USB Audio Simplified

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. The USB audio class specification is available to the public from the USB Implementers Forum (www.usb.org).

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.

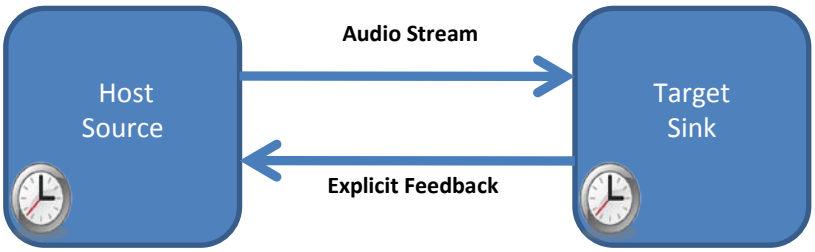

Figure 1. Asynchronous Mode

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

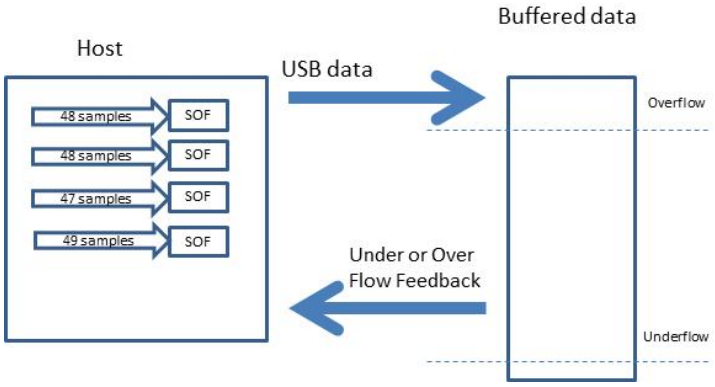

Figure 2. Buffered System to Support Asynchronous Mode

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-I2S 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.

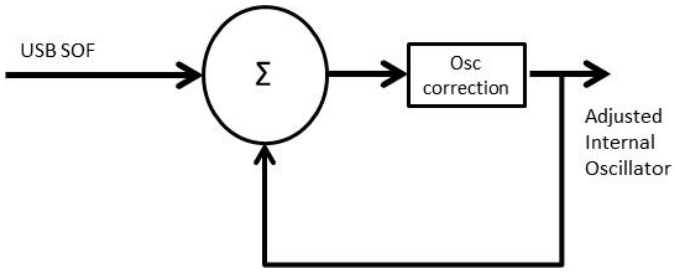

Figure 4. Closed-Loop Control to Support Synchronous Mode Using Internal Oscillator

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. Table 2 lists a portion of the CP2114 standard audio configuration programming interface.

Table 2. CP2114 Standard Audio Configuration Programming Interface

Byte	Name								
3	Audio_Props	Controls audio properties							
	Bit Position	7	6	5	4	3	2	1	0
	Bit Name	MB	ST	I2C_CK	I2C_PR	DRS	DVC	LJMS	AF
	MB	Mute Bit. 0: No affect 1: CP2114 will handle mute via mute bits at bytes 12,13,14,15 and 17							
	ST	Synchronization Type 0: Asynchronous. Will send feedback to USB host. 1: Synchronous. No feedback to USB host. Audio is synchronized via continuous clock adjustment of sample insert/drop, dependignon clock configuration.							
	I2C_CK	Maximum I2C clock rate supported by the DAC. 0: 400kHz 1: 100kHz							
	I2C_PR	I2C Protocol for read operations. 0: Stop 1: Repeated Start							
	DRS	DAC Register Size 0: 8 bit 1: 16 bit							
	DVC	DAC Volume Control. 0: No volume control supported by DAC 1: Volume control supported by DAC If set, the CP2114 populates volume control in the feature unit USB descriptor. If clear, 0 is specified in volume control to prevent the host from sending SET_CUR requests.							
	LJMS	I2S Mode. Only applies if using Left Justified format. 0: 16bit Left Justified Mode. 1: 24bit Left Justified Mode							
	AF	Audio Format 0: I2S format 1: Left Justified format							
4	Min_Volume	Minimum Volume in dB, 8bit signed. This corresponds to the volume control attribute MIN in USB Audio spec.							
5	Max_Volume	Maximum Volume in dB, 8bit signed. This corresponds to the volume control attribute MAX in USB Audio spec.							
6	Vol_Step	Volume Step Counts per dB. For instance, if volume resolution is 0.25 dB, 4 shall be written. A computed RES is returned in response to volume control attribute query of RES from the host.							

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.

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.

# # #