AN00162: Implementing an I2S loopback using the lib_i2s library

Publication Date: 2026/4/15
Document Number: XM-010239-AN v2.1.1

IN THIS DOCUMENT

1 Introduction

The XMOS I²S library provides software defined, industry-standard, I²S (Integrated Inter-chip Sound) components that allows you to stream audio between devices using xCORE GPIO ports.

I²S is a specific type of PCM digital audio communication using a serial clock (sometimes refered as bit clock) line, word clock line and at least one multiplexed data line.

The library includes features such as I²S master (newly termed controller), I²S slave (newly termed target), and TDM master components. This application note uses the library to create an I²S master digital loopback.

2 Block diagram

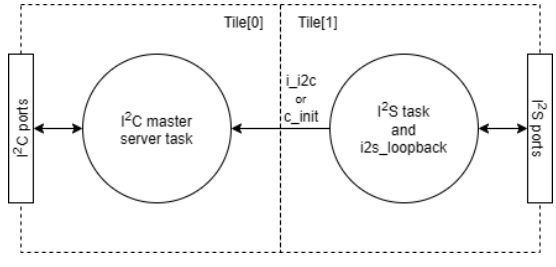

Fig. 1: Application block diagram

The main application fits within one thread with a remote I²C task to configure the audio hardware remotely from the other tile. The `lib_board_support` library, which includes I²C, takes care of the audio hardware setup.

The I²S task calls back to the `i2s_loopback` task, and the processing in the `i2s_loopback` task occurs in-between the I/O operations of I²S.

3 I²S loopback demo

3.1 The CMakeLists.txt file

XMOS applications use the xcommon-cmake build and dependency management system. This is bundled with the *XMOS* XTC tools.

To start using the I²S, include **lib_i2s** as a dependent module in the application's CMakeLists.txt file:

Listing 1: Identifying I²S as a dependent module

```
set(APP_DEPENDENT_MODULES   "lib_i2s")
```

This demo also uses the I²C library (**lib_i2c**) which **lib_board_support** includes as a dependent module. The application uses I²C to configure the audio CODECs. Consequently, the application's CMakeLists.txt includes both **lib_i2s** and **lib_board_support** as dependent modules.

Listing 2: **APP_DEPENDENT_MODULES** definition

```
set(APP_DEPENDENT_MODULES  "lib_board_support(1.3.0)"
                           "lib_i2s(6.0.1)")
```

3.2 Includes

All xC files which declare the application **main()** function need to include **platform.h**. XMOS xcore specific defines for declaring and initialising hardware appear in **xs1.h**.

Listing 3: Critical headers to include

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

The **i2s.h** file defines the I²S library functions. This header must be included to use the library.

Listing 4: The I²S header

```
#include "i2s.h"
```

Another include gives access to the the board setup code. It varies depending on the board in use.

Listing 5: Setup header for the XK-AUDIO-316-MC board

```
#include "xk_audio_316_mc_ab/board.h"
```

Listing 6: Setup header for the XK-EVK-XU316 board

```
#include "xk_evk_xu316/board.h"
```

3.3 Allocating hardware resources

An I²S interface requires both clock and data pins in order to communicate with the audio CODEC device. On an xcore the pins are controlled by **ports**.

The ports used by the I²S library are declared on the tile on which they reside. Their declaration includes each port's direction and buffered nature. Three 1-bit ports are used for the I²S clock signals.

Listing 7: Defining I^{2}S clock ports

```
on tile[1]: in port p_mclk =                            PORT_MCLK_IN;
on tile[1]: buffered out port:32 p_lrclk =              PORT_I2S_LRCLK;
on tile[1]: out port p_bclk =                           PORT_I2S_BCLK;
```

This loopback application uses 1-bit ports for input and output. The number of input and output ports, i.e., the value of **NUM_I2S_LINES** depends on the board in use.

Listing 8: Port definition for the XK-AUDIO-316-MC board

```
on tile[1]: buffered out port:32 p_dac[NUM_I2S_LINES] =     {PORT_I2S_DAC0, PORT_I2S_DAC1, PORT_I2S_DAC2, PORT_
↪I2S_DAC3};
on tile[1]: buffered in port:32 p_adc[NUM_I2S_LINES] =      {PORT_I2S_ADC0 ,PORT_I2S_ADC1, PORT_I2S_ADC2, PORT_
↪I2S_ADC3};
```

Listing 9: Port definition for the XK-EVK-XU316 board

```
on tile[1]: buffered out port:32 p_dac[NUM_I2S_LINES] =     {PORT_I2S_DAC_DATA};
on tile[1]: buffered in port:32 p_adc[NUM_I2S_LINES] =      {PORT_I2S_ADC_DATA};
```

The xcore also provides **clock block** hardware to efficiently generate a clock signal that can either be driven out of a port or used to control a port. This application uses one clock block.

Listing 10: Clock block definition

```
on tile[1]: clock bclk =                                XS1_CLKBLK_1;
```


3.4 The application `main()` function

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

Firstly, it declares `interfaces` and `channels`. XC interface and channel provides a means for concurrent tasks to communicate with each other. This application includes an interface for the I^{2}S master and an interface/channel for the I^{2}C master.

Listing 11: I^{2}S Interface definition

```
interface i2s_frame_callback_if i_i2s;
```

An interface or channel is used in different board for the I^{2}C master.

When running on an `XK-AUDIO-316-MC` board, it includes an interface for the I^{2}C master also.

Listing 12: I^{2}C Interface definition for the XK-AUDIO-316-MC board

```
interface i2c_master_if i_i2c[1]; // Cross tile interface
```

On the `XK-EVK-XU316` board, the application uses a channel instead.

Listing 13: I^{2}C Channel definition for the XK-EVK-XU316 board

```
chan c_init; // Cross tile channel
```

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

Listing 14: Tasks for the XK-AUDIO-316-MC board

```
par {
    on tile[0]: {
        xk_audio_316_mc_ab_board_setup(hw_config); // Setup must be done on tile[0]
        xk_audio_316_mc_ab_i2c_master(i_i2c);      // Run I2C master server task to allow control from tile[1]
    }

    on tile[1]: {
        interface i2s_frame_callback_if i_i2s;

        par {
            // The application - loopback the I2S samples - note callbacks are inlined so does not take a thread
            [[distribute]] i2s_loopback(i_i2s, i_i2c[0]);
            i2s_frame_master(i_i2s, p_dac, NUM_I2S_LINES, p_adc, NUM_I2S_LINES, DATA_BITS, p_bclk, p_lrclk, p_
↪mclk, bclk);
        }
    }
}
```

Listing 15: Tasks for the XK-EVK-XU316 board

```
par {
    on tile[0]: {
        xk_evk_xu316_AudioHwRemote(c_init);  // Run I2C master task to allow control from tile[1]
    }

    on tile[1]: {
        interface i2s_frame_callback_if i_i2s;
        xk_evk_xu316_AudioHwChanInit(c_init);

        par {
            // The application - loopback the I2S samples - note callbacks are inlined so does not take a thread
            [[distribute]] i2s_loopback(i_i2s);
            i2s_frame_master(i_i2s, p_dac, NUM_I2S_LINES, p_adc, NUM_I2S_LINES, DATA_BITS, p_bclk, p_lrclk, p_
↪mclk, bclk);
        }
    }
}
```

This code starts the I^{2}S master, the I^{2}C master, the board setup logic, and the loopback application.

The call to the `i2s_loopback` task in the `par` is marked with the `[[distribute]]` attribute, and the corresponding `i2s_loopback()` function is marked with the

`[[distributable]]` attribute. These attributes mean that the `i2s_loopback` task will run on an existing logical core if possible rather than creating a new one. In this case it will share the logical core used by the I^2S master.

3.5 Configuring audio CODECs

All of the audio hardware is setup using functions in `lib_board_support`. The previous inclusion of `board.h` from the `xk_audio_316_mc_ab` directory targets the hardware setup to the XU316 Multichannel Audio board(`XK-AUDIO-316-MC`) and the `xk_evk_xu316` directory targets the XCORE.AI Evaluation Kit(`XK-EVK-XU316`). These lines perform some board-specific initialisation and start the I^2C task.

Listing 16: Board initialisation for the XK-AUDIO-316-MC board

```
xk_audio_316_mc_ab_board_setup(hw_config); // Setup must be done on tile[0]
```

> **Note**
>
> The XK-EVK-XU316 board does not need board-specific initialisation.

The `hw_config` struct specifies the hardware configuration. In this case, it sets up the xcore to be an I^2S master with the following settings:

Listing 17: I2S master settings for the XK-AUDIO-316-MC board

```
#define SAMPLE_FREQUENCY        48000
#define MASTER_CLOCK_FREQUENCY  24576000
#define DATA_BITS               32
#define CHANS_PER_FRAME         2
#define NUM_I2S_LINES           4
```

Listing 18: I2S master settings for the XK-EVK-XU316 board

```
#define SAMPLE_FREQUENCY        48000
#define MASTER_CLOCK_FREQUENCY  24576000
#define DATA_BITS               32
#define CHANS_PER_FRAME         2
#define NUM_I2S_LINES           1
```

> **Note**
>
> `lib_board_support` only support changing master clock frequency in XK-EVK-XU316 Board

The following functions, called from the `i2s_loopback` task, complete the initialisation and configuration of the ADCs and DACs:

Listing 19: ADCs and DACs initialisation and configuration for the XK-AUDIO-316-MC board

```
xk_audio_316_mc_ab_AudioHwInit(i_i2c, hw_config);
xk_audio_316_mc_ab_AudioHwConfig(i_i2c, hw_config, SAMPLE_FREQUENCY, MASTER_CLOCK_FREQUENCY, 0, DATA_BITS, DATA_
↪BITS);
```

Listing 20: ADCs and DACs initialisation and configuration for the XK-EVK-XU316 board

```
xk_evk_xu316_AudioHwInit(hw_config);
xk_evk_xu316_AudioHwConfig(SAMPLE_FREQUENCY, MASTER_CLOCK_FREQUENCY, 0, DATA_BITS, DATA_BITS);
```

For full documentation of the `lib_board_support` API, please refer to the following link: lib_board_support.

3.6 The `i2s_loopback` application

The I²S loopback task provides the function of a digital loopback so that all I²S samples received by the device will be forwarded on.

The task itself is declared as a **`[[distributable]]`** function allowing it to share a logical core with other tasks. This xC feature can be enabled for any task with the form:

Listing 21: Distributable task pattern

```
// Task initialization code here
while(1) {
  select {
    // Event cases here
  }
}
```

The function takes a number of arguments:

Listing 22: i2s_loopback function for the XK-AUDIO-316-MC board

```
[[distributable]]
void i2s_loopback(server i2s_frame_callback_if i_i2s, client i2c_master_if i_i2c)
```

Listing 23: i2s_loopback function for the XK-EVK-XU316 board

```
[[distributable]]
void i2s_loopback(server i2s_frame_callback_if i_i2s)
```

The interface to the I²S master, **`server i2s_frame_callback_if i_i2s`**, provides a set of callback functions. The I²S master will call these functions as needed.

The **`i2s_loopback`** task uses the I²C master interface, **`client i2c_master_if i_i2c`**, to configure the CODECs (ADCs and DACs) remotely.

The main loop in the **`i2s_loopback`** task handles the I²S interface calls.

Listing 24: I²S task main loop

```
while (1) {
select {
   case i_i2s.init(i2s_config_t &?i2s_config, tdm_config_t &?tdm_config):
      i2s_config.mode = I2S_MODE_I2S;
      i2s_config.mclk_bclk_ratio = (MASTER_CLOCK_FREQUENCY / (SAMPLE_FREQUENCY * CHANS_PER_FRAME * DATA_BITS));
      ... // Audio HW Config for the board
      break;

   case i_i2s.receive(size_t n_chans, int32_t in_samps[n_chans]):
      for (int i = 0; i < n_chans; i++){
            samples[i] = in_samps[i]; // copy samples
      }
      break;

   case i_i2s.send(size_t n_chans, int32_t out_samps[n_chans]):
      for (int i = 0; i < n_chans; i++){
            out_samps[i] = samples[i]; // copy samples
      }
      break;

   case i_i2s.restart_check() -> i2s_restart_t restart:
      restart = I2S_NO_RESTART; // Keep on looping
      break;
   } // End select
} // End while (1)
```

The I²S master library calls the **`init()`** method before it starts any data streaming. This call allows the application to reset and configure the audio CODECs, for example when the sample rate changes.

The **`receive()`** interface method is called when the master has received a frame of audio samples (all channels in one sample period). The **`receive()`** method stores the samples in the **`samples`** array.

The I^{2}S master calls the `send()` interface method when it needs a new frame of samples to send. In this case the application simply returns the frame of samples previously received.

Finally, the `restart_check()` method is called by the I^{2}S master once per frame. It allows the application to control restart or shutdown of the I^{2}S master. In this case the application continues to run forever and so always returns `I2S_NO_RESTART`.

4 Building the application

The following section assumes you have downloaded and installed the XMOS XTC tools. See the *README* file for required version. Installation instructions can be found here. Be sure to pay attention to the section Installation of required third-party tools.

The application uses the xcommon-cmake build system as bundled with the XTC tools.

The `AN00162_i2s_loopback_demo` software zip-file should be downloaded and unzipped to a chosen directory.

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

```
cd an00162
cd app_an00162
cmake -G "Unix Makefiles" -B build
```

All required dependencies are included in the software download. However, if any are missing, they will be downloaded by the build system.

Finally, the application binaries can be built using `xmake`:

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

The application uses approximately 3 kiB on Tile[0] and 7 kiB on Tile[1] (each tile has 512 kiB available).

5 Demo hardware setup

Please refer to the XU316 Multichannel Audio board and XCORE.AI Evaluation Kit hardware platform documentation.

The demo is designed to run on the XU316 Multichannel Audio board and XCORE.AI Evaluation Kit. To run the demo:

- Connect a USB cable from your host to the DEBUG connector.
- Connect a USB cable from your host to the USB DEVICE connector.
- Connect a sound source to the 3.5mm line in. Channels 1-2, 3-4, 5-6 or 7-8 can be used.
- Connect headphones or speakers to the corresponding line out.

6 Running the demo application

To run the application return to the `app_an00162` directory and run the following command:

Listing 25: For the XK-AUDIO-316-MC board

```
xrun app_an00162_xk_audio_316_mc/bin/app_an00162_xk_audio_316_mc.xe
```

Listing 26: For the XK-EVK-XU316 board

```
xrun app_an00162_xk_evk_xu316/bin/app_an00162_xk_evk_xu316.xe
```

You should hear the audio connected to the analog input jacks looped back to the output jacks.

7 References

- XMOS Tools User Guide
  https://www.xmos.com/documentation/XM-014363-PC-9/html/

- XMOS xcore Programming Guide
  https://www.xmos.com/published/xmos-programming-guide
- XMOS Libraries
  https://www.xmos.com/libraries/
- I^2S Protocol
  https://en.wikipedia.org/wiki/I2S