Application Note: AN00162

# Using the I2S library

I$^2$S interfaces are key to many audio systems. XMOS technology is perfectly suited to these applications - supporting a wide variety of standard interfaces and also a large range of DSP functions.

This application note demonstrates the use of the XMOS I$^2$S library to create a digital audio loopback on an XMOS multicore microcontroller.

The code used in the application note configures the audio codecs to simultaneously send and receive audio samples. It then uses the I$^2$S library to loopback all 8 channels.

---

### Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0
- XMOS I$^2$S/TDM library - Version 2.1.0
- XMOS GPIO library - Version 1.0.0
- XMOS I$^2$C library - Version 3.1.0

### Required hardware

The example code provided with the application has been implemented and tested on the xCORE-200 Multichannel Audio Platform.

### Prerequisites

- This document assumes familiarity with I$^2$S interfaces, the XMOS xCORE architecture, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the references appendix.
- For a description of XMOS related terms found in this document please see the XMOS Glossary[1].

[1] http://www.xmos.com/published/glossary

Copyright 2016 XMOS Ltd.

# 1 Overview

## 1.1 Introduction

The XMOS I$^2$S library provides software defined, industry-standard, I$^2$S (Integrated Interchip Sound) components that allows you to stream audio between devices using xCORE GPIO ports.

I$^2$S is a specific type of PCM digital audio communication using a bit clock line, word clock line and at least one multiplexed data line.

The library includes features such as I$^2$S master, I$^2$S slave, and TDM master components. This application note uses the library to create an I$^2$S master digital loopback.

## 1.2 Block Diagram

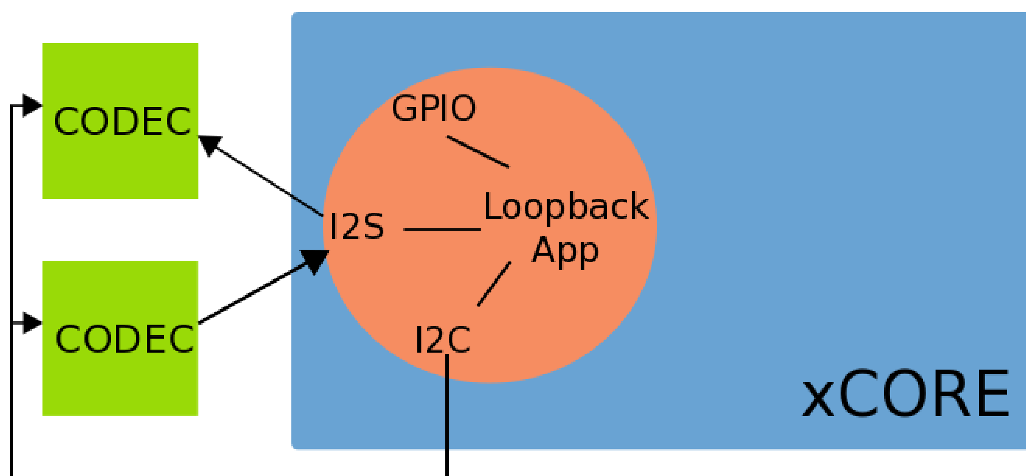


Figure 1: Application block diagram

The application fits entirely within one logical core. The I$^2$S, GPIO, and I$^2$C drivers are all time-multiplexed with the loopback application.

Copyright 2016 XMOS Ltd.

# 2 I$^2$S loopback demo

## 2.1 The Makefile

To start using the I$^2$S, you need to add `lib_i2s` to you Makefile:

```
USED_MODULES = .. lib_i2s ...
```

This demo also uses the I$^2$C library (`lib_i2c`) and the GPIO library (`lib_gpio`). The I$^2$C library is used to configure the audio codecs. The GPIO library abstracts the use of a 4-bit port to drive four individual reset and select lines. So the Makefile also includes:

```
USED_MODULES = .. lib_gpio lib_i2c ..
```

## 2.2 Includes

All xC files which declare the application `main()` function need to include `platform.h`. XMOS xCORE specific defines for declaring and initialising hardware are defined in `xs1.h`.

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

The I$^2$S library functions are defined in `i2s.h`. This header must be included in your code to use the library.

```
#include "i2s.h"
#include "i2c.h"
#include "gpio.h"
```

The other two includes give access to the I$^2$C and GPIO library functions which this application note relies on.

## 2.3 Allocating hardware resources

An I$^2$S 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 I$^2$S library are declared on the tile they reside and with their direction and buffered nature. This loopback application uses four 1-bit ports for input and four more for output:

```
on tile[0]: out buffered port:32 p_lrclk = XS1_PORT_1G;
on tile[0]: out buffered port:32 p_bclk = XS1_PORT_1H;
on tile[0]: in port p_mclk = XS1_PORT_1F;
on tile[0]: out buffered port:32 p_dout[4] = {XS1_PORT_1M, XS1_PORT_1N, XS1_PORT_1O, XS1_PORT_1P};
on tile[0]: in buffered port:32 p_din[4] = {XS1_PORT_1I, XS1_PORT_1J, XS1_PORT_1K, XS1_PORT_1L};
```

The xCORE also provides `clock block` hardware to efficiently generate clock singal that can either be driven out on a port or used to control a port. In this application two clocks blocks are used:

```
on tile[0]: clock mclk = XS1_CLKBLK_1;
on tile[0]: clock bclk = XS1_CLKBLK_2;
```

The other ports declared are used by the I$^2$C library to configure the audio codecs and by the application to control reset and chip select:

```
on tile[0]: port p_i2c = XS1_PORT_4A;
on tile[0]: port p_gpio = XS1_PORT_8C;
```

### 2.4 The application main() function

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

Firstly, the `interfaces` are declared. In xC interfaces provide a means of concurrent tasks communicating with each other. In this application there is an interface for the $I^2S$ master, an interface for the $I^2C$ master and one for the GPIO output.

```
interface i2s_callback_if i_i2s;
interface i2c_master_if i_i2c[1];
interface output_gpio_if i_gpio[4];
```

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

```
par {
  on tile[0]: {
    /* System setup, I2S + Codec control over I2C */
    configure_clock_src(mclk, p_mclk);
    start_clock(mclk);
    i2s_master(i_i2s, p_dout, 4, p_din, 4, p_bclk, p_lrclk, bclk, mclk);
  }

  on tile[0]: [[distribute]] i2c_master_single_port(i_i2c, 1, p_i2c, 100, 0, 1, 0);
  on tile[0]: [[distribute]] output_gpio(i_gpio, 4, p_gpio, gpio_pin_map);

  /* The application - loopback the I2S samples */
  on tile[0]: [[distribute]] i2s_loopback(i_i2s, i_i2c[0], i_gpio[0], i_gpio[1], i_gpio[2], i_gpio[3]);
}
```

This code starts the $I^2S$ master, the $I^2C$ master, the GPIO control and the loopback application.

Before the $I^2S$ master runs, the system configuration is run and the master clock is connected from the input port to the clock block and then started. The $I^2S$ master task then starts and consumes a logical core on the xCORE device.

The remaining tasks in the `par` are marked with the `[[distribute]]` attribute. This means they will run on an existing logical core if possible. In this case they will all share the one logical core with the $I^2S$ master so this entire application requires only one core.

## 2.5 Configuring audio codecs

Two audio codecs are used on the xCORE-200 Multichannel Audio Platform. A Cirrus CS5368 for audio input and a Cirrus CS4384 for audio output. These codecs have to be configured before they can be used. The `reset_codecs()` function writes to the appropriate codec registers to configure them:

```
void reset_codecs(client i2c_master_if i2c)
{
  /* Mode Control 1 (Address: 0x02) */
  /* bit[7] : Control Port Enable (CPEN)     : Set to 1 for enable
   * bit[6] : Freeze controls (FREEZE)       : Set to 1 for freeze
   * bit[5] : PCM/DSD Selection (DSD/PCM)    : Set to 0 for PCM
   * bit[4:1] : DAC Pair Disable (DACx_DIS)  : All Dac Pairs enabled
   * bit[0] : Power Down (PDN)               : Powered down
   */
  i2c.write_reg(CS4384_ADDR, CS4384_MODE_CTRL, 0b11000001);

  /* PCM Control (Address: 0x03) */
  /* bit[7:4] : Digital Interface Format (DIF) : 0b1100 for TDM
   * bit[3:2] : Reserved
   * bit[1:0] : Functional Mode (FM) : 0x11 for auto-speed detect (32 to 200kHz)
   */
  i2c.write_reg(CS4384_ADDR, CS4384_PCM_CTRL, 0b00010111);

  /* Mode Control 1 (Address: 0x02) */
  /* bit[7] : Control Port Enable (CPEN)     : Set to 1 for enable
   * bit[6] : Freeze controls (FREEZE)       : Set to 0 for freeze
   * bit[5] : PCM/DSD Selection (DSD/PCM)    : Set to 0 for PCM
   * bit[4:1] : DAC Pair Disable (DACx_DIS)  : All Dac Pairs enabled
   * bit[0] : Power Down (PDN)               : Not powered down
   */
  i2c.write_reg(CS4384_ADDR, CS4384_MODE_CTRL, 0b10000000);

  unsigned adc_dif = 0x01;  // I2S mode
  unsigned adc_mode = 0x03; // Slave mode all speeds

  /* Reg 0x01: (GCTL) Global Mode Control Register */
  /* Bit[7]: CP-EN: Manages control-port mode
   * Bit[6]: CLKMODE: Setting puts part in 384x mode
   * Bit[5:4]: MDIV[1:0]: Set to 01 for /2
   * Bit[3:2]: DIF[1:0]: Data Format: 0x01 for I2S, 0x02 for TDM
   * Bit[1:0]: MODE[1:0]: Mode: 0x11 for slave mode
   */
  i2c.write_reg(CS5368_ADDR, CS5368_GCTL_MDE, 0b10010000 | (adc_dif << 2) | adc_mode);

  /* Reg 0x06: (PDN) Power Down Register */
  /* Bit[7:6]: Reserved
   * Bit[5]: PDN-BG: When set, this bit powers-own the bandgap reference
   * Bit[4]: PDM-OSC: Controls power to internal oscillator core
   * Bit[3:0]: PDN: When any bit is set all clocks going to that channel pair are turned off
   */
  i2c.write_reg(CS5368_ADDR, CS5368_PWR_DN, 0b00000000);
}
```

If porting this application to a different xCORE development board then this is the function that will have to be modified to configure the relevant codecs for that board.

## 2.6 The i2s_loopback application

The $I^2S$ loopback task provides the function of a digital loopback so that all $I^2S$ samples received by the device will be forwarded on.

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

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

The function takes a number of arguments:

```
[[distributable]]
void i2s_loopback(server i2s_callback_if i2s,
                  client i2c_master_if i2c,
                  client output_gpio_if dac_reset,
                  client output_gpio_if adc_reset,
                  client output_gpio_if pll_select,
                  client output_gpio_if mclk_select)
```

The interface to the $I^2S$ master is a callback interface that the $I^2S$ master will call over when it has received data or is ready to send data.

The $I^2C$ interface is used to configure the codecs and the GPIO interfaces abstract the outputting of single-bit values on a multi-bit port.

The body of the loopback task handles the I$^2$S interface calls:

```
  int32_t samples[8] = {0, 0, 0, 0, 0, 0, 0, 0};
  while (1) {
    select {
    case 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)/64;

      // Set CODECs in reset
      dac_reset.output(0);
      adc_reset.output(0);

      // Select 48Khz family clock (24.576Mhz)
      mclk_select.output(1);
      pll_select.output(0);

      // Allow the clock to settle
      delay_milliseconds(2);

      // Take CODECs out of reset
      dac_reset.output(1);
      adc_reset.output(1);

      reset_codecs(i2c);
      break;

    case i2s.receive(size_t index, int32_t sample):
      samples[index] = sample;
      break;

    case i2s.send(size_t index) -> int32_t sample:
      sample = samples[index];
      break;

    case i2s.restart_check() -> i2s_restart_t restart:
      restart = I2S_NO_RESTART;
      break;
    }
  }
}
```

The I$^2$S master library calls the `init()` method before it starts any data streaming. This 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 an audio sample on the channel specified by `index`. The sample is stored in the `samples` array.

The `send()` interface method is called when the master needs a new sample to send on the specified channel. The `->` symbol shows the return value expected from this call. In this case the application simply returns the current sample on that channel.

Finally, the `restart_check()` method is called by the I$^2$S master once per frame and 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`.

# APPENDIX A - Demo hardware setup

The demo is designed to run on the xCORE-200 Multichannel Audio Platform. To run the demo:

- Connect the XTAG-2 or XTAG-3 USB debug adapter to the xCORE-200 Multichannel Audio Platform.
- Connect the XTAG-2 or XTAG-3 to the host PC using USB extension cable.
- 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.
- Connect the 12V power adaptor to the xCORE-200 Multichannel Audio Platform.

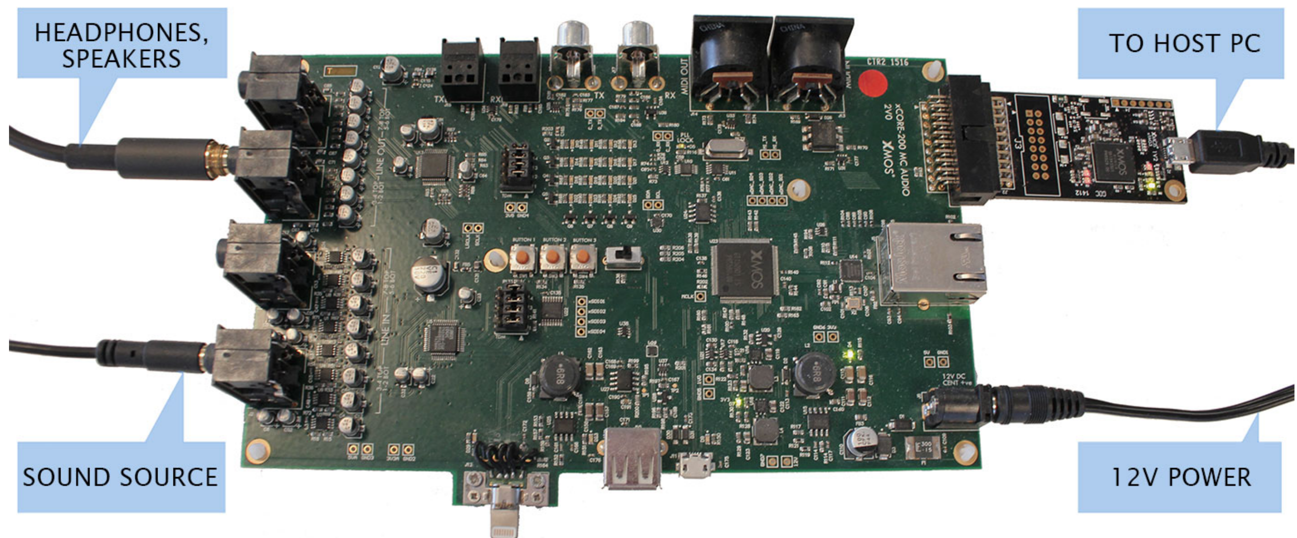


Figure 2: Hardware setup for XMOS I$^2$S master loopback demo

# APPENDIX B - Launching the demo device

Once the application source code is imported into the tools it can be built by pressing the **Build** button in the xTIMEcomposer. This will create the AN00162_i2s_loopback_demo.xe binary in the bin folder of the project. xTIMEcomposer may have to import the dependent libraries if you do not already have them in your workspace; this will occur automatically on build.

A *Run Configuration* then needs to be set up. This can be done by selecting the **Run** ▸ **Run Configurations...** menu. You can create a new run configuration by right clicking on the **xCORE application** group in the left hand pane and **New**.

Select your XMOS XTAG in the Target box and click **Apply**. You can now click the **Run** button to launch the application.

The audio from the input should now be playing on the corresponding output channel. All 8 channels should be active.

Alternatively, the command line tools can be used to build and run the application. Building is done by changing to the folder containing the src sub-folder and doing a call to xmake. Running the application is then done using xrun bin/AN00162_i2s_loopback_demo.xe.

# APPENDIX C - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

XMOS I$^2$S/TDM Library

http://www.xmos.com/support/libraries/lib_i2s

XMOS I$^2$C Library

http://www.xmos.com/support/libraries/lib_i2c

XMOS GPIO Library

http://www.xmos.com/support/libraries/lib_gpio

I$^2$S Protocol

https://en.wikipedia.org/wiki/I%C2%B2S

# APPENDIX D - Full source code listing

## D.1 Source code for main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <platform.h>
#include <xs1.h>
#include "i2s.h"
#include "i2c.h"
#include "gpio.h"

/* Ports and clocks used by the application */
on tile[0]: out buffered port:32 p_lrclk = XS1_PORT_1G;
on tile[0]: out buffered port:32 p_bclk = XS1_PORT_1H;
on tile[0]: in port p_mclk = XS1_PORT_1F;
on tile[0]: out buffered port:32 p_dout[4] = {XS1_PORT_1M, XS1_PORT_1N, XS1_PORT_1O, XS1_PORT_1P};
on tile[0]: in buffered port:32 p_din[4] = {XS1_PORT_1I, XS1_PORT_1J, XS1_PORT_1K, XS1_PORT_1L};

on tile[0]: clock mclk = XS1_CLKBLK_1;
on tile[0]: clock bclk = XS1_CLKBLK_2;

on tile[0]: port p_i2c = XS1_PORT_4A;
on tile[0]: port p_gpio = XS1_PORT_8C;

#define SAMPLE_FREQUENCY 48000
#define MASTER_CLOCK_FREQUENCY 24576000

#define CS5368_ADDR           0x4C // I2C address of the CS5368 DAC
#define CS5368_GCTL_MDE       0x01 // I2C mode control register number
#define CS5368_PWR_DN         0x06

#define CS4384_ADDR           0x18 // I2C address of the CS4384 ADC
#define CS4384_MODE_CTRL      0x02 // I2C mode control register number
#define CS4384_PCM_CTRL       0x03 // I2C PCM control register number

enum gpio_shared_audio_pins {
  GPIO_DAC_RST_N = 1,
  GPIO_PLL_SEL = 5,     // 1 = CS2100, 0 = Phaselink clock source
  GPIO_ADC_RST_N = 6,
  GPIO_MCLK_FSEL = 7,   // Select frequency on Phaselink clock. 0 = 24.576MHz for 48k, 1 = 22.5792MHz for 44.1
     ↪ k.
};

void reset_codecs(client i2c_master_if i2c)
{
  /* Mode Control 1 (Address: 0x02) */
  /* bit[7] : Control Port Enable (CPEN)     : Set to 1 for enable
   * bit[6] : Freeze controls (FREEZE)       : Set to 1 for freeze
   * bit[5] : PCM/DSD Selection (DSD/PCM)    : Set to 0 for PCM
   * bit[4:1] : DAC Pair Disable (DACx_DIS)  : All Dac Pairs enabled
   * bit[0] : Power Down (PDN)               : Powered down
   */
  i2c.write_reg(CS4384_ADDR, CS4384_MODE_CTRL, 0b11000001);

  /* PCM Control (Address: 0x03) */
  /* bit[7:4] : Digital Interface Format (DIF) : 0b1100 for TDM
   * bit[3:2] : Reserved
   * bit[1:0] : Functional Mode (FM) : 0x11 for auto-speed detect (32 to 200kHz)
   */
  i2c.write_reg(CS4384_ADDR, CS4384_PCM_CTRL, 0b00010111);

  /* Mode Control 1 (Address: 0x02) */
  /* bit[7] : Control Port Enable (CPEN)     : Set to 1 for enable
   * bit[6] : Freeze controls (FREEZE)       : Set to 0 for freeze
   * bit[5] : PCM/DSD Selection (DSD/PCM)    : Set to 0 for PCM
   * bit[4:1] : DAC Pair Disable (DACx_DIS)  : All Dac Pairs enabled
   * bit[0] : Power Down (PDN)               : Not powered down
   */
  i2c.write_reg(CS4384_ADDR, CS4384_MODE_CTRL, 0b10000000);

  unsigned adc_dif = 0x01;  // I2S mode
  unsigned adc_mode = 0x03; // Slave mode all speeds
```

```
  /* Reg 0x01: (GCTL) Global Mode Control Register */
  /* Bit[7]: CP-EN: Manages control-port mode
   * Bit[6]: CLKMODE: Setting puts part in 384x mode
   * Bit[5:4]: MDIV[1:0]: Set to 01 for /2
   * Bit[3:2]: DIF[1:0]: Data Format: 0x01 for I2S, 0x02 for TDM
   * Bit[1:0]: MODE[1:0]: Mode: 0x11 for slave mode
   */
  i2c.write_reg(CS5368_ADDR, CS5368_GCTL_MDE, 0b10010000 | (adc_dif << 2) | adc_mode);

  /* Reg 0x06: (PDN) Power Down Register */
  /* Bit[7:6]: Reserved
   * Bit[5]: PDN-BG: When set, this bit powers-own the bandgap reference
   * Bit[4]: PDM-OSC: Controls power to internal oscillator core
   * Bit[3:0]: PDN: When any bit is set all clocks going to that channel pair are turned off
   */
  i2c.write_reg(CS5368_ADDR, CS5368_PWR_DN, 0b00000000);
}

[[distributable]]
void i2s_loopback(server i2s_callback_if i2s,
                  client i2c_master_if i2c,
                  client output_gpio_if dac_reset,
                  client output_gpio_if adc_reset,
                  client output_gpio_if pll_select,
                  client output_gpio_if mclk_select)
{
  int32_t samples[8] = {0, 0, 0, 0, 0, 0, 0, 0};
  while (1) {
    select {
    case 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)/64;

      // Set CODECs in reset
      dac_reset.output(0);
      adc_reset.output(0);

      // Select 48Khz family clock (24.576Mhz)
      mclk_select.output(1);
      pll_select.output(0);

      // Allow the clock to settle
      delay_milliseconds(2);

      // Take CODECs out of reset
      dac_reset.output(1);
      adc_reset.output(1);

      reset_codecs(i2c);
      break;

    case i2s.receive(size_t index, int32_t sample):
      samples[index] = sample;
      break;

    case i2s.send(size_t index) -> int32_t sample:
      sample = samples[index];
      break;

    case i2s.restart_check() -> i2s_restart_t restart:
      restart = I2S_NO_RESTART;
      break;
    }
  }
}

static char gpio_pin_map[4] =  {
  GPIO_DAC_RST_N,
  GPIO_ADC_RST_N,
  GPIO_PLL_SEL,
  GPIO_MCLK_FSEL
};

int main()
{
```

```
  interface i2s_callback_if i_i2s;
  interface i2c_master_if i_i2c[1];
  interface output_gpio_if i_gpio[4];
  par {
    on tile[0]: {
      /* System setup, I2S + Codec control over I2C */
      configure_clock_src(mclk, p_mclk);
      start_clock(mclk);
      i2s_master(i_i2s, p_dout, 4, p_din, 4, p_bclk, p_lrclk, bclk, mclk);
    }

    on tile[0]: [[distribute]] i2c_master_single_port(i_i2c, 1, p_i2c, 100, 0, 1, 0);
    on tile[0]: [[distribute]] output_gpio(i_gpio, 4, p_gpio, gpio_pin_map);

    /* The application - loopback the I2S samples */
    on tile[0]: [[distribute]] i2s_loopback(i_i2s, i_i2c[0], i_gpio[0], i_gpio[1], i_gpio[2], i_gpio[3]);
  }
  return 0;
}
```

Copyright © 2016, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the "Information") and is providing it to you "AS IS" with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.