lib_xua

USB Audio Shared Components. For use in the XMOS USB Audio Refererence Designs.

This library enables the development of USB Audio devices on the XMOS xCORE architecture.

Features

Key features of the various applications in this repository are as follows

- USB Audio Class 1.0/2.0 Compliant
- Fully Asynchronous operation
- Support for the following sample frequencies: 8, 11.025, 12, 16, 32, 44.1, 48, 88.2, 96, 176.4, 192, 352.8, 384kHz
- Input/output channel and individual volume/mute controls supported
- Support for dynamically selectable output audio formats (e.g. resolution)
- Field firmware upgrade compliant to the USB Device Firmware Upgrade (DFU) Class Specification
- S/PDIF output
- S/PDIF input
- ADAT output
- ADAT input
- MIDI input/output (Compliant to USB Class Specification for MIDI devices)
- DSD output (Native and DoP mode) at DSD64 and DSD128 rates
- Mixer with flexible routing
- Simple playback controls via Human Interface Device (HID)
- Support for operation with Apple devices (requires software module sc_mfi for MFI licensees only - please contact XMOS)

Note, not all features may be supported at all sample frequencies, simultaneously or on all devices. Some features also require specific host driver support.

Software version and dependencies

This document pertains to version 0.2.0 of this library. It is known to work on version 14.3.2 of the xTIMEcomposer tools suite, it may work on other versions.

This library depends on the following other libraries:

- lib_xud (>=0.1.0)
- lib_spdif
- lib_logging (>=2.1.0)

Related Application Notes

The following application notes use this library:

- AN000246 - Simple USB Audio Device using lib_xua

1 About This Document

This document describes the structure of the library, its use and resources required. It also covers some implementation detail.

This document assumes familiarity with the XMOS xCORE architecture, the Universal Serial Bus 2.0 Specification (and related specifications), the XMOS tool chain and XC language.

2 Host System Requirements

USB Audio devices built using *lib_xua* have the following host system requirements.

- Mac OSX version 10.6 or later
- Windows Vista, 7, 8 or 10 with Thesycon Audio Class 2.0 driver for Windows (Tested against version 3.20). Please contact XMOS for details.
- Windows Vista, 7, 8 or 10 with built-in USB Audio Class 1.0 driver.

Older versions of Windows are not guaranteed to operate as expected. Devices are also expected to operate with various Linux distributions including mobile variants.

3 Overview

Functionality	
Provides USB interface to audio I/O.	
Supported Standards	
USB	USB 2.0 (Full-speed and High-speed) USB Audio Class 1.0[1] USB Audio Class 2.0[2] USB Firmware Upgrade (DFU) 1.1[3] USB Midi Device Class 1.0[4]
Audio	I2S/TDM S/PDIF ADAT Direct Stream Digital (DSD) PDM Microphones MIDI
Supported Sample Frequencies	
44.1kHz, 48kHz, 88.2kHz, 96kHz, 176.4kHz, 192kHz, 352.8kHz, 384kHz	
Supported Devices	
XMOS Devices	xCORE-200 Series
Requirements	
Development Tools	xTIMEcomposer Development Tools v14 or later
USB	xCORE-200 Series device with integrated USB Phy
Audio	External audio DAC/ADC/CODECs (and required supporting componentry) supporting I2S/TDM
Boot/Storage	Compatible SPI Flash device (or xCORE-200 device with internal flash)
Licensing and Support	
Reference code provided without charge under license from XMOS. Please visit http://www.xmos.com/support/contact for support. Reference code is maintained by XMOS Limited.	

[1] http://www.usb.org/developers/devclass_docs/audio10.pdf
[2] http://www.usb.org/developers/devclass_docs/Audio2.0_final.zip
[3] http://www.usb.org/developers/devclass_docs/DFU_1.1.pdf
[4] http://www.usb.org/developers/devclass_docs/midi10.pdf

4 Hardware Platforms

A range of hardware platforms for evaluating USB Audio on XMOS devices.

Specific, in depth, details for each platform/board are out of scope of this library documentation however, the features of the most popular platform are described below with the view of providing a worked example.

Please also see application note AN00246.

4.1 xCORE-200 Multi-Channel Audio Board

The XMOS xCORE-200 Multi-channel Audio board[5] (XK-AUDIO-216-MC) is a complete hardware and reference software platform targeted at up to 32-channel USB and networked audio applications, such as DJ decks and mixers.

The Multichannel Audio Platform hardware is based around the XE216-512-TQ128 multicore microcontroller; an dual-tile xCORE-200 device with an integrated High Speed USB 2.0 PHY, RGMII (Gigabit Ethernet) interface and 16 logical cores delivering up to 2000MIPS of deterministic and responsive processing power.

Exploiting the flexible programmability of the xCORE-200 architecture, the Multi-channel Audio Platform supports either USB or network audio source, streaming 8 analogue input and 8 analogue output audio channels simultaneously - at up to 192kHz.

For full details regarding the hardware please refer to xCORE-200 Multichannel Audio Platform Hardware Manual[6].

The reference board has an associated firmware application that uses *lib_xua* to implemented a USB Audio Devicce. Full details of this application can be found in the USB Audio Design Guide.

4.1.1 Analogue Input & Output

A total of eight single-ended analog input channels are provided via 3.5mm stereo jacks. Each is fed into a CirrusLogic CS5368 ADC. Similarly a total of eight single-ended analog output channels are provided. Each is fed into a CirrusLogic CS4384 DAC.

The four digital I2S/TDM input and output channels are mapped to the xCORE input/outputs through a header array. This jumper allows channel selection when the ADC/DAC is used in TDM mode

4.1.2 Digital Input & Output

Optical and coaxial digital audio transmitters are used to provide digital audio input output in formats such as IEC60958 consumer mode (S/PDIF) and ADAT. The output data streams from the xCORE-200 are re-clocked using the external master clock to synchronise the data into the audio clock domain. This is achieved using simple external D-type flip-flops.

4.1.3 MIDI

MIDI I/O is provided on the board via standard 5-pin DIN connectors. The signals are buffered using 5V line drivers and are then connected to 1-bit ports on the xCORE-200, via a 5V to 3.3V buffer.

[5] https://www.xmos.com/support/boards?product=18334
[6] https://www.xmos.com/support/boards?product=18334&component=18687

4.1.4 Audio Clocking

A flexible clocking scheme is provided for both audio and other system services. In order to accommodate a multitude of clocking options, the low-jitter master clock is generated locally using a frequency multiplier PLL chip. The chip used is a Phaselink PL611-01, which is pre-programmed to provide a 24MHz clock from its CLK0 output, and either 24.576 MHz or 22.5792MHz from its CLK1 output.

The 24MHz fixed output is provided to the xCORE-200 device as the main processor clock. It also provides the reference clock to a Cirrus Logic CS2100, which provides a very low jitter audio clock from a synchronisation signal provided from the xCORE-200.

Either the locally generated clock (from the PL611) or the recovered low jitter clock (from the CS2100) may be selected to clock the audio stages; the xCORE-200, the ADC/DAC and Digital output stages. Selection is conntrolled via an additional I/O, bit 5 of PORT 8C.

4.1.5 LEDs, Buttons and Other IO

An array of 4*4 green LEDs, 3 buttons and a switch are provided for general purpose user interfacing. The LED array is driven by eight signals each controlling one of 4 rows and 4 columns.

A standard XMOS xSYS interface is provided to allow host debug of the board via JTAG.

5 Software Overview

This section describes the software architecture of a USB Audio device implemented using *lib_xua*, its dependencies and other supporting libraries.

lib_xua provides fundamental building blocks for producing USB Audio products on XMOS devices. Every system is required to have the components from *lib_xua* listed in Table 1.

Component	Description
Endpoint 0	Provides the logic for Endpoint 0 which handles enumeration and control of the device including DFU related requests.
Endpoint buffer	Buffers endpoint data packets to and from the host. Manages delivery of audio packets between the endpoint buffer component and the audio components. It can also handle volume control processing.Note, this currently utlises two cores
AudioHub	Handles audio I/O over I2S and manages audio data to/from other digital audio I/O components.

Table 1: Required XUA Components

In addition low-level USB I/0 is required and is provided by the external dependency *lib_xud*

Component	Description
XMOS USB Device Driver (XUD)	Handles the low level USB I/O.

Table 2: Additional Components Required

In addition Table 3 shows optional components that can be added/enabled from within *lib_xua*

Component	Description
Mixer	Allows digital mixing of input and output channels. It can also handle volume control instead of the decoupler.
Clockgen	Drives an external frequency generator (PLL) and manages changes between internal clocks and external clocks arising from digital input.
MIDI	Outputs and inputs MIDI over a serial UART interface.

Table 3: Optional Components

lib_xua also provides optional support for integrating with the following eternal dependencies:

Component	Description
S/PDIF Transmitter (lib_spdif)	Outputs samples of an S/PDIF digital audio interface.
S/PDIF Receiver (lib_spdif)	Inputs samples of an S/PDIF digital audio interface (requires the clockgen component).
ADAT Receiver (lib_adat)	Inputs samples of an ADAT digital audio interface (requires the clockgen component).
PDM Microphones (lib_mic_array)	Receives PDM data from microphones and performs PDM to PCM conversion

Table 4: Optional Components

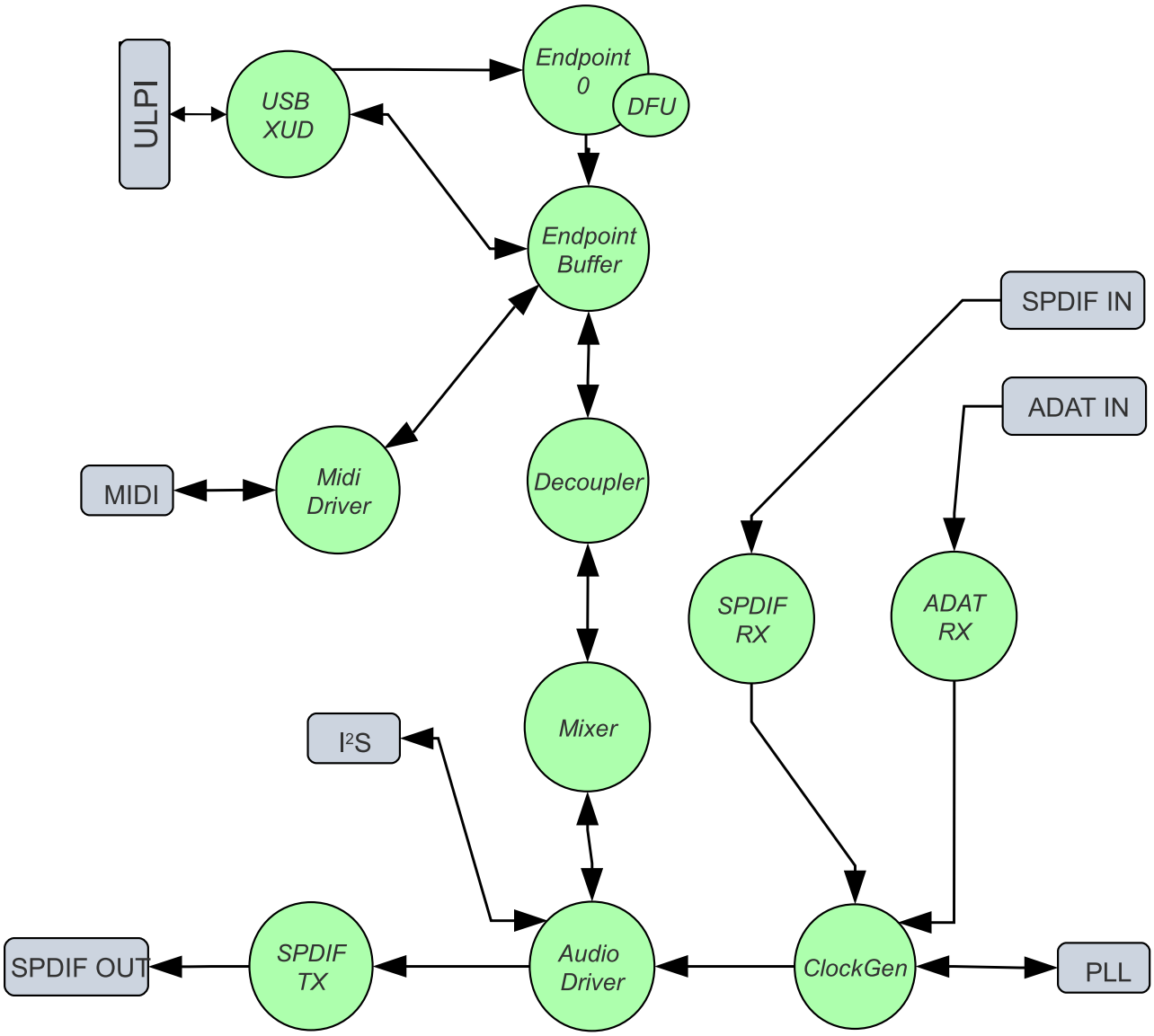

Figure 1: USB Audio Core Diagram

Figure 1 shows how the components interact with each other in a typical system. The green circles represent cores with arrows indicating inter-core communications.

6 Using lib_xua

This sections describes the basic usage of *lib_xud*. It provides a guide on how to program the USB Audio Devices using *lib_xud*.

Reviewing application note AN00246 is highly recommended at this point.

6.1 Library structure

The code is split into several directories.

core	Common code for USB audio applications
midi	MIDI I/O code
dfu	Device Firmware Upgrade code

Table 5: lib_xua structure

Note, the midi and dfu directories are potential candidates for separate libs in their own right.

6.2 Including in a project

All *lib_xua* functions can be accessed via the xua.h header filer:

```
#include <xua.h>
```

It is also required to add lib_xua to the USED_MODULES field of your application Makefile:

```
USED_MODULES = .. lib_xua ...
```

6.3 Core hardware resources

The user must declare and initialise relevant hardware resources (globally) and pass them to the relevant function of *lib_xua*.

As an absolute minimum the following resources are required:

- A 1-bit port for audio master clock input
- A n-bit port for internal feedback calculation (typically a free, unused port is used e.g. *16B*)
- A clock-block, which will be clocked from the master clock input port

Example declaration of these resources might look as follows:

```
in port p_mclk_in                   = PORT_MCLK_IN;
in port p_for_mclk_count            = PORT_MCLK_COUNT;   /* Extra port for counting master clock ticks */
clock clk_audio_mclk                = on tile[0]: XS1_CLKBLK_5;   /* Master clock */
```

The *PORT_MCLK_IN* and *PORT_MCLK_COUNT* defintions are derived from the projects XN file

The XUA_AudioHub() function requires an audio master clock input to clock the physical audio I/O. Less obvious is the reasoning for the XUA_Buffer() task having the same requirement - it is used for the USB feedback system and packet sizing.

Due to the above, if the XUD_AudioHub() and XUA_Buffer() cores must reside on separate tiles a separate master clock input port must be provided to each, for example:

```
/* Master clock for the audio IO tile */
in port p_mclk_in                  = PORT_MCLK_IN;

/* Resources for USB feedback */
in port p_mclk_in_usb              = PORT_MCLK_IN_USB;  /* Extra master clock input for the USB tile */
```

Whilst the hardware resources described in this section satisfy the basic requirements for the operation (or build) of *lib_xua* projects typically also needs some additional audio I/O, I2S or SPDIF for example.

These should be passed into the various cores as required - see API and Features sections.

6.4 Running the core components

In their most basic form the core components can be run as follows:

```
par
{
    /* Endpoint 0 core from lib_xua */
    XUA_Endpoint0(c_ep_out[0], c_ep_in[0], c_aud_ctl, null, null, null, null);

    /* Buffering cores - handles audio data to/from EP's and gives/gets data to/from the audio I/O core */
    /* Note, this spawns two cores */
    XUA_Buffer(c_ep_out[1], c_ep_in[1], c_sof, c_aud_ctl, p_for_mclk_count, c_aud);

    /* AudioHub/IO core does most of the audio IO i.e. I2S (also serves as a hub for all audio) */
    XUA_AudioHub(c_aud, ...) ;
}
```

XUA_Buffer() expects its p_for_mclk_count argument to be clocked from the audio master clock before being passed it. The following code satisfies this requirement:

```
{
        /* Connect master-clock clock-block to clock-block pin */
        set_clock_src(clk_audio_mclk_usb, p_mclk_in_usb);           /* Clock clock-block from mclk pin */
        set_port_clock(p_for_mclk_count, clk_audio_mclk_usb);       /* Clock the "count" port from the clock
          ↪ block */
        start_clock(clk_audio_mclk_usb);                            /* Set the clock off running */

        XUA_Buffer(c_ep_out[1], c_ep_in[1], c_sof, c_aud_ctl, p_for_mclk_count, c_aud);

}
```

Keeping this configuration outside of XUA_Buffer() does not preclude the possiblity of sharing p_mclk_in_usb port with additional components

To produce a fully operating device a call to XUD_Main() (from lib_xud) must also be made for USB connectivity:

```
/* Low level USB device layer core */
on tile[1]: XUD_Main(c_ep_out, 2, c_ep_in, 2, c_sof, epTypeTableOut, epTypeTableIn, null, null, -1,
  ↪ XUD_SPEED_HS, XUD_PWR_SELF);
```

Additionally the required communication channels must also be declared:

```
/* Channel arrays for lib_xud */
chan c_ep_out[2];
chan c_ep_in[2];

/* Channel for communicating SOF notifications from XUD to the Buffering cores */
chan c_sof;

/* Channel for audio data between buffering cores and AudioHub/IO core */
chan c_aud;

/* Channel for communicating control messages from EP0 to the rest of the device (via the buffering cores) */
chan c_aud_ctl;
```

This section provides enough information to implement a skeleton program for a USB Audio device. When running the xCORE device will present itself as a USB Audio Class device on the bus.

6.5 Configuring XUA

Configuration of the various build time options of `lib_xua` is done via the optional header *xua_conf.h*. Such build time options include audio class version, sample rates, channel counts etc. Please see the API section for full listings.

The build system will automatically include the *xua_conf.h* header file as appropriate - the user should continue to include *xua.h* as previously directed. A simple example is shown below:

```
#ifndef _XUA_CONF_H_
#define _XUA_CONF_H_

/* Output channel count */
#define XUA_NUM_USB_CHAN_OUT (2)

/* Product string */
#define XUA_PRODUCT_STR_A2 "My Product"

#endif
```

6.6 User functions

To enable custom functionality, such as configuring external audio hardware, custom functionality on stream start/stop etc various user overridable functions are provided (see API section for full listings). The default implementations are empty.

6.7 Codeless programming model

Whilst it is possible to code a USB Audio device using the building blocks provided by *lib_xua* it is realised that this might not be desirable for some classes of customers or product.

For instance, some users may not have a large software development experience and simply want to customise some basic settings such as strings. Others may want to fully customise the implementation - adding additional functionality such as adding DSD or possibly only using a subset of the functions provided - just XUA_AudioHub, for example.

In addition, the large number of supported features can lead a large number of tasks, hardware resources, communication channels etc, requiring quite a lot of code to be authored for each product.

In order to cater for the former class of users, a “codeless” option is provided. Put simply, a file `main.xc` is provided which includes a pre-authored `main()` function along with all of the required hardware resource declarations. Code is generated based on the options provided in `xua_conf.h`

Using this development model the user simply must include a xua_conf.h with their settings and optionally implementations of any 'user functions' as desired. This, along with an XN file for their hardware platform, is all that is required to build a fully featured and functioning product.

This model also provides the benefit of a known-good, full codebase as a basis for a product.

This behaviour described in this section is the default behaviour of *lib_xua*, to disable this please set EXCLUDE_USB_AUDIO_MAIN to 1 in the application makefile or xua_conf.h.

7 Features

The previous sections describes only the basic core set of lib_xua details on enabling additional features e.g. S/PDIF are discussed in this section.

Where something must be defined, it is recommened this is done in *xua_conf.h* but could also be done in the application Makefile.

For each feature steps are listed for if calling lib_xua functions manually - if using the "codeless" programming model then these steps informational only. Each section also includes a sub-section on enabling the feature using the "codeless" model.

For full details of all options please see the API section

7.1 I2S/TDM

I2S/TDM is typically fundamental to most products and is built into the XUA_AudioHub() core.

In order to enable I2S on must declare an array of ports for the data-lines (one for each direction):

```
/* Port declarations. Note, the defines come from the xn file */
buffered out port:32 p_i2s_dac[]    = {PORT_I2S_DAC0};   /* I2S Data-line(s) */
buffered in port:32 p_i2s_adc[]     = {PORT_I2S_ADC0};   /* I2S Data-line(s) */
```

Ports for the sample and bit clocks are also required:

```
buffered out port:32 p_lrclk        = PORT_I2S_LRCLK;    /* I2S Bit-clock */
buffered out port:32 p_bclk         = PORT_I2S_BCLK;     /* I2S L/R-clock */
```

All of these ports must be buffered, width 32. Based on whether the xCORE is bus slave/master the ports must be declared as input/output respectively

These ports must then be passed to the XUA_AudioHub() task appropriately.

I2S functionality also requires two clock-blocks, one for bit and sample clock e.g.:

```
/* Clock-block declarations */
clock clk_audio_bclk                = on tile[0]: XS1_CLKBLK_4;   /* Bit clock */
clock clk_audio_mclk                = on tile[0]: XS1_CLKBLK_5;   /* Master clock */
```

These hardware resources must be passed into the call to XUA_AudioHub():

```
/* AudioHub/IO core does most of the audio IO i.e. I2S (also serves as a hub for all audio) */
on tile[0]: XUA_AudioHub(c_aud, clk_audio_mclk, clk_audio_bclk, p_mclk_in, p_lrclk, p_bclk);
```

7.1.1 Codeless Programming Model

All ports and hardware resources are already fully declared, one must simply set the following:

- *I2S_CHANS_DAC* must be set to the desired number of output channels via I2S
- *I2S_CHANS_ADC* must be set to the desired number of input channels via I2S
- *AUDIO_IO_TILE* must be set to the tile where the physical I2S connections reside

For configuration options, master vs slave, TDM etc please see the API section.

7.2 S/PDIF Transmit

lib_xua supports the development of devices with S/PDIF transmit functionality through the use of lib_spdif. The XMOS S/PDIF transmitter runs in a single core and supports rates up to 192kHz.

The S/PDIF transmitter core takes PCM audio samples via a channel and outputs them in S/PDIF format to a port. The channel should be declared a normal:

```
chan c_spdif_tx
```

Samples are provided to the S/PDIF transmitter task from the XUA_AudioHub() task.

In order to use the S/PDIF transmmiter with lib_xua hardware resources must be declared e.g:

```
buffered out port:32 p_spdif_tx         = PORT_SPDIF_OUT;     /* SPDIF transmit port */
```

This port should be clocked from the master-clock, lib_spdif provides a helper function for setting up the port:

```
spdif_tx_port_config(p_spdif_tx2, clk_audio_mclk, p_mclk_in, delay);
```

If sharing the master-clock port and clockblock with XUA_AudioHub() (or any other task) then this setup should be done before running the tasks in a par{}

Finally the S/PDIF transmitter task must be run - passing in the port and channel for communication with XUA_AudioHub. For example:

```
par
{
    while(1)
    {
        /* Run the S/PDIF transmitter task */
        spdif_tx(p_spdif_tx2, c_spdif_tx);
    }

    /* AudioHub/IO core does most of the audio IO i.e. I2S (also serves as a hub for all audio) */
    /* Note, since we are not using I2S we pass in null for LR and Bit clock ports and the I2S dataline ports
       ↪ */
    XUA_AudioHub(c_aud, clk_audio_mclk, null, p_mclk_in, null, null, null, null, c_spdif_tx);
}
```

For further details please see the documentation, application notes and examples provided for lib_spdif.

7.2.1 Codeless Programming Model

If using the codeless programming method one must simply ensure the following:

- *PORT_SPDIF_OUT* is correctly defined in the XN file
- *XUA_SPDIF_TX_EN* should be defined as non-zero
- *SPDIF_TX_TILE* is correctly defined (note, this defaults to *AUDIO_IO_TILE*)

For further configuration options please see the API section.

8 Software Detail

This section describes the software architecture of a USB Audio device implemented using *lib_xua*, it's dependancies and other supporting libraries.

This section will now examine the operation of these components in further detail.

8.1 AudioHub/I2S

The AudioHub task performs many functions. It receives and transmits samples from/to the decoupler or mixer core over an XC channel.

It also drives several in and out I2S/TDM channels to/from a CODEC, DAC, ADC etc - from now on termed "audio hardware".

If the firmware is configured with the xCORE as I2S master the requred clock lines will also be driven out from this task also.

It also has the task of formwarding on and reciving samples to/from other audio related tasks such as S/PDIF tasks, ADAT tasks etc.

The AudioHub task must be connected to external audio hardware that supports I2S (other modes such as "left justified" can be supported with firmware changes).

In master mode, the XMOS device acts as the master generating the I2S "Continous Serial Clock (SCK)" typically called the Bit-Clock (BCLK) and the "Word Select (WS)" line typically called left-right clock (LRCLK) signals. Any CODEC or DAC/ADC combination that supports I2S and can be used.

The LR-clock, bit-clock and data are all derived from the incoming master clock (typically the output of the external oscillator or PLL) - This is not part of the I2S standard but is commonly included for synchronizing the internal operation of the analog/digital converters.

The AudioHub task is implemented in the file `xua_audiohub.xc`.

Table 6 shows the signals used to communicate audio between the XMOS device and the external audio hardware.

Signal	Description
LRCLK	The word clock, transition at the start of a sample
BCLK	The bit clock, clocks data in and out
SDIN	Sample data in (from CODEC/ADC to the XMOS device)
SDOUT	Sample data out (from the XMOS device to CODEC/DAC)
MCLK	The master clock running the CODEC/DAC/ADC

Table 6: I2S Signals

The bit clock controls the rate at which data is transmitted to and from the external audio hardware.

In the case where the XMOS device is the master, it divides the MCLK to generate the required signals for both BCLK and LRCLK, with BCLK then being used to clock data in (SDIN) and data out (SDOUT) of the external audio hardware.

Table 7 shows some example clock frequencies and divides for different sample rates:

Sample Rate (kHz)	MCLK (MHz)	BCLK (MHz)	Divide
44.1	11.2896	2.819	4
88.2	11.2896	5.638	2
176.4	11.2896	11.2896	1
48	24.576	3.072	8
96	24.576	6.144	4
192	24.576	12.288	2

Table 7: Clock Divide examples

The master clock must be supplied by an external source e.g. clock generator, fixed oscillators, PLL etc to generate the two frequencies to support 44.1kHz and 48kHz audio frequencies (e.g. 11.2896/22.5792MHz and 12.288/24.576MHzrespectively). This master clock input is then provided to the external audio hardware and the xCORE device.

8.1.1 Port Configuration (xCORE Master)

The default software configuration is xCORE is I2S master. That is, the XMOS device provides the BCLK and LRCLK signals to the external audio hardware

xCORE ports and XMOS clocks provide many valuable features for implementing I2S. This section describes how these are configured and used to drive the I2S interface.

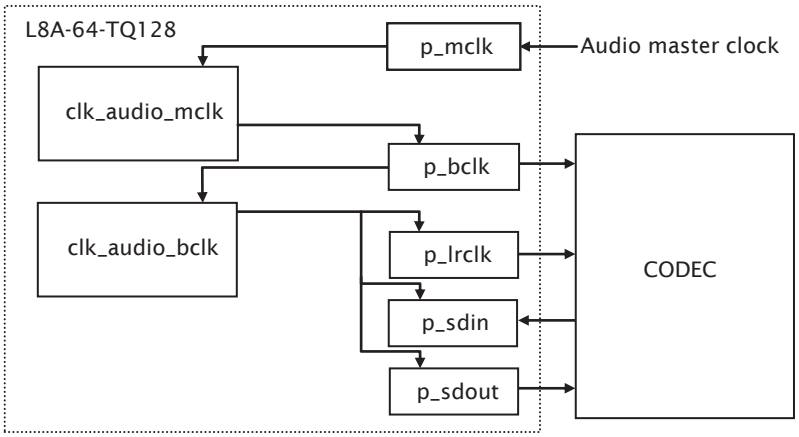

Figure 2: Ports and Clocks (xCORE master)

The code to configure the ports and clocks is in the `ConfigAudioPorts()` function. Developers should not need to modify this.

The xCORE inputs MCLK and divides it down to generate BCLK and LRCLK.

To achieve this MCLK is input into the device using the 1-bit port `p_mclk`. This is attached to the clock block `clk_audio_mclk`, which is in turn used to clock the BCLK port, `p_bclk`. BCLK is used to clock the LRCLK (`p_lrclk`) and data signals SDIN (`p_sdin`) and SDOUT (`p_sdout`).

Again, a clock block is used (`clk_audio_bclk`) which has `p_bclk` as its input and is used to clock the ports `p_lrclk`, `p_sdin` and `p_sdout`. The preceding diagram shows the connectivity of ports and clock blocks.

p_sdin and p_sdout are configured as buffered ports with a transfer width of 32, so all 32 bits are input in one input statement. This allows the software to input, process and output 32-bit words, whilst the ports serialize and deserialize to the single I/O pin connected to each port.

xCORE-200 series devices have the ability to divide an extenal clock in a clock-block.

However, XS1 based devices do not have this functionality. In order achieve the reqired master-clock to bit-clock/LR-clock divicd on XS1 devices, buffered ports with a transfer width of 32 are also used for p_bclk and p_lrclk. The bit clock is generated by performing outputs of a particular pattern to p_bclk to toggle the output at the desired rate. The pattern depends on the divide between the master-clock and bit-clock. The following table shows the required pattern for different values of this divide:

Divide	Output pattern	Outputs per sample
2	0xAAAAAAAA	2
4	0xCCCCCCCC	4
8	0xF0F0F0F0	8

Table 8: Output patterns

In any case, the bit clock outputs 32 clock cycles per sample. In the special case where the divide is 1 (i.e. the bit clock frequency equals the master clock frequency), the p_bclk port is set to a special mode where it simply outputs its clock input (i.e. p_mclk). See configure_port_clock_output() in xs1.h for details.

p_lrclk is clocked by p_bclk. In I2S mode the port outputs the pattern 0x7fffffff followed by 0x80000000 repeatedly. This gives a signal that has a transition one bit-clock before the data (as required by the I2S standard) and alternates between high and low for the left and right channels of audio.

8.1.2 Changing Audio Sample Frequency

When the host changes sample frequency, a new frequency is sent to the audio driver core by Endpoint 0 (via the buffering cores and mixer).

First, a change of sample frequency is reported by sending the new frequency over an XC channel. The audio core detects this by checking for the presence of a control token on the channel channel

Upon receiving the change of sample frequency request, the audio core stops the I2S/TDM interface and calls the CODEC/port configuration functions.

Once this is complete, the I2S/TDM interface (i.e. the main look in AudioHub) is restarted at the new frequency.

8.2 S/PDIF Transmit

lib_xua supports the development of devices with S/PDIF transmit throught the use of lib_spdif. The XMOS S/SPDIF transmitter component runs in a single core and supports sample-rates upto 192kHz.

The S/PDIF transmitter core takes PCM audio samples via a channel and outputs them in S/PDIF format to a port. A lookup table is used to encode the audio data into the required format.

It receives samples from the Audio I/O core two at a time (for left and right). For each sample, it performs a lookup on each byte, generating 16 bits of encoded data which it outputs to a port.

S/PDIF sends data in frames, each containing 192 samples of the left and right channels.

Audio samples are encapsulated into S/PDIF words (adding preamble, parity, channel status and validity bits) and transmitted in biphase-mark encoding (BMC) with respect to an *external* master clock.

Note that a minor change to the SpdifTransmitPortConfig function would enable *internal* master clock generation (e.g. when clock source is already locked to desired audio clock).

Sample frequencies	44.1, 48, 88.2, 96, 176.4, 192 kHz
Master clock ratios	128x, 256x, 512x
Library	lib_spdif

Table 9: S/PDIF Capabilities

8.2.1 Clocking

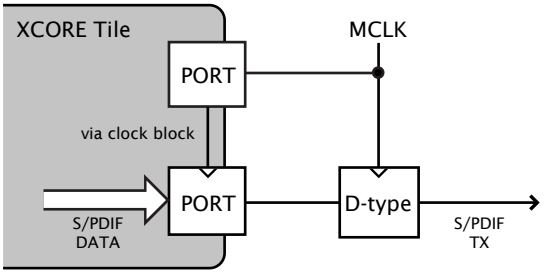

Figure 3: D-Type Jitter Reduction

The S/PDIF signal is output at a rate dictated by the external master clock. The master clock must be 1x 2x or 4x the BMC bit rate (that is 128x 256x or 512x audio sample rate, respectively). For example, the minimum master clock frequency for 192kHz is therefore 24.576MHz.

This resamples the master clock to its clock domain (oscillator), which introduces jitter of 2.5-5 ns on the S/PDIF signal. A typical jitter-reduction scheme is an external D-type flip-flop clocked from the master clock (as shown in the preceding diagram).

8.2.2 Usage

The interface to the S/PDIF transmitter core is via a normal channel with streaming built-ins (`outuint`, `inuint`). Data format should be 24-bit left-aligned in a 32-bit word: 0x12345600

The following protocol is used on the channel:

`outct`	New sample rate
`outuint`	Sample frequency (Hz)
`outuint`	Master clock frequency (Hz)
`outuint`	Left sample
`outuint`	Right sample
`outuint`	Left sample
`outuint`	Right sample
...	
...	

Table 10: S/PDIF Component Protocol

8.2.3 Output stream structure

The stream is composed of words with the following structure shown in Table 11. The channel status bits are 0x0nc07A4, where c=1 for left channel, c=2 for right channel and n indicates sampling frequency as shown in Table 12.

Bits		
0:3	Preamble	Correct B M W order, starting at sample 0
4:27	Audio sample	Top 24 bits of given word
28	Validity bit	Always 0
29	Subcode data (user bits)	Unused, set to 0
30	Channel status	See below
31	Parity	Correct parity across bits 4:30

Table 11: S/PDIF Stream Structure

Frequency (kHz)	n
44.1	0
48	2
88.2	8
96	A
176.4	C
192	E

Table 12: Channel Status Bits

9 Known Issues

- Quad-SPI DFU will corrupt the factory image with tools version < 14.0.4 due to an issue with libquad-flash
- (#14762) When in DSD mode with S/PDIF output enabled, DSD samples are transmitted over S/PDIF if the DSD and S/PDIF channels are shared, this may or may not be desired
- (#14173) I2S input is completely disabled when DSD output is active - any input stream to the host will contain 0 samples
- (#14780) Operating the design at a sample rate of less than or equal to the SOF rate (i.e. 8kHz at HS, 1kHz at FS) may expose a corner case relating to 0 length packet handling in both the driver and device and should be considered un-supported at this time.
- (#14883) Before DoP mode is detected a small number of DSD samples will be played out as PCM via I2S
- (#14887) Volume control settings currently affect samples in both DSD and PCM modes. This results in invalid DSD output if volume control not set to 0
- Windows XP volume control very sensitive. The Audio 1.0 driver built into Windows XP (usbaudio.sys) does not properly support master volume AND channel volume controls, leading to a very sensitive control. Descriptors can be easily modified to disable master volume control if required (one byte - bmaControls(0) in Feature Unit descriptors)
- 88.2kHz and 176.4kHz sample frequencies are not exposed in Windows control panels. These are known OS restrictions.

10 lib_xua Change Log

10.1 0.2.0

- ADDED: Initial library documentation
- ADDED: Application note AN00247: Using lib_xua with lib_spdif (transmit)
- CHANGE: I2S hardware resources no longer used globally and must be passed to XUA_AudioHub()
- CHANGE: NO_USB define renamed to XUA_USB_EN
- CHANGE: XUA_AudioHub() no longer pars S/PDIF transmitter task
- CHANGE: Moved to lib_spdif (from module_spdif_tx & module_spdif_rx)

10.2 0.1.2

- ADDED: Application note AN00246: Simple USB Audio Device using lib_xua
- CHANGE: xmosdfu emits warning if empty image read via upload
- CHANGE: Simplified mclk port sharing - no longer uses unsafe pointer
- RESOLVED: Runtime exception issues when incorrect feedback calculated (introduced in sc_usb_audio 6.13)
- RESOLVED: Output sample counter reset on stream start. Caused playback issues on some Linux based hosts

10.3 0.1.1

- RESOLVED: Configurations where I2S_CHANS_DAC and I2S_CHANS_ADC are both 0 now build
- RESOLVED: Deadlock in mixer when MAX_MIX_COUNT > 0 for larger channel counts
- Changes to dependencies:
  - lib_logging: Added dependency 2.1.1
  - lib_xud: Added dependency 0.1.0

10.4 0.1.0

- ADDED: FB_USE_REF_CLOCK to allow feedback generation from xCORE internal reference
- ADDED: Linux Makefile for xmosdfu host application
- ADDED: Raspberry Pi Makefile for xmosdfu host application
- ADDED: Documentation of PID argument to xmosdfu
- ADDED: Optional build time microphone delay line (MIC_BUFFER_DEPTH)
- CHANGE: Removal of audManage_if, users should define their own interfaces as required
- CHANGE: Vendor specific control interface in UAC1 descriptor now has a string descriptor so it shows up with a descriptive name in Windows Device Manager
- CHANGE: DFU_BCD_DEVICE removed (now uses BCD_DEVICE)
- CHANGE: Renaming in descriptors.h to avoid clashes with application
- CHANGE: Make device reboot function no-argument (was one channel end)
- RESOLVED: FIR gain compensation for PDM mics set incorrectly for divide of 8
- RESOLVED: Incorrect xmosdfu DYLD path in test script code
- RESOLVED: xmosdfu cannot find XMOS device on modern MacBook Pro (#17897)
- RESOLVED: Issue when feedback is initially incorrect when two SOF's are not yet received
- RESOLVED: AUDIO_TILE and PDM_TILE may now share the same value/tile
- RESOLVED: Cope with out of order interface numbers in xmosdfu
- RESOLVED: DSD playback not functional on xCORE-200 (introduced in sc_usb_audio 6.14)
- RESOLVED: Improvements made to clock sync code in TDM slave mode

10.5 Legacy release history

(Note: Forked from sc_usb_audio at this point)

10.6 7.4.1

- RESOLVED: Exception due to null chanend when using NO_USB

10.7 7.4.0

- RESOLVED: PID_DFU now based on AUDIO_CLASS. This potentially caused issues with UAC1 DFU

10.8 7.3.0

- CHANGE: Example OSX DFU host app updated to now take PID as runtime argument. This enabled multiple XMOS devices to be attached to the host during DFU process

10.9 7.2.0

- ADDED: DFU to UAC1 descriptors (guarded by DFU and FORCE_UAC1_DFU)
- RESOLVED: Removed 'reinterpretation to type of larger alignment' warnings
- RESOLVED: DFU flash code run on tile[0] even if XUD_TILE and AUDIO_IO_TILE are not 0

10.10 7.1.0

- ADDED: UserBufferManagementInit() to reset any state required in UserBufferManagement()
- ADDED: I2S output up-sampling (enabled when AUD_TO_USB_RATIO is > 1)
- ADDED: PDM Mic decimator output rate can now be controlled independently (via AUD_TO_MICS_RATIO)
- CHANGE: Rename I2S input down-sampling (enabled when AUD_TO_USB_RATIO is > 1, rather than via I2S_DOWNSAMPLE_FACTOR)
- RESOLVED: Crosstalk between input channels when I2S input down-sampling is enabled
- RESOLVED: Mic decimation data tables properly sized when mic sample-rate < USB audio sample-rate

10.11 7.0.1

- RESOLVED: PDM microphone decimation issue at some sample rates caused by integration

10.12 7.0.0

- ADDED: I2S down-sampling (I2S_DOWNSAMPLE_FACTOR)
- ADDED: I2S resynchronisation when in slave mode (CODEC_MASTER=1)
- CHANGE: Various memory optimisations when MAX_FREQ = MIN_FREQ
- CHANGE: Memory optimisations in audio buffering
- CHANGE: Various memory optimisations in UAC1 mode
- CHANGE: user_pdm_process() API change
- CHANGE: PDM Mic decimator table now related to MIN_FREQ (memory optimisation)
- RESOLVED: Audio request interrupt handler properly eliminated

10.13 6.30.0

- **RESOLVED: Number of PDM microphone channels configured now based on NUM_PDM_MICS define** (previously hard-coded)
- RESOLVED: PDM microphone clock divide now based MCLK defines (previously hard-coded)
- CHANGE: Second microphone decimation core only run if NUM_PDM_MICS > 4

10.14 6.20.0

- RESOLVED: Intra-frame sample delays of 1/2 samples on input streaming in TDM mode
- RESOLVED: Build issue with NUM_USB_CHAN_OUT set to 0 and MIXER enabled
- RESOLVED: SPDIF_TX_INDEX not defined build warning only emitted when SPDIF_TX defined
- RESOLVED: Failure to enter DFU mode when configured without input volume control

10.15 6.19.0

- RESOLVED: SPDIF_TX_INDEX not defined build warning only emitted when SPDIF_TX defined
- RESOLVED: Failure to enter DFU mode when configured without input volume control

10.16 6.18.1

- **ADDED: Vendor Specific control interface added to UAC1 descriptors to allow control of** XVSM params from Windows (via lib_usb)

10.17 6.18.0

- ADDED: Call to VendorRequests() and VendorRequests_Init() to Endpoint 0
- ADDED: VENDOR_REQUESTS_PARAMS define to allow for custom parameters to VendorRequest calls
- RESOLVED: FIR gain compensation set appropriately in lib_mic_array usage
- CHANGE: i_dsp interface renamed i_audManage

10.18 6.16.0

- ADDED: Call to UserBufferManagement()
- ADDED: PDM_MIC_INDEX in devicedefines.h and usage
- CHANGE: pdm_buffer() task now combinable
- CHANGE: Audio I/O task now takes i_dsp interface as a parameter
- CHANGE: Removed built-in support for A/U series internal ADC
- CHANGE: User PDM Microphone processing now uses an interface (previously function call)

10.19 6.15.2

- RESOLVED: interrupt.h (used in audio buffering) now compatible with xCORE-200 ABI

10.20 6.15.1

- RESOLVED: DAC data mis-alignment issue in TDM/I2S slave mode
- CHANGE: Updates to support API changes in lib_mic_array version 2.0

10.21 6.15.0

- **RESOLVED: UAC 1.0 descriptors now support multi-channel volume control (previously were** hard-coded as stereo)
- **CHANGE: Removed 32kHz sample-rate support when PDM microphones enabled (lib_mic_array** currently does not support non-integer decimation factors)

10.22 6.14.0

- **ADDED: Support for for master-clock/sample-rate divides that are not a power of 2** (i.e. 32kHz from 24.567MHz)
- **ADDED: Extended available sample-rate/master-clock ratios. Previous restriction was <=** 512x (i.e. could not support 1024x and above e.g. 49.152MHz MCLK for Sample Rates below 96kHz) (#13893)
- **ADDED: Support for various "low" sample rates (i.e. < 44100) into UAC 2.0 sample rate** list and UAC 1.0 descriptors
- **ADDED: Support for the use and integration of PDM microphones (including PDM to PCM** conversion) via lib_mic_array
- **RESOLVED: MIDI data not accepted after "sleep" in OSX 10.11 (El Capitan) - related to sc_xud** issue #17092
- **CHANGE: Asynchronous feedback system re-implemented to allow for the first two ADDED** changelog items
- **CHANGE: Hardware divider used to generate bit-clock from master clock (xCORE-200 only).** Allows easy support for greater number of master-clock to sample-rate ratios.
- CHANGE: module_queue no longer uses any assert module/lib

10.23 6.13.0

- **ADDED: Device now uses implicit feedback when input stream is available (previously explicit** feedback pipe always used). This saves chanend/EP resources and means less processing burden for the host. Previous behaviour available by enabling UAC_FORCE_FEEDBACK_EP
- **RESOLVED: Exception when SPDIF_TX and ADAT_TX both enabled due to clock-block being configured** after already started. Caused by SPDIF_TX define check typo
- **RESOLVED: DFU flag address changed to properly conform to memory address range allocated to** apps by tools
- RESOLVED: Build failure when DFU disabled
- RESOLVED: Build issue when I2S_CHANS_ADC/DAC set to 0 and CODEC_MASTER enabled
- RESOLVED: Typo in MCLK_441 checking for MIN_FREQ define
- CHANGE: Mixer and non-mixer channel comms scheme (decouple <-> audio path) now identical
- **CHANGE: Input stream buffering modified such that during overflow older samples are removed** rather than ignoring most recent samples. Removes any chance of stale input packets being sent to host
- CHANGE: module_queue (in sc_usb_audio) now uses lib_xassert rather than module_xassert

10.24 6.12.6

- RESOLVED: Build error when DFU is disabled
- RESOLVED: Build error when I2S_CHANS_ADC or I2S_CHANS_DAC set to 0 and CODEC_MASTER enabled

10.25 6.12.5

- RESOLVED: Stream issue when NUM_USB_CHAN_IN < I2S_CHANS_ADC

10.26 6.12.4

- RESOLVED: DFU fail when DSD enabled and USB library not running on tile[0]

10.27 6.12.3

- **RESOLVED: Method for storing persistent state over a DFU reboot modified to improve resilience** against code-base and tools changes

10.28 6.12.2

- RESOLVED: Reboot code (used for DFU) failure in tools versions > 14.0.2 (xCORE-200 only)
- RESOLVED: Run-time exception in mixer when MAX_MIX_COUNT > 0 (xCORE-200 only)
- RESOLVED: MAX_MIX_COUNT checked properly for mix strings in string table
- **CHANGE: DFU code re-written to use an XC interface. The flash-part may now be connected** to a separate tile to the tile running USB code
- CHANGE: DFU code can now use quad-SPI flash
- **CHANGE: Example xmos_dfu application now uses a list of PIDs to allow adding PIDs easier.** -listdevices command also added.
- CHANGE: I2S_CHANS_PER_FRAME and I2S_WIRES_xxx defines tidied

10.29 6.12.1

- RESOLVED: Fixes to TDM input timing/sample-alignment when BCLK=MCLK
- RESOLVED: Various minor fixes to allow ADAT_RX to run on xCORE 200 MC AUDIO hardware
- CHANGE: Moved from old SPDIF define to SPDIF_TX

10.30 6.12.0

- ADDED: Checks for XUD_200_SERIES define where required
- **RESOLVED: Run-time exception due to decouple interrupt not entering correct issue mode** (affects XCORE-200 only)
- CHANGE: SPDIF Tx Core may now reside on a different tile from I2S
- CHANGE: I2C ports now in structure to match new module_i2c_singleport/shared API.

- Changes to dependencies:
  - sc_util: 1.0.4rc0 -> 1.0.5alpha0
    - xCORE-200 Compatiblity fixes to module_locks

10.31 6.11.3

- RESOLVED: (Major) Streaming issue when mixer not enabled (introduced in 6.11.2)

10.32 6.11.2

- **RESOLVED: (Major) Enumeration issue when MAX_MIX_COUNT > 0 only. Introduced in mixer** optimisations in 6.11.0. Only affects designs using mixer functionality.
- **RESOLVED: (Normal) Audio buffering request system modified such that the mixer output is**

not silent when in underflow case (i.e. host output stream not active) This issue was introduced with the addition of DSD functionality and only affects designs using mixer functionality.
- **RESOLVED: (Minor) Potential build issue due to duplicate labels in inline asm in** set_interrupt_handler macro
- **RESOLVED: (Minor) BCD_DEVICE define in devicedefines.h now guarded by ifndef (caused issues** with DFU test build configs.
- RESOLVED: (Minor) String descriptor for Clock Selector unit incorrectly reported
- **RESOLVED: (Minor) BCD_DEVICE in devicedefines.h now guarded by #ifndef (Caused issues with** default DFU test build configs.
- CHANGE: HID report descriptor defines added to shared user_hid.h
- CHANGE: Now uses module_adat_rx from sc_adat (local module_usb_audio_adat removed)

10.33 6.11.1

- ADDED: ADAT transmit functionality, including SMUX. See ADAT_TX and ADAT_TX_INDEX.
- RESOLVED: (Normal) Build issue with CODEC_MASTER (xCore is I2S slave) enabled
- RESOLVED: (Minor) Channel ordering issue in when TDM and CODEC_MASTER mode enabled
- **RESOLVED: (Normal) DFU fails when SPDIF_RX enabled due to clock block being shared between SPDIF** core and FlashLib

10.34 6.11.0

- ADDED: Basic TDM I2S functionality added. See I2S_CHANS_PER_FRAME and I2S_MODE_TDM
- **CHANGE: Various optimisations in 'mixer' core to improve performance for higher** channel counts including the use of XC unsafe pointers instead of inline ASM
- CHANGE: Mixer mapping disabled when MAX_MIX_COUNT is 0 since this is wasted processing.
- **CHANGE: Descriptor changes to allow for channel input/output channel count up to 32** (previous limit was 18)

10.35 6.10.0

- **CHANGE: Endpoint management for iAP EA Native Transport now merged into buffer() core.** Previously was separate core (as added in 6.8.0).
- CHANGE: Minor optimisation to I2S port code for inputs from ADC

10.36 6.9.0

- **ADDED: ADAT S-MUX II functionality (i.e. 2 channels at 192kHz) - Previously only S-MUX** supported (4 channels at 96kHz).
- **ADDED: Explicit build warnings if sample rate/depth & channel combination exceeds** available USB bus bandwidth.
- **RESOLVED: (Major) Reinstated ADAT input functionality, including descriptors and clock** generation/control and stream configuration defines/tables.
- **RESOLVED: (Major) S/PDIF/ADAT sample transfer code in audio() (from ClockGen()) moved to** aid timing.
- **CHANGE: Modifying mix map now only affects specified mix, previous was applied to all** mixes. CS_XU_MIXSEL control selector now takes values 0 to MAX_MIX_COUNT + 1 (with 0 affecting all mixes).
- **CHANGE: Channel c_dig_rx is no longer nullable, assists with timing due to removal of** null checks inserted by compiler.
- **CHANGE: ADAT SMUX selection now based on device sample frequency rather than selected** stream format - Endpoint 0 now configures clockgen() on a sample-rate change rather than

stream start.

10.37 6.8.0

- ADDED: Evaluation support for iAP EA Native Transport endpoints
- **RESOLVED: (Minor) Reverted change in 6.5.1 release where sample rate listing in Audio Class** 1.0 descriptors was trimmed (previously 4 rates were always reported). This change appears to highlight a Windows (only) enumeration issue with the Input & Output configs
- **RESOLVED: (Major) Mixer functionality re-instated, including descriptors and various required** updates compatibility with 13 tools
- RESOLVED: (Major) Endpoint 0 was requesting an out of bounds channel whilst requesting level data
- **RESOLVED: (Major) Fast mix code not operates correctly in 13 tools, assembler inserting long jmp** instructions
- RESOLVED: (Minor) LED level meter code now compatible with 13 tools (shared mem access)
- **RESOLVED (Minor) Ordering of level data from the device now matches channel ordering into** mixer (previously the device input data and the stream from host were swapped)
- CHANGE: Level meter buffer naming now resemble functionality

10.38 Legacy release history

Please see changelog in sw_usb_audio for changes prior to 6.8.0 release.