XMOS

an00162: Implementing an I2S loopback using the lib_i2s library

Publication Date: 2025/3/19
Document Number: XM-010239-AN v2.0.1

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

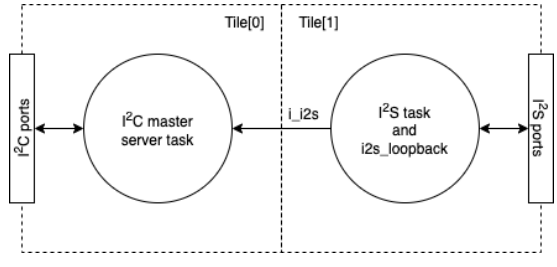

Fig. 1: Application block diagram

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:

```
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.

```
set(APP_DEPENDENT_MODULES  "lib_board_support(1.1.1)"
                           "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**.

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

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

```
#include "i2s.h"
#include "xk_audio_316_mc_ab/board.h"
```

The other include gives access to the the board setup code.

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. This loopback application uses four 1-bit ports for input and four more for output.

```
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;
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};
```

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.

```
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`. An xC interface 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 for the I^{2}C master.

```
            interface i2s_frame_callback_if i_i2s;
```

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

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

```
    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);
            }
        }
    }
```

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^{2}S 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`). These lines perform some board-specific initialisation and start the I^{2}C task.

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

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

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

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

```
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);
```

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 I^{2}S loopback task provides the function of a digital loopback so that all I^{2}S 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:

```
...
while(1) {
  select {
    ...
  }
}
```

The function takes a number of arguments:

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

The interface to the I^{2}S master, **`server i2s_frame_callback_if i_i2s`**, provides a set of callback functions. The I^{2}S master will call these functions as needed.

The **`i2s_loopback`** task uses the I^{2}C 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 I^{2}S interface calls.

```
    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));
            xk_audio_316_mc_ab_AudioHwConfig(i_i2c, hw_config, SAMPLE_FREQUENCY, MASTER_CLOCK_FREQUENCY, 0, DATA_
↪BITS, DATA_BITS);
            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 I^{2}S 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 hardware platform documentation.

The demo is designed to run on the XU316 Multichannel Audio board. 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:

```
xrun bin/app_an00162.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/I%C2%B2S

8 Full source code listing

8.1 Source code for main.xc

```
// Copyright 2014-2024 XMOS LIMITED.
// This Software is subject to the terms of the XMOS Public Licence: Version 1.

#include <platform.h>
#include <xs1.h>
#include "i2s.h"
#include "xk_audio_316_mc_ab/board.h"

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

// I2S resources
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;
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};
on tile[1]: clock bclk =                                        XS1_CLKBLK_1;


// 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
        MASTER_CLOCK_FREQUENCY,
        0,                      // pll_sync_freq (unused when driving fixed clock)
        AUD_316_PCM_FORMAT_I2S,
        DATA_BITS,
        CHANS_PER_FRAME
};

[[distributable]]
void i2s_loopback(server i2s_frame_callback_if i_i2s, client i2c_master_if i_i2c)
{
    int32_t samples[NUM_I2S_LINES * CHANS_PER_FRAME] = {0}; // Array used for looping back samples
    // Config can be done remotely via i_i2c
    xk_audio_316_mc_ab_AudioHwInit(i_i2c, hw_config);

    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));
            xk_audio_316_mc_ab_AudioHwConfig(i_i2c, hw_config, SAMPLE_FREQUENCY, MASTER_CLOCK_FREQUENCY, 0, DATA_
↪BITS, DATA_BITS);
            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)
} // End i2s_loopback


int main(void)
{
    interface i2c_master_if i_i2c[1]; // Cross tile interface

    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;
```

(continues on next page)

(continued from previous page)

```
            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);
            }
        }
    }
    return 0;
}
```