AN00247: Using lib_xua with lib_spdif (transmit)

Publication Date: 2025/10/28
Document Number: XM-013121-AN v1.0.0

IN THIS DOCUMENT

1 Introduction . . . 1
2 Example application . . . 1

1 Introduction

The XMOS USB Audio (XUA) library provides an implementation of USB Audio Class versions 1.0 and 2.0.

This application note demonstrates the implementation of a basic USB Audio Device with S/PDIF transmit functionality on the XCORE.AI Multichannel (MC) Audio board.

To reduce complexity this application note does not enable any other audio interfaces other than S/PDIF transmit (i.e. no I2S).

2 Example application

2.1 The CMakeLists.txt file

To start using the XMOS XUA library, **lib_xua** needs to be added to the dependent module list in the CMakeLists.txt file. **lib_spdif** is a dependency of **lib_xua** so does not need to be explicitly added. The **lib_board_support** software repository is also used to provide common code to configure the XCORE.AI Multichannel (MC) Audio board for use. This should also be added to the list of dependent modules in the CMakeLists.txt file:

```
set(APP_DEPENDENT_MODULES "lib_xua"
                          "lib_board_support")
```

The **lib_board_support** requires a compiler flag to select the hardware type:

```
set(APP_COMPILER_FLAGS ... -DBOARD_SUPPORT_BOARD=XK_AUDIO_316_MC_AB ...)
```

2.2 Includes

This application requires the system header that defines XMOS XCORE specific defines for declaring and initialising hardware:

```
#include <xs1.h>
#include <platform.h>
```

The XUA library functions are defined in **xua.h**. This header must be included in your code to use the library. Headers are also required for **lib_xud**, **lib_spdif** and the board setup code for the XCORE.AI Multichannel Audio board.

```
#include "xua.h"
#include "xud_device.h"

/* From lib_spdif */
#include "spdif.h"
#include "xk_audio_316_mc_ab/board.h"
```

2.3 Declarations

Allocating hardware resources for lib_xua

A minimal implementation of a USB Audio device, without I2S functionality, using `lib_xua` requires the follow pins:

- Audio Master clock (from clock source to XCORE)

On an XCORE the pins are controlled by `ports`. The application therefore declares a port for the master clock input signal.

```
/* Lib_xua port declarations. Note, the defines come from the xn file */
in port p_mclk_in                  = PORT_MCLK_IN;      /* Master clock for the audio IO tile */
```

`lib_xua` also requires two ports for internally calculating USB feedback. Please refer to the `lib_xua` library documentation for further details. The additional input port for the master clock is required since USB and S/PDIF do not reside of the same tiles on the example hardware.

These ports are declared as follows:

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

In addition to `port` resources two clock-block resources are also required:

```
/* Clock-block declarations */
clock clk_audio_mclk               = on tile[1]: XS1_CLKBLK_5;   /* Master clock */
clock clk_audio_mclk_usb           = on tile[0]: XS1_CLKBLK_1;   /* Master clock for USB tile */
```

Again, for the same reasoning as the master-clock ports, two master-clock clock-blocks are required - one on each tile.

Allocating hardware resources for lib_spdif

The S/PDIF transmitter requires a single (buffered) 1-bit port:

```
/* Lib_spdif port declarations. Note, the defines come from the xn file */
buffered out port:32 p_spdif_tx    = PORT_SPDIF_OUT;            /* SPDIF transmit port */
```

This port must be clocked from the audio master clock. This application note chooses to declare an extra clock-block as follows:

```
clock clk_spdif_tx                 = on tile[1]: XS1_CLKBLK_4;   /* Clock block for S/PDIF transmit */
```

Other declarations

`lib_xua` requires the manual declaration of tables for the endpoint types for `lib_xud` and the calling the main XUD function in a par (`XUD_Main()`).

For a simple application the following endpoints are required:

- `Control` endpoint zero
- `Isochonous` endpoint for each direction for audio data to/from the USB host

These are declared as follows:

```
/* Endpoint type tables - informs XUD what the transfer types for each Endpoint in use and also
 * if the endpoint wishes to be informed of USB bus resets */
XUD_EpType epTypeTableOut[]   = {XUD_EPTYPE_CTL | XUD_STATUS_ENABLE, XUD_EPTYPE_ISO};
XUD_EpType epTypeTableIn[]    = {XUD_EPTYPE_CTL | XUD_STATUS_ENABLE, XUD_EPTYPE_ISO};
```

2.4 Hardware specific setup

Some code is needed to perform the hardware-specific setup for the board being used in this application note.

The `xk_audio_316_mc_ab_config_t` structure is used to specify hardware-specific configuration options, such as clocking modes and frequencies.

The `i_i2c_client` unsafe client interface is required to have a globally-scoped variable for gaining access to the `i2c_master_if` interface from the audio hardware functions.

```
/* Board configuration from lib_board_support */
static const xk_audio_316_mc_ab_config_t hw_config = {
        CLK_FIXED,              // clk_mode. Drive a fixed MCLK output
        0,                      // 1 = dac_is_clock_master
        MCLK_48,
        0,                      // pll_sync_freq (unused when driving fixed clock)
        AUD_316_PCM_FORMAT_I2S,
        32,                     // data bits
        2                       // channels per frame
};

unsafe client interface i2c_master_if i_i2c_client;
```

The following functions are called by `XUA_AudioHub` to configure the hardware; they are defined as wrapper functions around the board-specific code from `lib_board_support`.

```
void AudioHwInit()
{
    unsafe {
        xk_audio_316_mc_ab_AudioHwInit((client interface i2c_master_if)i_i2c_client, hw_config);
        }
}

void AudioHwConfig(unsigned samFreq, unsigned mClk, unsigned dsdMode, unsigned sampRes_DAC, unsigned sampRes_ADC)
{
    unsafe {
        xk_audio_316_mc_ab_AudioHwConfig((client interface i2c_master_if)i_i2c_client, hw_config, samFreq, mClk,
↪dsdMode, sampRes_DAC, sampRes_ADC);
    }
}
```

2.5 Configuring lib_xua

`lib_xua` must be configured to enable S/PDIF Tx functionality.

`lib_xua` has many parameters than can be configured at build time, some examples include:

- Sample-rates
- Channel counts
- Audio Class version
- Product/Vendor IDs
- Various product strings
- Master clock frequency

To enable S/PDIF functionality `XUA_SPDIF_TX_EN` must be set to a non-zero value. Setting this will cause the `XUA_AudioHub` tasks to forward samples and sample rate information to the S/PDIF transmitter task.

These parameters are set via defines in an optional `xua_conf.h` header file. For this simple application the complete contents of this file are as follows:

```
// Copyright 2017-2025 XMOS LIMITED.
// This Software is subject to the terms of the XMOS Public Licence: Version 1.

#ifndef _XUA_CONF_H_
#define _XUA_CONF_H_

#define NUM_USB_CHAN_OUT   (2)
#define NUM_USB_CHAN_IN    (0)
#define I2S_CHANS_DAC      (0)
#define I2S_CHANS_ADC      (0)
#define MCLK_441           (512 * 44100)
#define MCLK_48            (512 * 48000)
#define MIN_FREQ           (48000)
#define MAX_FREQ           (48000)

#define EXCLUDE_USB_AUDIO_MAIN

#define XUA_SPDIF_TX_EN    (1)
#define SPDIF_TX_INDEX     (0)
#define VENDOR_STR         "XMOS"
#define VENDOR_ID          0x20B1
#define PRODUCT_STR_A2     "XUA SPDIF Example"
#define PRODUCT_STR_A1     "XUA SPDIF Example"
#define PID_AUDIO_1        (1)
#define PID_AUDIO_2        (2)
#define AUDIO_CLASS        (2)
#define AUDIO_CLASS_FALLBACK (0)
#define BCD_DEVICE         (0x1234)
#define XUA_DFU_EN         (0)

#endif
```

2.6 The application main() function

The `main()` function sets up the tasks in the application.

Various channels/interfaces are required in order to allow the required tasks to communicate. These must first be declared:

```
    /* Channels 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;

    /* Channel for communication between AudioHub and S/PDIF transmitter */
    chan c_spdif_tx;

    /* Interface for access to I2C for setting up hardware */
    interface i2c_master_if i2c[1];
```

The rest of the `main()` function starts all of the tasks in parallel using the xC `par` construct:

```
    par
    {
        /* Low level USB device layer core */
        on tile[0]: XUD_Main(c_ep_out, 2, c_ep_in, 2, c_sof, epTypeTableOut, epTypeTableIn, XUD_SPEED_HS, XUD_
↪PWR_SELF);

        /* Endpoint 0 core from lib_xua */
        /* Note, since we are not using many features we pass in null for quite a few params.. */
        on tile[0]: XUA_Endpoint0(c_ep_out[0], c_ep_in[0], c_aud_ctl, 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 */
        on tile[0]: {

                        /* 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);
                    }
```

(continues on next page)

(continued from previous page)

```
        /* AudioHub() (I2S) and S/SPDIF Tx are on the same tile */
        on tile[1]: {

                        /* Setup S/PDIF tx port from clock etc - note we do this before par to avoid parallel
↪usage */
                        spdif_tx_port_config(p_spdif_tx, clk_spdif_tx, p_mclk_in, 7);

                        start_clock(clk_spdif_tx);

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

                            {
                                unsafe {
                                    i_i2c_client = i2c[0];
                                }
                                /* 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);
                            }
                        }
                    }

        on tile[0]: {
                        xk_audio_316_mc_ab_board_setup(hw_config);
                        xk_audio_316_mc_ab_i2c_master(i2c);
                    }
    }
```

This code starts the low-level USB task, an Endpoint 0 task, an Audio buffering task and a task to handle the audio I/O. Note, since there is no I2S functionality in this example this task simply forwards samples to the SPDIF transmitter task. In addition the `spdif_tx()` task is also run.

Note that the `spdif_tx_port_config()` function is called before a nested `par` of `spdif_tx()` and `XUA_AudioHub()`. This is because of the "shared" nature of `p_mclk_in` and avoids a parallel usage check failure by the XMOS tool-chain.

It also runs `xk_audio_316_mc_ab_board_setup()` and `xk_audio_316_mc_ab_i2c_master()` from `lib_board_support` that are used for setting up the hardware.

2.7 Building the example

This section assumes that the XMOS XTC Tools have been downloaded and installed. The required version is specified in the accompanying `README`.

Installation instructions can be found here.

Special attention should be paid to the section on Installation of Required Third-Party Tools.

The application is built using the xcommon-cmake build system, which is provided with the XTC tools and is based on CMake.

The `an00247` software ZIP package should be downloaded and extracted to a chosen working directory.

To configure the build, the following commands should be run from an XTC command prompt:

```
cd an00247
cd app_an00247
cmake -G "Unix Makefiles" -B build
```

All required dependencies are included in the software package. If any dependencies are missing, they will be retrieved automatically during this step.

The application binaries should then be built using **xmake**:

```
xmake -j -C build
```

Binary artifacts (.xe files) will be generated under the appropriate subdirectories of the **app_an00247/bin** directory — one for each supported build configuration.

For subsequent builds, the **cmake** step may be omitted. If **CMakeLists.txt** or other build files are modified, **cmake** will be re-run automatically by **xmake** as needed.

2.8 Hardware setup

To run the application, use a USB cable to connect the on-board xTAG debug adapter (marked DEBUG) to your development computer. Use another USB cable to connect the USB receptacle marked USB DEVICE to the device you wish to play audio from.

A device capable of receiving an S/PDIF signal (i.e. a speaker) should be connected to COAX TX.

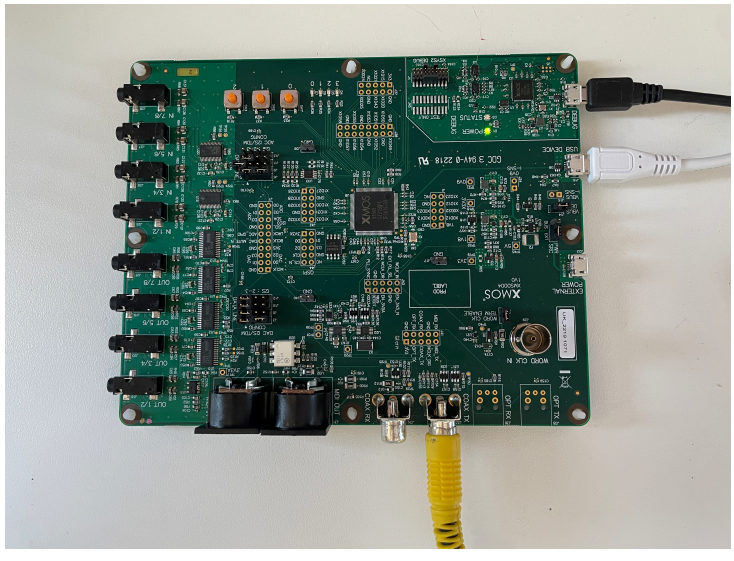

Fig. 1: Hardware setup

2.9 Running the example

From an XTC command prompt, the following command should be run from the **an00247/app_an00247** directory:

```
xrun ./bin/app_an00247.xe
```

Alternatively, the application can be programmed into flash memory for standalone execution:

```
xflash ./bin/app_an00247.xe
```

Once running, a USB audio device called **XUA SPDIF Example** should enumerate on the host machine - note, Windows operating systems may require a third party driver for correct operation. The USB OUT stream from the host to device is routed to SPDIF TX.