# I2S

## Overview

I2S (Inter-IC Sound) is a serial, synchronous communication protocol that is usually used for transmitting audio data between two digital audio devices.

An I2S bus consists of the following lines:

- Bit clock line
- Channel select line
- Serial data line

Each I2S controller has the following features that can be configured using the I2S driver:

- Operation as system master or slave
- Capable of acting as transmitter or receiver
- Dedicated DMA controller that allows for streaming sample data without requiring the CPU to copy each data sample

Each controller can operate in half-duplex communication mode. Thus, the two controllers can be combined to establish full-duplex communication.

I2S0 output can be routed directly to the digital-to-analog converter's (DAC) output channels (GPIO 25 & GPIO 26) to produce direct analog output without involving any external I2S codecs. I2S0 can also be used for transmitting PDM (Pulse-density modulation) signals.

The I2S peripherals also support LCD mode for communicating data over a parallel bus, as used by some LCD displays and camera modules. LCD mode has the following operational modes:

- LCD master transmitting mode
- Camera slave receiving mode
- ADC/DAC mode

**note**

For high accuracy clock applications, use the APLL_CLK clock source, which has the frequency range of 16 ~ 128 MHz. You can enable the APLL_CLK clock source by setting :cppi2s_config_t::use_apll to `TRUE`.

If :cppi2s_config_t::use_apll = `TRUE` and :cppi2s_config_t::fixed_mclk > 0, then the master clock output frequency for I2S will be equal to the value of :cppi2s_config_t::fixed_mclk, which means that the mclk frequency is provided by the user, instead of being calculated by the driver.

The clock rate of the word select line, which is called audio left-right clock rate (LRCK) here, is always the divisor of the master clock output frequency and for which the following is always true: $0 < MCLK/LRCK/channels/bits\_per\_sample < 64$.

# Functional Overview

## Installing the Driver

Install the I2S driver by calling the function :cpp:func'i2s_driver_install' and passing the following arguments:

- Port number
- The structure :cppi2s_config_t with defined communication parameters
- Event queue size and handle

Configuration example:

```
static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX,
    .sample_rate = 44100,
    .bits_per_sample = 16,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = I2S_COMM_FORMAT_STAND_I2S,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

i2s_driver_install(I2S_NUM, &i2s_config, 0, NULL);
```

## Setting Communication Pins

Once the driver is installed, configure physical GPIO pins to which signals will be routed. For this, call the function :cpp:func'i2s_set_pin' and pass the following arguments to it:

- Port number
- The structure :cppi2s_pin_config_t defining the GPIO pin numbers to which the driver should route the BCK, WS, DATA out, and DATA in signals. If you want to keep a currently allocated pin number for a specific signal, or if this signal is unused, then pass the macro :cI2S_PIN_NO_CHANGE. See the example below.

```
static const i2s_pin_config_t pin_config = {
    .bck_io_num = 26,
    .ws_io_num = 25,
    .data_out_num = 22,
    .data_in_num = I2S_PIN_NO_CHANGE
};

i2s_set_pin(i2s_num, &pin_config);
```

## Running I2S Communication

To perform a transmission:

- Prepare the data for sending
- Call the function :cppi2s_write and pass the data buffer address and data length to it

The function will write the data to the I2S DMA Tx buffer, and then the data will be transmitted automatically.

```
i2s_write(I2S_NUM, samples_data, ((bits+8)/16)*SAMPLE_PER_CYCLE*4, &i2s_bytes_write, 100);
```

To retrieve received data, use the function :cppi2s_read. It will retrieve the data from the I2S DMA Rx buffer, once the data is received by the I2S controller.

You can temporarily stop the I2S driver by calling the function :cppi2s_stop, which will disable the I2S Tx/Rx units until the function :cppi2s_start is called. If the function :cpp:func'i2s_driver_install' is used, the driver will start up automatically eliminating the need to call :cppi2s_start.

**Deleting the Driver**

If the established communication is no longer required, the driver can be removed to free allocated resources by calling :cppi2s_driver_uninstall.

# Application Example

A code example for the I2S driver can be found in the directory peripherals/i2s.

In addition, there are two short configuration examples for the I2S driver.

## I2S configuration

```
#include "driver/i2s.h"
#include "freertos/queue.h"

static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX,
    .sample_rate = 44100,
    .bits_per_sample = 16,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = I2S_COMM_FORMAT_STAND_I2S,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

static const i2s_pin_config_t pin_config = {
    .bck_io_num = 26,
    .ws_io_num = 25,
    .data_out_num = 22,
    .data_in_num = I2S_PIN_NO_CHANGE
};

...

    i2s_driver_install(i2s_num, &i2s_config, 0, NULL);   //install and start i2s driver

    i2s_set_pin(i2s_num, &pin_config);

    i2s_set_sample_rates(i2s_num, 22050); //set sample rates
```

```
        i2s_driver_uninstall(i2s_num); //stop & destroy i2s driver
```

## Configuring I2S to use internal DAC for analog output

```
#include "driver/i2s.h"
#include "freertos/queue.h"

static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX | I2S_MODE_DAC_BUILT_IN,
    .sample_rate = 44100,
    .bits_per_sample = 16, /* the DAC module will only take the 8bits from MSB */
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

...

    i2s_driver_install(i2s_num, &i2s_config, 0, NULL);   //install and start i2s driver

    i2s_set_pin(i2s_num, NULL); //for internal DAC, this will enable both of the internal ch

    //You can call i2s_set_dac_mode to set built-in DAC output mode.
    //i2s_set_dac_mode(I2S_DAC_CHANNEL_BOTH_EN);

    i2s_set_sample_rates(i2s_num, 22050); //set sample rates

    i2s_driver_uninstall(i2s_num); //stop & destroy i2s driver
```

# API Reference