## 3.14 Speaker

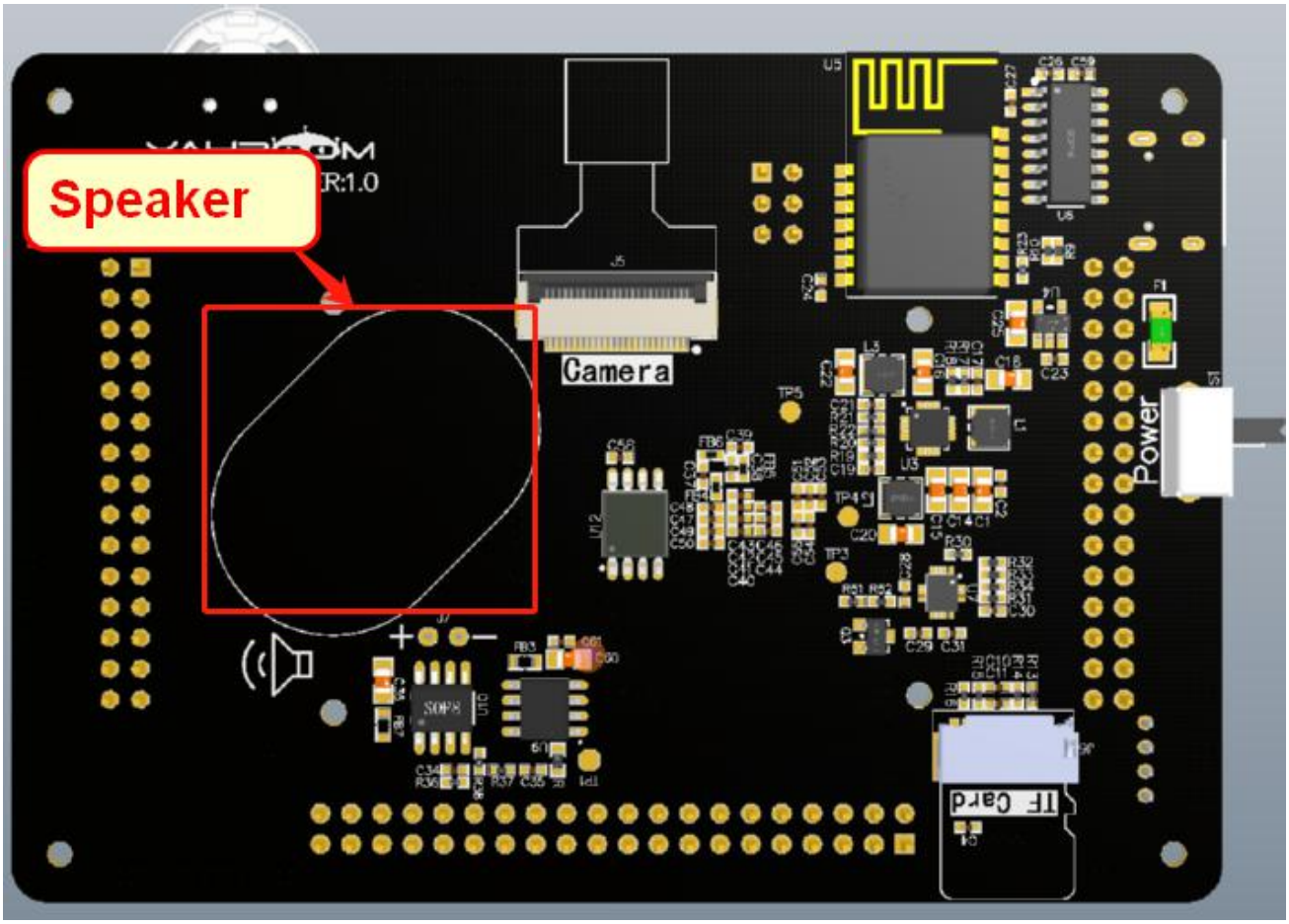
**1. Experiment purpose**

In this lesson, we will focus on how to make K210 play PCM audio data through I2S.

**2.Experiment preparation**

2.1 components

Speaker




2.2 Component characteristics

K210 development board has already welded the power amplifier and speaker by default.

The function of the power amplifier is to amplify the weak signal of the sound source, then make the speaker to sound. The power amplifier communicates with the K210 chip through the I2S protocol.

I2S bus is specially used for data transmission between audio devices, and it uses a synchronous serial communication interface.

There are 3 audio buses built in the integrated circuit ($I^2S0$, $I^2S1$, $I^2S2$), all in MASTER mode.

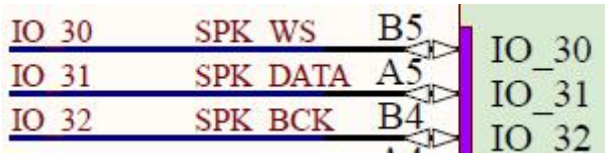
There are some common features:

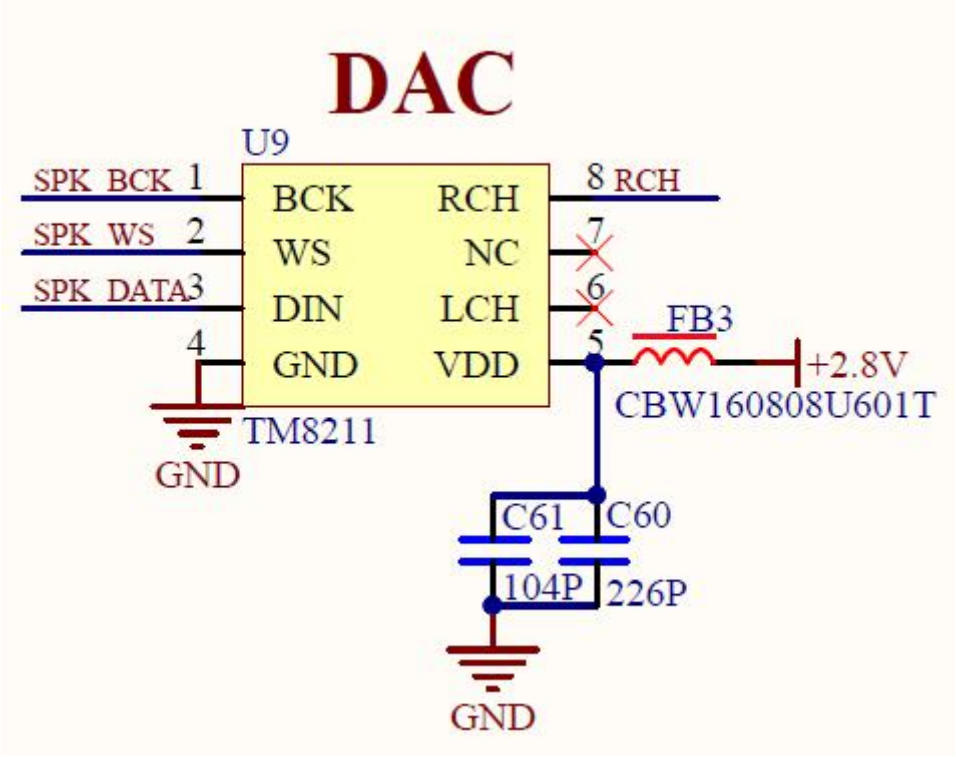
- The bus width can be configured as 8, 16, and 32 bits
- Each interface supports up to 4 stereo channels
- Due to the independence of the transmitter and receiver, full-duplex communication is supported

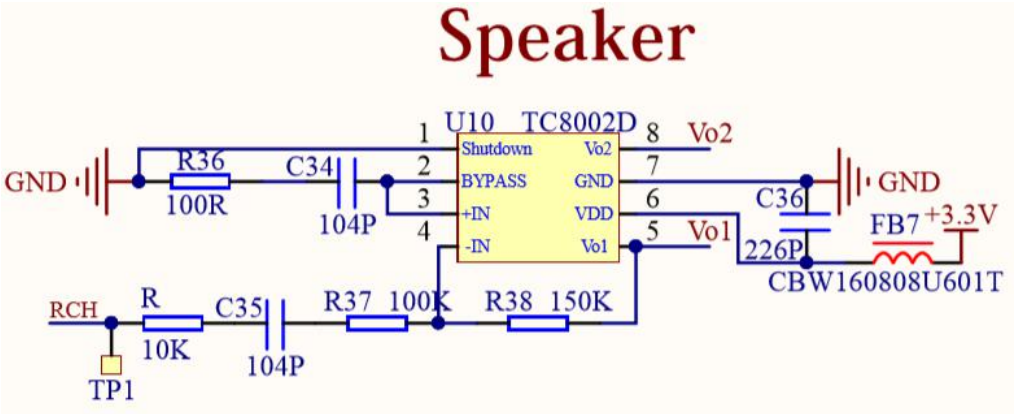
• Asynchronous clock of APB bus and I²S SCLK
• Audio data resolution is 12, 16, 20, 24 and 32 bits
• I²S0 transmit FIFO depth is 64 bytes, receive 8 bytes, I²S1 and I²S2 transmit and receive FIFO depth are both 8 bytes
• Support DMA transfer
• Programmable FIFO threshold

2.3 Hardware connection

K210 development board has already welded the speaker by default. SPK_WS connect IO30，SPK_DATA connect IO31，SKP_BCK connect IO32.



2.4 SDK API function

The header file is i2s.h

I2S standard bus defines three types of signals: clock signal BCK, channel selection signal WS and serial data signal SD.

A basic I2S data bus has a master and a slave. The roles of master and slave remain unchanged during the communication process. The I2S module contains independent transmit and receive channels to ensure excellent communication performance.

We will provide following interfaces to users:

• i2s_init: Initialize the I2S device, set the receive or transmit mode, and channel mask.
• i2s_send_data_dma: I2S send data.
• i2s_recv_data_dma: I2S receive data.
• i2s_rx_channel_config: Set the receiving channel parameters.
• i2s_tx_channel_config: Set the sending channel parameters.
• i2s_play: send PCM data, such as playing music
• i2s_set_sample_rate: Set the sampling rate.
• i2s_set_dma_divide_16: set dma_divide_16, set dma_divide_16 for 16-bit data, and automatically divide the 32-bit INT32 data into two 16-bit left and right channel data during DMA transmission.
• i2s_get_dma_divide_16: Get the value of dma_divide_16. Used to determine whether to set dma_divide_16.
• i2s_handle_data_dma: I2S transfers data through DMA.

**3. Experimental principle**

I2S possess 3 main signals:

1) Serial clock SCLK, it also called bit clock (BCLK), corresponds to each bit of digital audio data, SCLK has 1 pulse. SCLK frequency = 2 × sampling frequency × number of sampling bits.

2) The frame clock LRCK is used to switch the data of the left and right channels.

LRCK is"1" , which means that the data of the left channel is being transmitted,

LRCK is"1", which means that the data of the right channel is being transmitted.

The frequency of LRCK is equal to the sampling frequency.

3) Serial data SDATA is audio data expressed in twos complement.

**4. Experiment procedure**

4.1 According to the above hardware connection pin diagram, K210 hardware pins and software functions use FPIOA mapping relationship.

```
#ifndef _PIN_CONFIG_H_
#define _PIN_CONFIG_H_
/*****************************HEAR-FILE*************************************/
#include "fpioa.h"

/*****************************HARDWARE-PIN**********************************/
//Hardware IO port, corresponding Schematic
#define PIN_SPK_WS              (30)
#define PIN_SPK_DATA            (31)
#define PIN_SPK_BCK             (32)

/*****************************SOFTWARE-GPIO*********************************/
//Software GPIO port, corresponding program


/*****************************FUNC-GPIO*************************************/
//Function of GPIO port, bound to hardware IO po
#define FUNC_SPK_WS             FUNC_I2S2_WS
#define FUNC_SPK_DATA           FUNC_I2S2_OUT_D0
#define FUNC_SPK_BCK            FUNC_I2S2_SCLK
```

```
void hardware_init(void)
{
    /*
    ** SPK_WS---IO30
    ** SPK_DATA-IO31
    ** SPK_BCK--IO32
    */
    fpioa_set_function(PIN_SPK_WS, FUNC_SPK_WS);
    fpioa_set_function(PIN_SPK_DATA, FUNC_SPK_DATA);
    fpioa_set_function(PIN_SPK_BCK, FUNC_SPK_BCK);
}
```

4.2 Set the system clock frequency. Since the Uarths 'clock is from PLL0, it is necessary to re-initialize the following Uarths after setting PLL0, otherwise printf may print confusing code.

```
/* set the system clock frequency */
sysctl_pll_set_freq(SYSCTL_PLL0, 320000000UL);
sysctl_pll_set_freq(SYSCTL_PLL1, 160000000UL);
sysctl_pll_set_freq(SYSCTL_PLL2, 45158400UL);
uarths_init();
```

4.3 Initialize I2S, the third parameter is to set the channel mask, channel 0: 0x03, channel 1: 0x0C, channel 2: 0x30, channel 3: 0xC0

```
/* Initialize I2S, the third parameter is to set the c
i2s_init(I2S_DEVICE_2, I2S_TRANSMITTER, 0x03);
```

4.4 Set the parameters of I2S sending data channel 0, receive data bit 16bit, single clock data bit 32, DMA trigger FIFO depth is 4, working mode is right channel.

```
/* Set the channel parameters of I2S sending data*/
i2s_tx_channel_config(
    I2S_DEVICE_2, /* I2S device number*/
    I2S_CHANNEL_0, /* I2S channel */
    RESOLUTION_16_BIT, /* Number of data received */
    SCLK_CYCLES_32, /* Number of single data clocks */
    TRIGGER_LEVEL_4, /* FIFO depth when DMA is triggered*/
    RIGHT_JUSTIFYING_MODE); /* work mode/
```

4.5 i2S plays music, it needs to use a DMA channel.

```
/* Play the music once. If you need to play it again, please press
i2s_play(
    I2S_DEVICE_2, /* I2S device number*/
    DMAC_CHANNEL0, /* DMA channel number */
    (uint8_t *)test_pcm, /* Playback of PCM data */
    sizeof(test_pcm), /* Length of PCM data */
    1024, /* ingle delivery quantity */
    16, /* Single sampling bit width */
    2); /* Track number */
```

4.6 while(1) loop, Play the music once. If you need to play it again, please press the reset button.

```
while (1);
return 0;
```

4.7 Compile and debug, burn and run

Copy the sdcard to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

**cmake .. -DPROJ=play_pcm -G "MinGW Makefiles"**

**make**

```
[ 97%] Building C object CMakeFiles/play_pcm.dir/src/play_pcm/main.c.obj
[100%] Linking C executable play_pcm
Generating .bin file ...
[100%] Built target play_pcm
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **play_pcm.bin** file will be generated in the build folder.

We need to use the type-C data cable to connect the computer and the K210 development board. Open kflash, select the corresponding device, and then burn the **play_pcm.bin** file to the K210 development board.

**5. Experimental phenomenon**

K210 development board plays the music once and then stops. When you need to play the music again, please press the reset button of the K210 development board。

**6. Experiment summary**

6.1 Speaker does not directly receive the K210 chip data. The signal is amplified by the power amplifier after DAC conversion, and finally the signal received by the speaker.

6.2 The speaker needs to use DMA channel to play music.

6.3 I2S is a protocol dedicated to sound transmission, only need three wires to transmit music data.

**Appendix -- API**

**i2s_init**

Description: Initialize I2S.

Function prototype: **void i2s_init(i2s_device_number_t device_num, i2s_transmit_t rxtx_mode, uint32_t channel_mask)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| rxtx_mode | Receive or send mode | Input |
| channel_mask | Channel mask | Input |

Return value: No

**i2s_send_data_dma**

Description: I2S send data

Function prototype: **void i2s_send_data_dma(i2s_device_number_t device_num, const void *buf, size_t buf_len, dmac_channel_number_t channel_num)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| buf | send data address | Input |
| buf_len | length of data | Input |
| channel_num | DMA channel number | Input |

Return value: No

**i2s_recv_data_dma**

Description: I2S receive data

Function prototype: **void i2s_recv_data_dma(i2s_device_number_t device_num, uint32_t *buf, size_t buf_len, dmac_channel_number_t channel_num)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| buf | Receive data address | Output |
| buf_len | length of data | Input |
| channel_num | DMA channel number | Input |

Return value: No

**i2s_rx_channel_config**

Description: Set the receiving channel parameters.

Function prototype: **void i2s_rx_channel_config(i2s_device_number_t device_num, i2s_channel_num_t channel_num, i2s_word_length_t word_length, i2s_word_select_cycles_t word_select_size, i2s_fifo_threshold_t trigger_level, i2s_work_mode_t word_mode)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| channel_num | Channel number | Input |
| word_length | Number of received data | Output |
| word_select_size | Number of single data clocks | Input |
| trigger_level | FIFO depth when DMA trigger | Input |
| word_mode | Working mode | Input |

Return value: No

**i2s_tx_channel_config**

Description: Set the sending channel parameters.

Function prototype: **void i2s_tx_channel_config(i2s_device_number_t device_num, i2s_channel_num_t channel_num, i2s_word_length_t word_length, i2s_word_select_cycles_t word_select_size, i2s_fifo_threshold_t trigger_level, i2s_work_mode_t word_mode)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| channel_num | Channel number | Input |
| word_length | Number of received data | Output |
| word_select_size | Number of single data clocks | Input |
| trigger_level | FIFO depth when DMA trigger | Input |
| word_mode | Working mode | Input |

Return value: No

**i2s_play**

Description: Send PCM data, such as, play music

Function prototype: **void i2s_play(i2s_device_number_t device_num, dmac_channel_number_t channel_num, const uint8_t *buf, size_t buf_len, size_t frame, size_t bits_per_sample, uint8_t track_num)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| channel_num | Channel number | Input |
| buf | PCM data | Input |
| buf_len | Length of PCM data | Input |
| frame | Number of single sending | Input |
| bits_per_sample | Single sampling bit width | Input |
| track_num | Number of channels | Input |

Return value: No

**i2s_set_sample_rate**

Description: Set the sampling rate.

Function prototype: **uint32_t i2s_set_sample_rate(i2s_device_number_t device_num, uint32_t sample_rate)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| sample_rate | Sampling rate | Input |

Return value: The actual sampling rate.

**i2s_set_dma_divide_16**

Description: Set dma_divide_16, set dma_divide_16 for 16-bit data, and automatically divide the 32-bit INT32 data into two 16-bit left and right channel data during DMA transmission.

Function prototype: **int i2s_set_dma_divide_16(i2s_device_number_t device_num, uint32_t enable)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| enable | 0-disable 1-enable | Input |

Return value:

| Return value | Description |
|---|---|
| 0 | Successful |
| !0 | Failure |

**i2s_get_dma_divide_16**

Description: Gets the dMA_divide_16 value, which used to determine if dMA_divide_16 needs to be set.

Function prototype: **int i2s_get_dma_divide_16(i2s_device_number_t device_num)**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |

Return value:

| Return value | Description |
|---|---|
| 1 | enable |
| 0 | disable |
| <0 | Failure |

**i2s_handle_data_dma**

Description: I2S transfers data through DMA.

Function prototype: **void i2s_handle_data_dma(i2s_device_number_t device_num, i2s_data_t data, plic_interrupt_t *cb);**

Parameter:

| Parameter name | Description | Input/Output |
|---|---|---|
| device_num | I2S number | Input |
| data | I2S data related parameters | Input |
| cb | DMA interrupt callback function, if it is set to NULL, it is in blocking mode, and the function exits after the transmission is completed | Input |

Return value: No

**Eg:**

```
/* I2S0 channel 0 is set as the receiving channel, receiving 16-bit data, transmitting 32 clocks at a time, FIFO depth is 4, standard mode. Receive 8 sets of data*/
/* I2S2 channel 1 is set as the sending channel, sending 16-bit data, single transmission 32 clocks, FIFO depth is 4, right-justified mode. Send 8 sets of data*/
uint32_t buf[8];
i2s_init(I2S_DEVICE_0, I2S_RECEIVER, 0x3);
i2s_init(I2S_DEVICE_2, I2S_TRANSMITTER, 0xC);
i2s_rx_channel_config(I2S_DEVICE_0, I2S_CHANNEL_0, RESOLUTION_16_BIT, SCLK_CYCLES_32, TRIGGER_LEVEL_4, STANDARD_MODE);
i2s_tx_channel_config(I2S_DEVICE_2, I2S_CHANNEL_1, RESOLUTION_16_BIT, SCLK_CYCLES_32, TRIGGER_LEVEL_4, RIGHT_JUSTIFYING_MODE);
i2s_recv_data_dma(I2S_DEVICE_0, rx_buf, 8, DMAC_CHANNEL1);
i2s_send_data_dma(I2S_DEVICE_2, buf, 8, DMAC_CHANNEL0);
```

**Data type**

The related data types and data structure are defined as follows:

i2s_device_number_t: I2S number.

i2s_channel_num_t: I2S channel number.

i2s_transmit_t: I2S transmit.

i2s_work_mode_t: I2S Working mode.
i2s_word_select_cycles_t: I2S single transmission clock number.
i2s_word_length_t: I2S transmission data bits
i2s_fifo_threshold_t: I2S FIFO depth.
i2s_data_t: Data related parameters during DMA transfer.
i2s_transfer_mode_t: I2S transfer mode.

**i2s_device_number_t**
Description: I2S number
**Define**

```
typedef enum _i2s_device_number
{
    I2S_DEVICE_0 = 0,
    I2S_DEVICE_1 = 1,
    I2S_DEVICE_2 = 2,
    I2S_DEVICE_MAX
} i2s_device_number_t;
```

**member**

| Member name | Description |
|---|---|
| I2S_DEVICE_0 | I2S 0 |
| I2S_DEVICE_1 | I2S 1 |
| I2S_DEVICE_2 | I2S 2 |

**i2s_channel_num_t**
Description: I2S channel number
**Define**

```
typedef enum _i2s_channel_num
{
    I2S_CHANNEL_0 = 0,
    I2S_CHANNEL_1 = 1,
    I2S_CHANNEL_2 = 2,
    I2S_CHANNEL_3 = 3
} i2s_channel_num_t;
```

**member**

| Member name | Description |
|---|---|
| I2S_CHANNEL_0 | I2S channel 0 |
| I2S_CHANNEL_1 | I2S channel 1 |
| I2S_CHANNEL_2 | I2S channel 2 |
| I2S_CHANNEL_3 | I2S channel 3 |

**i2s_transmit_t**

Description: I2S transmit mode

**Define**

```
typedef enum _i2s_transmit
{
    I2S_TRANSMITTER = 0,
    I2S_RECEIVER = 1
} i2s_transmit_t;
```

**member**

| Member name | Description |
|---|---|
| I2S_TRANSMITTER | Sending mode |
| I2S_RECEIVER | Receiving mode |

**i2s_work_mode_t**

Description: I2S working mode

**Define**

```
typedef enum _i2s_work_mode
{
    STANDARD_MODE = 1,
    RIGHT_JUSTIFYING_MODE = 2,
    LEFT_JUSTIFYING_MODE = 4
} i2s_work_mode_t;
```

**member**

| Member name | Description |
|---|---|
| STANDARD_MODE | standard mode |
| RIGHT_JUSTIFYING_MODE | right justifying mode |
| LEFT_JUSTIFYING_MODE | left justifying mode |

**i2s_word_select_cycles_t**

Description: I2S single transmission clock number

**Define**

```
typedef enum _word_select_cycles
{
    SCLK_CYCLES_16 = 0x0,
    SCLK_CYCLES_24 = 0x1,
    SCLK_CYCLES_32 = 0x2
} i2s_word_select_cycles_t;
```

**member**

| Member name | Description |
|---|---|
| SCLK_CYCLES_16 | 16 clocks |
| SCLK_CYCLES_24 | 24 clocks |
| SCLK_CYCLES_32 | 32 clocks |

**i2s_word_length_t**

Description: I2S transmits data bits.

**Define**

```
typedef enum _word_length
{
    IGNORE_WORD_LENGTH = 0x0,
    RESOLUTION_12_BIT = 0x1,
    RESOLUTION_16_BIT = 0x2,
    RESOLUTION_20_BIT = 0x3,
    RESOLUTION_24_BIT = 0x4,
    RESOLUTION_32_BIT = 0x5
} i2s_word_length_t;
```

**member**

| Member name | Description |
|---|---|
| IGNORE_WORD_LENGTH | Ignore the length |
| RESOLUTION_12_BIT | 12-bit data length |
| RESOLUTION_16_BIT | 16-bit data length |
| RESOLUTION_20_BIT | 20-bit data length |
| RESOLUTION_24_BIT | 24-bit data length |
| RESOLUTION_32_BIT | 32-bit data length |

**i2s_fifo_threshold_t**

Description: I2S FIFO depth

**Define**

```
typedef enum _fifo_threshold
{
    /* Interrupt trigger when FIFO level is 1 */
    TRIGGER_LEVEL_1 = 0x0,
    /* Interrupt trigger when FIFO level is 2 */
    TRIGGER_LEVEL_2 = 0x1,
    /* Interrupt trigger when FIFO level is 3 */
    TRIGGER_LEVEL_3 = 0x2,
    /* Interrupt trigger when FIFO level is 4 */
    TRIGGER_LEVEL_4 = 0x3,
    /* Interrupt trigger when FIFO level is 5 */
    TRIGGER_LEVEL_5 = 0x4,
```

```
    /* Interrupt trigger when FIFO level is 6 */
    TRIGGER_LEVEL_6 = 0x5,
    /* Interrupt trigger when FIFO level is 7 */
    TRIGGER_LEVEL_7 = 0x6,
    /* Interrupt trigger when FIFO level is 8 */
    TRIGGER_LEVEL_8 = 0x7,
    /* Interrupt trigger when FIFO level is 9 */
    TRIGGER_LEVEL_9 = 0x8,
    /* Interrupt trigger when FIFO level is 10 */
    TRIGGER_LEVEL_10 = 0x9,
    /* Interrupt trigger when FIFO level is 11 */
    TRIGGER_LEVEL_11 = 0xa,
    /* Interrupt trigger when FIFO level is 12 */
    TRIGGER_LEVEL_12 = 0xb,
    /* Interrupt trigger when FIFO level is 13 */
    TRIGGER_LEVEL_13 = 0xc,
    /* Interrupt trigger when FIFO level is 14 */
    TRIGGER_LEVEL_14 = 0xd,
    /* Interrupt trigger when FIFO level is 15 */
    TRIGGER_LEVEL_15 = 0xe,
    /* Interrupt trigger when FIFO level is 16 */
    TRIGGER_LEVEL_16 = 0xf
} i2s_fifo_threshold_t;
```

**member**

| Member name | Description |
|---|---|
| TRIGGER_LEVEL_1 | 1 byte FIFO depth |
| TRIGGER_LEVEL_2 | 2 byte FIFO depth |
| TRIGGER_LEVEL_3 | 3 byte FIFO depth |
| TRIGGER_LEVEL_4 | 4 byte FIFO depth |
| TRIGGER_LEVEL_5 | 5 byte FIFO depth |
| TRIGGER_LEVEL_6 | 6 byte FIFO depth |
| TRIGGER_LEVEL_7 | 7 byte FIFO depth |
| TRIGGER_LEVEL_8 | 8 byte FIFO depth |
| TRIGGER_LEVEL_9 | 9 byte FIFO depth |
| TRIGGER_LEVEL_10 | 10 byte FIFO depth |
| TRIGGER_LEVEL_11 | 11 byte FIFO depth |
| TRIGGER_LEVEL_12 | 12 byte FIFO depth |
| TRIGGER_LEVEL_13 | 13 byte FIFO depth |
| TRIGGER_LEVEL_14 | 14 byte FIFO depth |
| TRIGGER_LEVEL_15 | 15 byte FIFO depth |
| TRIGGER_LEVEL_16 | 16 byte FIFO depth |

**i2s_data_t**

Description: Data related parameters during DMA transfer.

**Define**

```
typedef struct _i2s_data_t
{
    dmac_channel_number_t tx_channel;
    dmac_channel_number_t rx_channel;
    uint32_t *tx_buf;
    size_t tx_len;
    uint32_t *rx_buf;
    size_t rx_len;
    i2s_transfer_mode_t transfer_mode;
    bool nowait_dma_idle;
    bool wait_dma_done;
} i2s_data_t;
```

**member**

| Member name | Description |
|---|---|
| tx_channel | The DMA channel number used when sending |
| rx_channel | The DMA channel number used when receive |
| tx_buf | Data sent |
| tx_len | Length of data sent |
| rx_buf | Data received |
| rx_len | Length of data received |
| transfer_mode | Transfer mode, send or receive |
| nowait_dma_idle | Whether to wait for the DMA channel to be free before DMA transfer |
| wait_dma_done | Whether to wait for the completion of the transfer after DMA transfer, if cb is not empty, this function is invalid |

**i2s_transfer_mode_t**

Description: I2S transfer mode

**Define**

```
typedef enum _i2s_transfer_mode
{
    I2S_SEND,
    I2S_RECEIVE,
} i2s_transfer_mode_t;
```

**member**

| Member name | Description |
|---|---|
| I2S_SEND | send |
| I2S_RECEIVE | software |