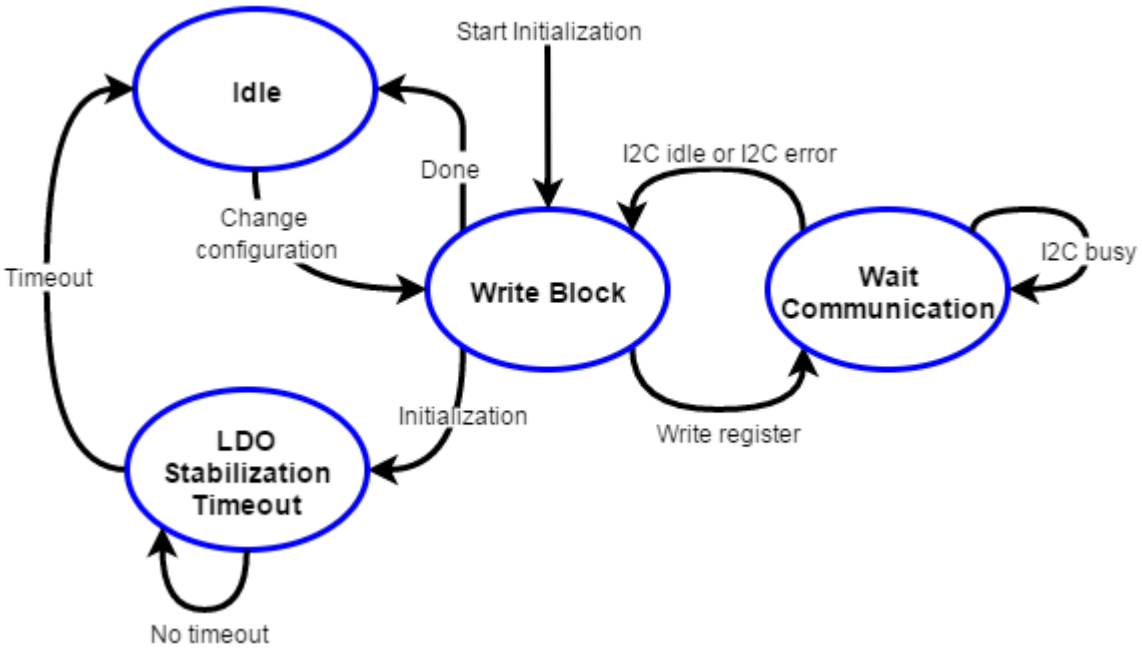

Figure 5. Codec communication states diagram

The DA7212 is configured to be the I^2S master. This means it provides the Bit clock (also known as BCLK or SCLK) and the Frame clock (also known as Frame synch or LRCLK). The audio codec PLL is used to generate the proper clocks based on the 8 MHz crystal oscillator from the FRDM-KL46Z.

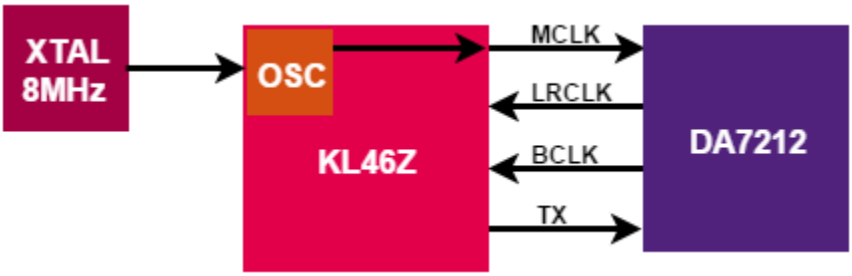

Figure 6. I^2S clock configuration

Volume control is handled through the audio codec and also by changing the headphone amplifier gain based on USB host requests.

3.2. USB communication

A USB composite device is implemented to provide a USB Audio class 1.0 and HID interfaces.

3.2.1. USB audio

A USB audio device class v1.0 interface is used to declare a speaker with the following characteristics:

- Master controls for volume and mute
- Two bytes frames

- 16-bit samples
- One sampling frequency: 48 KHz
- Two channels

For clock synchronization, an asynchronous with explicit feedback method is implemented. To achieve this, besides the audio endpoint, a feedback endpoint with a refresh rate of 8 ms is declared. More information on synchronization methods is provided in section 4.1 *Feedback*.

The audio endpoint size is calculated with the system sampling rate, amount of channels, and sample size in bytes:

$$EndpointSize = \frac{SamplingRate * Channels * SampleInBytes}{1000}$$

In this application, the endpoint size is 192 bytes:

$$EndpointSize = \left(\frac{48000 * 2 * 2}{1000}\right) = 192$$

As the asynchronous method is used, the host might send 1 more sample or 1 less sample. In order to accommodate the extra sample, the endpoint size must be increased by a sample per channel:

$$EndPointSize = \left(\frac{SamplingRate * Channels * SampleInBytes}{1000}\right) + (Channels * SampleInBytes) = 196$$

With this formula we can calculate the final endpoint size to 196 bytes.

The USB host sends a packet with audio every 1 ms. The application is notified of a new packet though a callback with the amount of bytes received.

An USB audio device can receive notification from the host to change the current volume. For this purpose, the device must declare its audio capabilities, including maximum, minimum and resolution. Table 1 shows the volume settings for the USB Speakers reference design. These are based on the audio codec capabilities.

Table 1. Volume settings

Parameter	Value in dB	USB code
Min Volume	-57	0xC700
Max Volume	+6	0x0600
Resolution	1	0x0100

The USB host updates the volume through a request “SetCurVolume”. The volume parameter range from +127.9961 dB (0x7FFF) to -127.9961 dB (0x8001) in 0.00390625 dB (0x0001) steps. The application must translate USB dB codes to its own format to update the volume. In the USB Speakers reference design, this translation is a simple liner interpolation between the USB code and the DA7212 audio codec volume settings.

Mute control is handled by “SetCurMute” request. The parameter is a boolean value, muted when TRUE and not muted when FALSE.

The USB Audio device class requires the devices to declare a zero bandwidth interface setting and an active setting. The zero bandwidth is used when no audio is being streamed. Whenever the host starts sending audio, it firstly changes the interface setting from zero bandwidth to the active setting. The application is notified of these changes through a callback to perform proper configuration changes.

3.2.2. USB HID

A HID Keyboard interface is available. The following figure shows the report descriptor used in the application.

```
USAGE_PAGE (Consumer Devices)          05 0C
USAGE (Consumer Control)               09 01
COLLECTION (Application)               A1 01
  LOGICAL_MINIMUM (0)                  15 00
  LOGICAL_MAXIMUM (1)                  25 01
  USAGE_PAGE (Consumer Devices)        05 0C
  USAGE (Volume Up)                    09 E9
  USAGE (Volume Down)                  09 EA
  USAGE (Mute)                         09 E2
  REPORT_COUNT (3)                     95 03
  REPORT_SIZE (1)                      75 01
  INPUT (Data,Var,Abs)                 81 02
  REPORT_SIZE (5)                      75 05
  REPORT_COUNT (1)                     95 01
  INPUT (Cnst,Var,Abs)                 81 03
END_COLLECTION                         C0
```

Figure 7. Report Descriptor

The report descriptor implements three bit entries, volume up, down, and mute. To get a full byte, the remaining 5 bits are declared as constants. Each option is represented by a bit offset.

Table 2. Report bits representation

Bit Offset	Description
0	Volume increment
1	Volume decrement
2	Mute
3:7	Unused

The current application only uses volume up and down. If a volume change is required, the corresponding bit is set to ‘1’ on the HID report and sent to the host. When the volume change is no longer required, the bit is set to ‘0’ and the report is sent to the host.

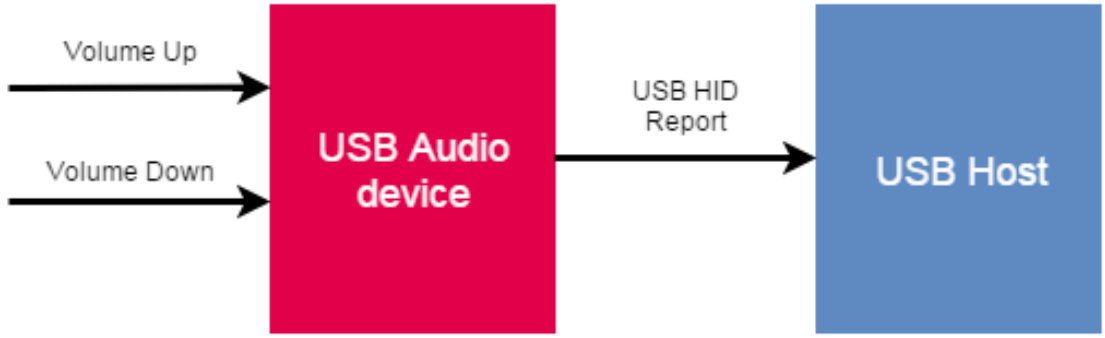

Figure 8. Report volume change

3.3. Audio buffer handler

This layer implements a circular buffer large enough to store approximately 10 ms of audio. Audio output is handled by a DMA channel based on I^2S transmitter request. An interrupt is generated when the DMA channel has moved 1 ms of samples, allowing synchronization methods to use it as a reference.

When the DMA source pointer reaches the end of the buffer, it automatically returns to the beginning due to the DMA circular buffer feature. This method works for power-of-2 size buffers only. To meet this requirement, the buffer size is set to 2048 bytes.

To allow faster processing, a second DMA channel is used to configure the first DMA channel every time the byte count reaches zero, using the link channel feature.

When the circular buffer is half filled, the playback starts, enabling I^2S and the DMA channel. The application must keep feeding the buffer with audio samples or overrun/underrun can occur.

If the audio playback is no longer required, the I^2S and DMA channel can be disabled and the circular buffer is reset to initial values.

3.4. Software timers

The application requires some tasks to execute periodically. To enable this, a software timer service was implemented with a resolution of 1 ms.

The application can request a timer, start it, or stop it at any time. When the programmed timeout expires, a registered callback is executed and the timeout is reloaded automatically.

3.5. KSDK

Kinetis Software Development Kit v1.3 is used for this application. The following KSDK blocks are used:

- USB device stack
- OS Abstraction layer
- Non-preemptive scheduler
- Peripheral drivers
  - SAI
  - DMA
  - I^2C
  - PIT
  - GPIO

3.5.1. USB Device Stack

In order to implement all features, a few configuration changes on the USB device stack were required:

Table 3. USB stack changes

USB Stack Module	Changes
Audio class	Modified Maximum allowed endpoints for the class
Device configuration	Enabled advance suspend handling, used for detach detection
Framework	Implemented advance suspend event

All the source files with changes are provided. Follow the user's guide steps to replace the current stack files with the modifications and successfully run the application.

4. Application Description

There are 4 tasks created at application initialization:

- Keyboard
- USB Audio data
- Audio codec task
- Software Timers

The keyboard task is in charge of sending the proper HID report to the host, requesting an increase or decrease in volume. To achieve this, both pushbuttons generate an interrupt when pressed and when released. On the interrupt service routine (ISR), an event is set to notify the task of either decrement or increment pushbutton action. The task processes the event and sends the report accordingly.

The USB audio data task handles the audio playback events. Every time a new audio packet is received, the USB class callback sets an event. The task takes the packet and sends it to the audio buffer handler for playback. This task also handles stop playback events due to device detach or host selecting the zero bandwidth setting.

The audio codec task is in charge of executing the DA7212 state machine driver. Each volume and mute request from the USB host sets events polled by this task, updating the audio codec settings as per the request.

The SW Timers task is to keep the timers counting and execute the callbacks when each timer reaches the programmed timeout.

4.1. Feedback

To avoid audio buffer underrun/overrun due to clocks mismatch between the MCU and the USB host, a synchronization method must be implemented.

The USB specification provides 3 methods to achieve synchronization for audio sink:

- Synchronous: Locks the audio clock to the USB Start Of Frame
- Adaptive: Recovers the audio clock from the data rate and adjusts the rate
- Asynchronous: Free running clocks use explicit feedback to adjust the samples sent

Implementing the first two methods might require external tunable PLL to move the clocks and match the host rate or asynchronous sample rate conversion algorithm to resample the audio to match the new rate, which is CPU intensive.

For the third method, the sink tells the host its current data rate, then, the host re-samples the audio to match that new rate (sending more or less samples). This explicit feedback allows low cost implementation without the need for dedicated external PLL or extra CPU usage.

The application sends feedback data every 8 ms. To calculate the feedback value, a simple buffer level status algorithm is used. At initialization, the feedback algorithm is configured with upper and lower limits, audio sample rate, buffer size, and feedback refresh rate. Based on this information, the algorithm calculates the minimum change possible for the sample rate and sets the upper and lower limits. The feedback value is sent as a 10.14 fractional number as per the USB 2.0 specification.

Every 8 ms, the algorithm checks the buffer status, if is below the lower limit, the application requests more samples, if it is above the upper limit, the application request less samples.

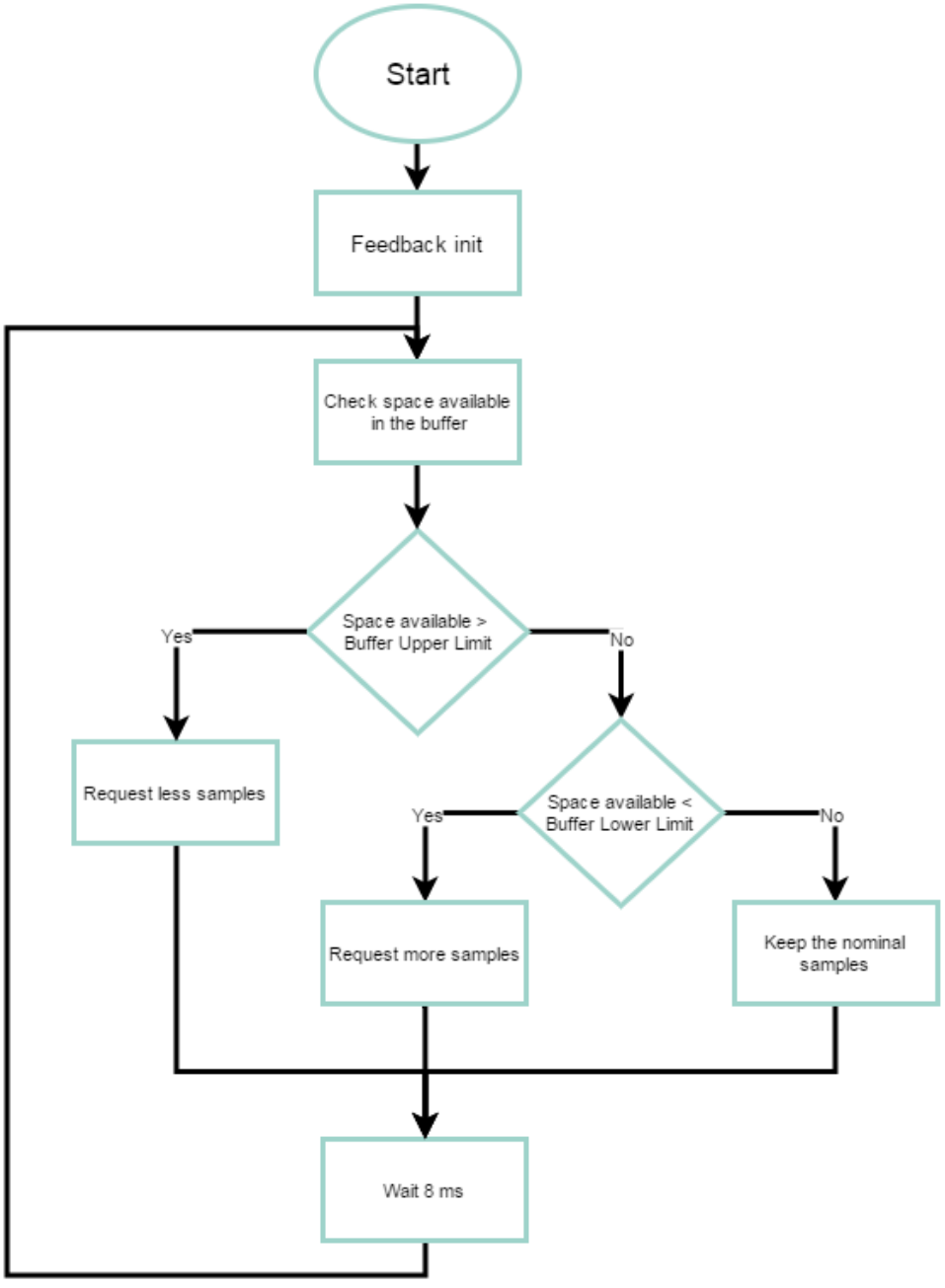

Figure 9. Feedback flow chart

5. Revision History

Table 4. Revision history

Revision number	Date	Substantive changes
0	03/2016	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Document Number: USBSPRDRM
Rev. 0
03/2016