RESEARCH ARTICLE OPEN ACCESS

USB Audio::An upgrade of general Multimedia speakers, generally of Laptop/Computer.

NIKHIL
Electronics and Communication
National Institute of Technology
Jamshedpur, INDIA

Date of Submission: 10-11-2023 Date of acceptance: 26-11-2023

***Abstract*—USB Audio: a standard for digital audio used in PCs, smart phones and tablets to interface with audio peripherals such as speakers, microphones, or mixing desks. In this article we set out to show how USB Audio works, what to watch out for, and how to use USB Audio for high-fidelity multi-channel input and output.**

***Keywords*—USB Protocol, Audio, Practical, CP2114**

I. INTRODUCTION

USB is a protocol where the PC, the *USB-host*, initiates a *transfer*, and the *device* (for example a USB speaker) responds. Each transfer is addressed to a specific device, and to a specific *endpoint* on the device. *IN-transfers* send data to the PC. When the host initiates an IN-transfer the device has to respond with data for the host. OUT-transfers send data to the device.

When the host performs an *OUT-transfer* it sends a packet of data that the device must capture. In the world of USB Audio, IN and OUT transfers may be used to transport audio samples: an OUT-transfer to send audio data from a PC to a speaker, whereas an IN-transfer is used to send audio data from a microphone to the PC.

There are four sorts of IN and OUT-transfers in USB: *Bulk*, *Isochronous*, *Interrupt*, and *Control* transfers.

A bulk transfer is used to *reliably* transfer data between host and device. All USB transfers carry a CRC (checksum) that indicates whether an error has occurred. On a bulk transfer, the receiver of the data has to verify the CRC. If the CRC is correct the transfer is acknowledged, and the data is assumed to have been transferred error-free. If the CRC is not correct, the transfer is not acknowledged and will be retried.

If the device is not ready to accept data it can send a negative-acknowledgment, *NAK*, which will cause the host to retry the transfer. Bulk transfers are not considered time criticial, and are scheduled around the time critical transfers discussed below.

Isochronous transfers are used to transfer data in *real-time* between host and device. When an isochronous endpoint is set up by the host, the host allocates a specific amount of bandwidth to the isochronous endpoint, and it regularly performs an IN- or OUT-transfer on that endpoint. For example, the host may OUT 1 KByte of data every 125 us to the device. Since a fixed and limited amount of bandwidth has been allocated, there is no time to resend data if anything goes wrong. The data has a CRC as normal, but if the receiving side detects an error there is no resend mechanism.

Interrupt transfers are used by the host to regularly poll the device to find out whether something worthwhile has happened. For example, a host may poll an audio device to check whether the MUTE button has been pressed. The name *Interrupt* transfer is slightly confusing, since they do not interrupt anything. However, regular polling of data gives the same sort of functionality that an host-interrupt would provide.

Control transfers are very much like bulk transfers. Control transfers are acknowledged, can be NAKed, and are delivered in a non-real-time fashion. Control transfers are used for operations that are outside normal data flow, such as querying the device capabilities, or endpoint status. An explanation on how device capabilities are described is outside the scope of this article, and we just state that there are predefined *classes* such as ‘USB Audio Class’ or ‘USB Mass Storage Class’ that enable cross platform interoperability.

All transfers are made in USB *frames*. High Speed USB frames span 125 us (Full Speed USB are 1 ms) and are marked by the host sending a Start-Of-Frame (SOF) message. Isochronous and Interrupt transfers are transmitted at most once a frame.

II. USB AUDIO

1. USB Audio uses isochronous, interrupt and control transfers. All audio data is transferred over isochronous transfers; interrupt transfers are used to relay information regarding the availability of audio clocks; control transfers are used used to set volume, request sample rates, etc. These are shown in Figure 1.

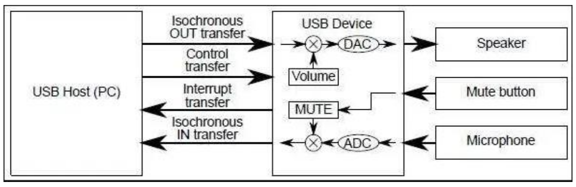


Figure 1: Transfers between a host and a USB device: Isochronous IN and OUT for audio-data, Control for setting parameters, and Interrupt for status monitoring.

The data requirements of a USB Audio system depends on the number of *channels* , the number of *bits* to represent each sample, and the *sample rate* . Typical channel counts are 2 (stereo), 6 (5.1) or much higher for studio and DJ use. Typically sample size is 24 bits, although 16 bits is available for legacy audio, and 32 bits for high quality audio. Typical sample rates are 44.1, 48, 96, and 192 kHz. The latter is used for high quality audio.

Suppose that we design a stereo audio speaker with a 96 kHz sample rate and 24-bit samples. In order to simplify data marshalling on host and device, 24-bit values are typically padded with a zero byte, so the total data throughput is 96,000 x 2 channels x 4 bytes = 768,000 bytes per second. The isochronous endpoints run at a rate of one transfer per 125 us; or 8,000 transfers per second. Dividing the required byte rate over the frame rate gives us the number of bytes for each isochronous transfer: 768,000/8,000 = 96 bytes per transfer.

When using CD rates, such as 44,100 Hz, the transfer rate works out as 44.1 transfers per second. In USB Audio each transfer always carries a whole number of samples; alternating transfers carry 48 and 40 bytes (6 and 5 stereo samples), so that the average rate works out as 44.1 bytes per transfer.

A single isochronous transfer can carry 1024 bytes, and can carry at most 256 samples (at 24/32 bits). This means that a single isochronous endpoint can transfer 42 channels at 48 kHz, or 10 channels at 192 kHz (assuming that High Speed USB is used – Full Speed USB cannot carry more than a single stereo IN and OUT pair at 48 kHz).

When transmitting digital audio, latency is introduced. In the case of High Speed USB this latency is 250 us. A packet of data is transferred once in every 125 us window, but given that it may be sent anytime in this window a 250 us buffer is required. On top of this 250 us delay, extra delay may be incurred in the O/S driver, and in the CODEC. Note that Full Speed USB has a much higher intrinsic latency of 2 ms, as data is only sent once in every 1 ms window.

III. ADVANTAGE

[1] USB Audio Class 2.0 takes advantage of High Speed USB 2.0, enabling low latency transfer of audio between PC and a connected audio device. The high throughput of High Speed USB 2.0 can be utilised to deliver many audio channels, and with high audio quality. The USB Audio Class standard caters for a wide range of devices, from complex mixing desks with many channels, multiple clock sources and complex controls, to surround sound systems, PC speakers and microphones.

IV. USB AUDIO SIMPLIFIED

[2] The rapid expansion of the universal serial bus (USB) standard in consumer electronics products has extended the use of USB connectivity to propagate and control digital audio. USB provides ample bandwidth to support high-quality audio; its ease of use has been well accepted by consumers and has made USB a popular audio interface. However, extracting the audio data from a USB port is not a simple task. USB itself is a complex protocol that requires considerable domain expertise.

In addition, other audio-related challenges, such as synchronization of data streams and programming codec and digital-to-analog converter (DAC) configurations, can challenge even the most experienced embedded and audio designers. USB bridge devices are now available that not only eliminate USB software development complexity but also provide a novel standard audio configuration interface and methods to synchronize audio data streams in a low-cost, highly-integrated single-chip solution.

USB is a versatile interface that provides many ways to propagate and control digital audio; however, it is important for the industry to follow a standardized mechanism for transporting audio over USB to secure interoperability, which has been the cornerstone for the adoption of USB. To respond to this fundamental request, the USB organization has developed the Audio Devices Class, which defines a very robust standardized mechanism for transporting audio over USB.

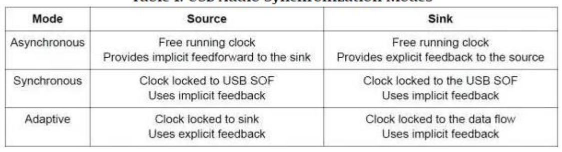
One of the major issues with streaming audio over USB is the synchronization of data streams from the host (source) to the device (sink); this has been addressed by developing a robust synchronization

scheme on “isochronous transfers,” which has been incorporated into the USB specification.
The Audio Device Class definition adheres to this synchronization scheme to transport audio data reliably over the bus. However, the implementation of this synchronization mechanism is not a trivial task, and legacy implementations have required high-end embedded systems with complex data rate converters or expensive phase-locked loops (PLLs) to support the clock accuracy demanded by the system.

In a system with a sampling rate of 48 kHz, the host sends a frame containing 48 analog output samples every millisecond. The sink must buffer the audio output data so it can be sent to the DAC one sample at a time. Any clock mismatch between host and device (however slight) will result in an overrun or underrun condition. The USB specification defines several methods for accommodating host/device clock mismatch.

USB defines modes that govern the operation of sources and sinks according to Table 1. (For audio-out, the host is the source and the device is the sink. For audio-in, the device is the source and the host is the sink.)

Table 1. USB Audio Synchronization Modes



Mode	Source	Sink
Asynchronous	Free running clock Provides implicit feedforward to the sink	Free running clock Provides explicit feedback to the source
Synchronous	Clock locked to USB SOF Uses implicit feedback	Clock locked to the USB SOF Uses implicit feedback
Adaptive	Clock locked to sink Uses explicit feedback	Clock locked to the data flow Uses implicit feedback

Asynchronous Mode

For asynchronous operation, the sink provides explicit feedback to the source. Based on this feedback, the source adjusts the number of samples that it sends to the sink. Figure 1 illustrates asynchronous mode with an analog output device.

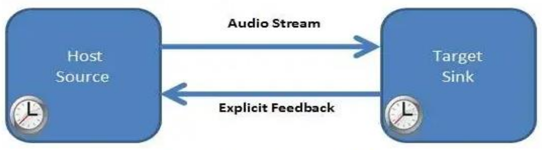


Figure 1: Asynchronous Mode

This feedback mechanism accommodates source/sink clock mismatch without requiring the sink device to implement PLL hardware to synchronize with the host clock.

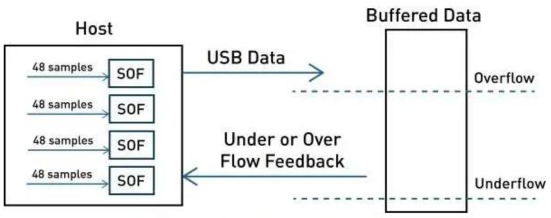


Figure 2. Buffered System to Support Asynchronous Mode

Figure 2 shows a buffered system for a 48 kHz sampling rate. Initially, the host starts streaming data at 48 samples every USB start-of-frame (SOF) operation, which occurs each millisecond. However, if the device’s buffer begins to approach the full or empty condition due to clock mismatch, the device can request that the host send more (49) or fewer (47) samples so that buffer overrun or underrun does not occur.

This method is implemented in Silicon Labs’ CP2114 USB-to-I2 S digital audio bridge device. The Audio Device Class is supported by the CP2114 device without any additional software development.

Synchronous Mode

For synchronous operation, the source and the sink use implicit feedback, and clocks are locked to the USB SOF. The sink device must synchronize with the USB SOF as shown in Figure 3.

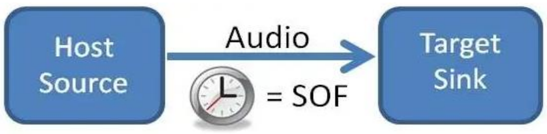


Figure 3. Synchronous Mode

A simple yet robust implementation of synchronous mode can be accomplished by a closed-loop control that can correct any mismatches from the USB SOF and the internal oscillator of the sink device. This implementation is shown in Figure 4.

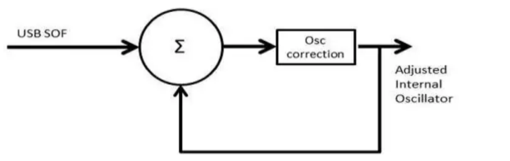


Figure 4. Closed-Loop Control to Support Synchronous Mode Using Internal Oscillator

The USB SOF that is sent by the host every millisecond is used to calibrate the internal oscillator. For this method to work properly, the internal oscillator of the sink device must be adjustable through a calibration register that can move the internal oscillator frequency up or down in very small steps. The CP2114 digital audio bridge device is able to implement this functionality due to the dynamic trim capability of its internal oscillator.

The CP2114 audio bridge enables the developer to select between synchronous and asynchronous modes depending on the host capabilities available in the system design. All popular platforms (Windows, Linux, Mac OS and iOS for the Apple iPad) now support asynchronous mode.

Standard Codec/DAC Configuration Interface

Today's leading codec and DAC suppliers provide unique ways to configure the capabilities of their devices. However, this variability in device configuration increases the complexity of software design for developers needing to support multiple codec/DAC platforms across their product lines.

A solution to this design challenge is to offer a standard codec/DAC configuration interface that can group the most typical capabilities to configure a codec or DAC. This interface would enable a smooth transition among codecs and DACs, and would enable quick evaluation of multiple codec/DAC options.

An example of this interface can be found in the CP2114 audio bridge, which supports a wide range of codecs/DACs using a standard configuration interface. Page 30 of the CP2114 datasheet contains a listing of the CP2114 standard audio configuration programming interface.

The standard programming interface of the CP2114 device enables the most common capabilities found in codecs and DACs, such as DAC register sizes, audio format, volume control and audio clock ratio. In addition, the interface offers open fields for custom programming and an abstraction layer encapsulating the most typical configuration capabilities in an easy-to-understand format. Once the developer is familiar with this interface, switching between codec and DAC devices becomes a simple task.

The CP2114 digital audio bridge provides access to this interface via USB to allocate all needed values to configure codecs or DACs. The configuration is applied once and resides in EPROM memory. Dynamic changes are also allowed from the host to dynamically access the codec/DAC and change its configuration values.

V. PRACTICAL IMPLEMENTATION

The full practical demonstration can be found here:

Link:1

Make USB only Speakers with No 3.5mm Jack | How to fix Laptop Sound Card at Home

Link:2
Make USB only SPEAKER

Link:3
USB Audio:: An upgrade of general Multimedia Speaker.

Materials required:

1. USB Sound Card

2. Multimedia speaker of Laptop/Computer

3. Soldering Iron/Solder/Flux

Note : Steps to be followed as per Video demonstration.

VI. CONCLUSION

The popularity of USB is extending its use to applications for propagating and controlling audio. However, streaming audio over USB is a complex and time-consuming design task. Major design issues, such as synchronization of audio data streams and codec/DAC configurations, can challenge even the most expert embedded and audio designers.

Digital audio bridges, such as the CP2114 device, minimize this complexity by providing a plug-and-play solution that does not require software development. Next-generation digital audio bridge solutions implement novel methods of supporting a wide range of codecs and DACs through a standard configuration interface, support asynchronous and synchronous modes of operation with minimal external components, and eliminate the need for external components, such as crystal oscillators and EEPROM. The USB audio class specification is available to the public from the USB Implementers Forum (www.usb.org).

AUTHOR
The author name is NIKHIL. He is currently DV Engineer in Qualcomm India Pvt. Ltd. For any query send eMail to laptopcomputermistri@gmail.com .
Linkedin(Click here)

ACKNOWLEDGMENT

The author wants to acknowledge yourself for glancing the paper and the author is thankful to People of NIT Jamshedpur. Thankful to Qualcomm India Pvt. Ltd. .

REFERENCES.

[1]. Pedro Pachuca manages Silicon Labs' global microcontroller (MCU) interface product business
[2]. Henk Muller is the Principal Technologist at XMOS Ltd.