3.15 MIC recording

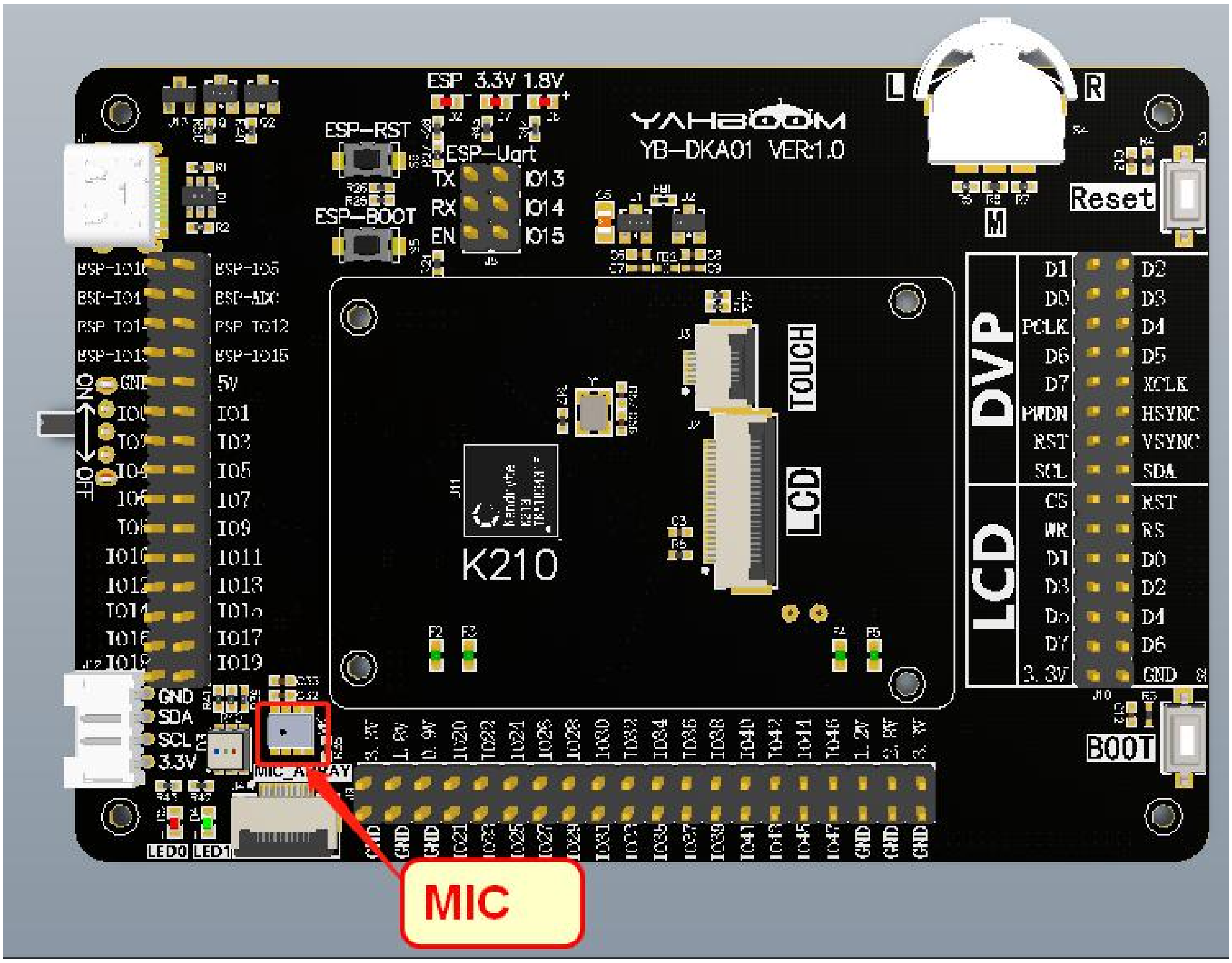
1. Experiment purpose

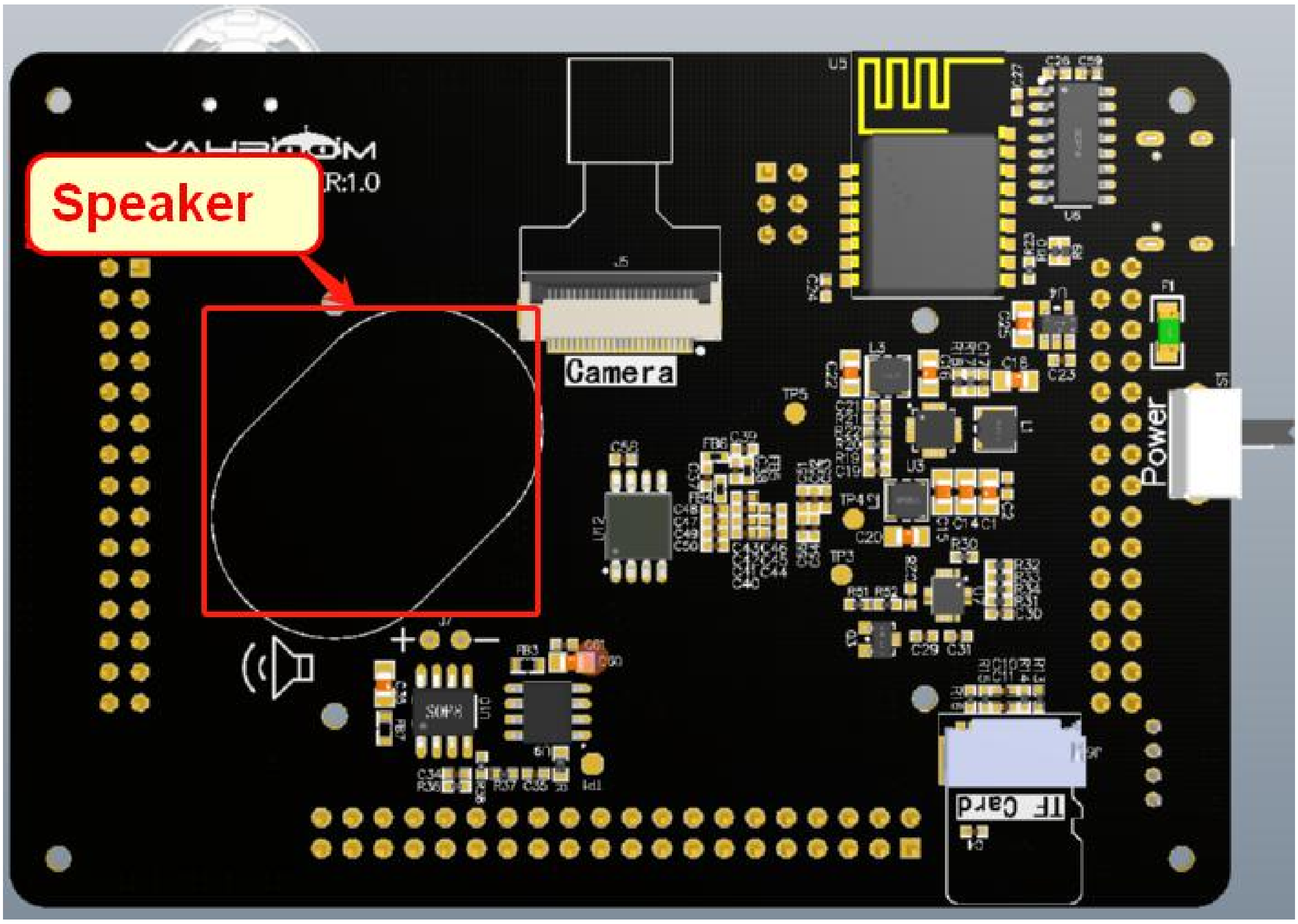
In this lesson, we mainly learn K210 receiving and sending through I2S, to achieve microphone recording and speaker playing.

2.Experiment preparation

2.1 components

Speaker, microphone



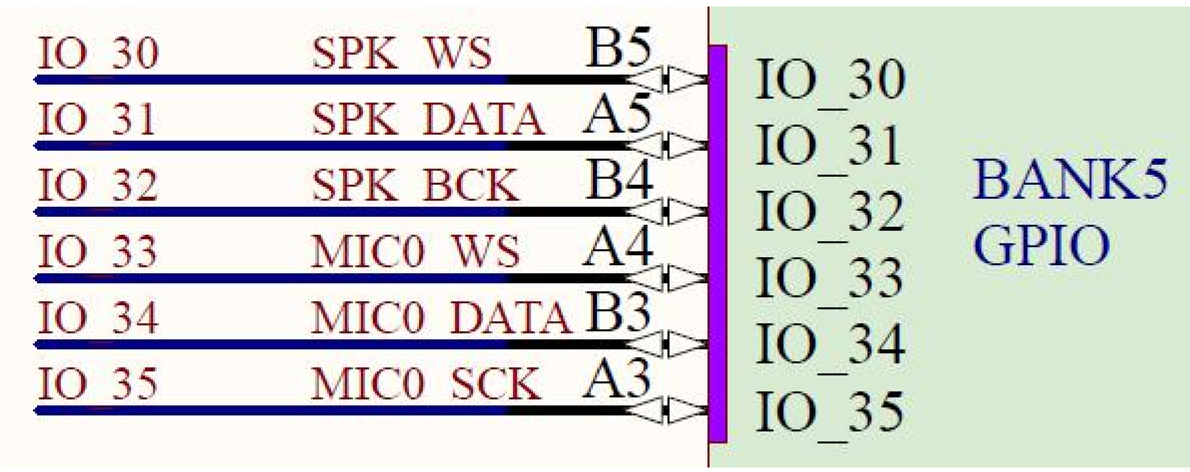


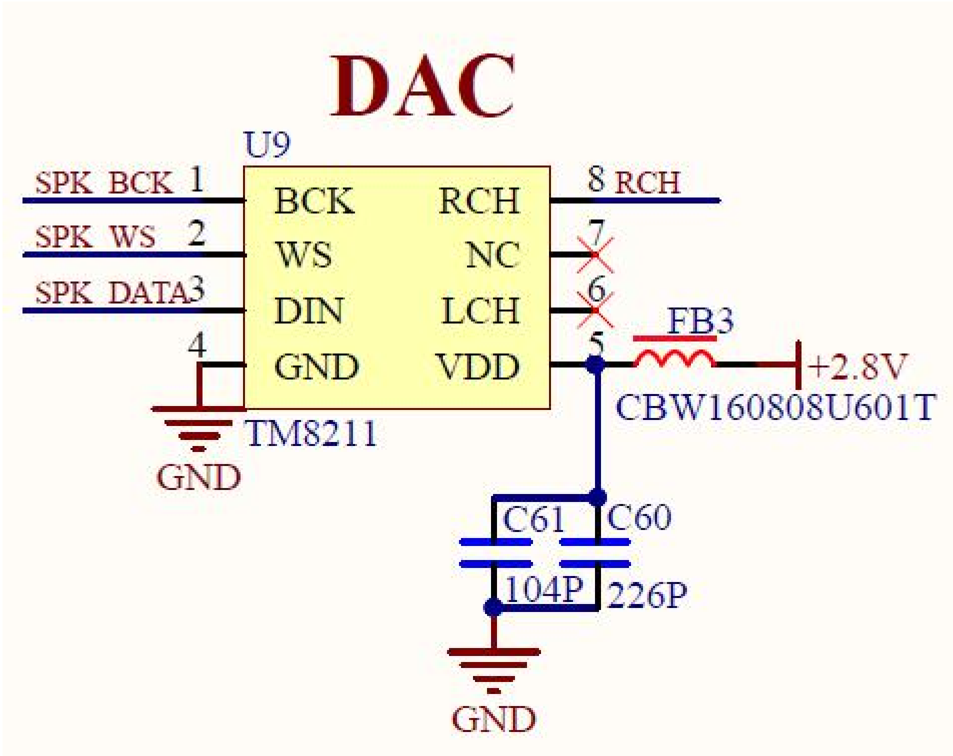
2.2 Component characteristics

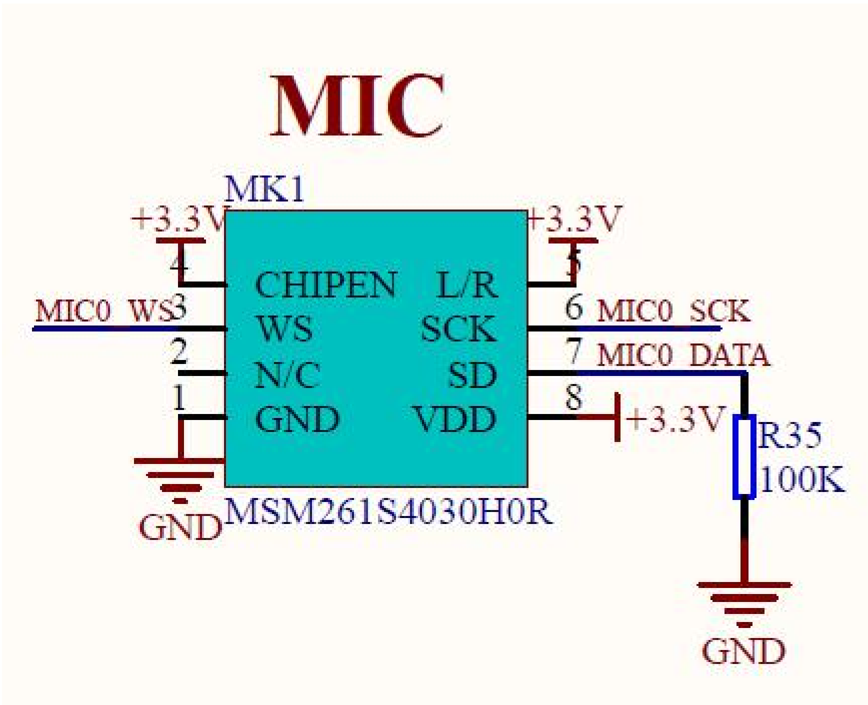
K210 development board's microphone is in I2S input mode.

2.3 Hardware connection

K210 development board has been welded with microphones, speakers and related accessories by default. SPK_WS is connected to IO30, SPK_DATA is connected to IO31, and SKP_BCK is connected to 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 possess a master and a slave.

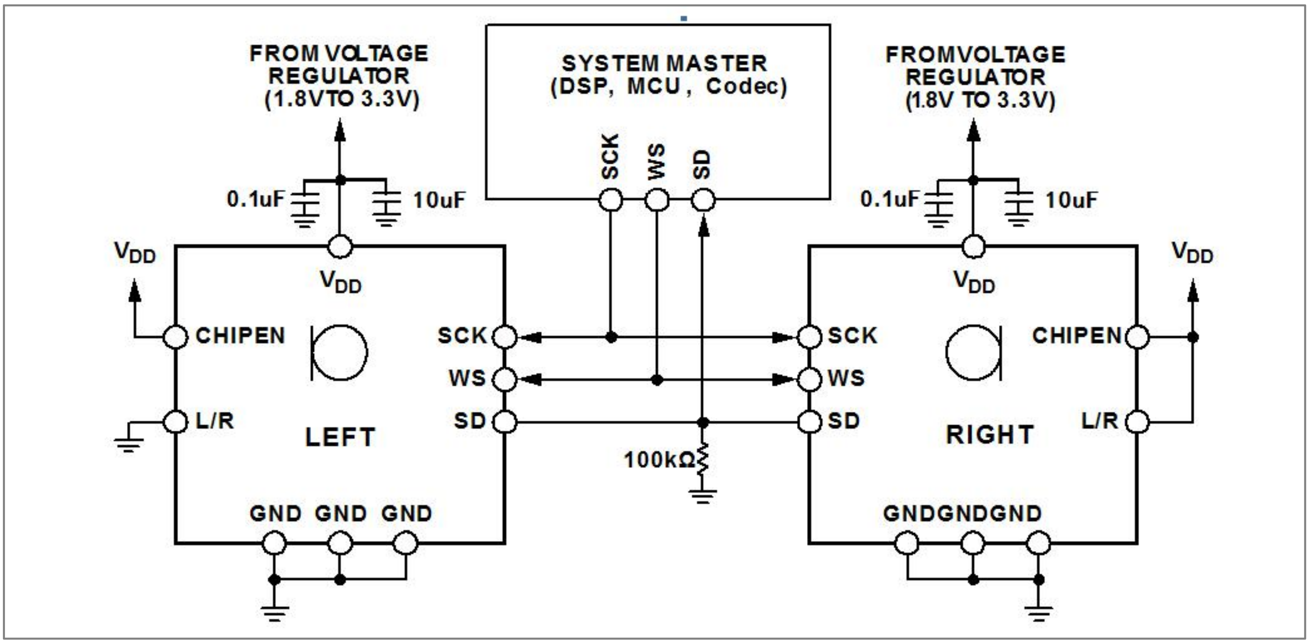
We will provide following interfaces to users:

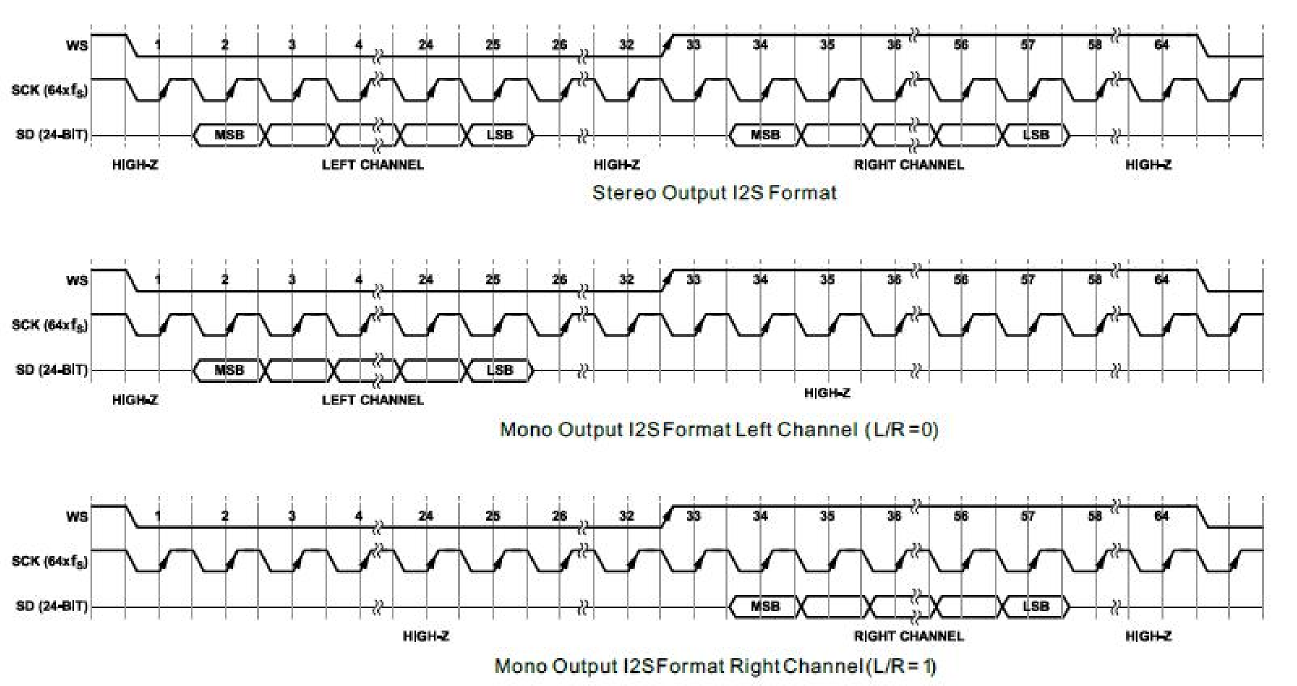
• i2s_init: Initialize I2S device, set receive or send mode, channel mask.
• i2s_send_data_dma: I2S send data.
• i2s_recv_data_dma: I2S receive data.
• i2s_rx_channel_config: Set the receive channel parameters.
• i2s_tx_channel_config: Set the send channel parameters.
• i2s_play: Send PCM data, such as play music
• i2s_set_sample_rate: Set 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 dma_divide_16 value, which used to determine whether to set dma_divide_16.
• i2s_handle_data_dma:I2S transfers data through DMA.

3. Experimental principle

MEMS (Micro Electro Mechanical System) microphones are generally composed of MEMS microcapacitance sensors, micro integrated conversion circuits, acoustic cavities, and RF anti-interference circuits.

As shown in the figure below, the microphone reads data through I2S. When WS is low, the data is collected from the left channel, and when WS is high, the data is collected from the right channel.





4. Experiment procedure

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

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

#define PIN_MIC0_WS            (33)
#define PIN_MIC0_DATA          (34)
#define PIN_MIC0_SCK           (35)

/*****************************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

#define FUNC_MIC0_WS           FUNC_I2S0_WS
#define FUNC_MIC0_DATA         FUNC_I2S0_IN_D1
#define FUNC_MIC0_SCK          FUNC_I2S0_SCLK
```

```
void hardware_init(void)
{
    /* mic */
    fpioa_set_function(PIN_MIC0_WS,   FUNC_MIC0_WS);
    fpioa_set_function(PIN_MIC0_DATA, FUNC_MIC0_DATA);
    fpioa_set_function(PIN_MIC0_SCK,  FUNC_MIC0_SCK);

    /* speak dac */
    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 system interrupts, enable global interrupts, and initialize dmac.

```
/* Initialize interrupt, enable global interrupt, and initialize DMAC*/
plic_init();
sysctl_enable_irq();
dmac_init();
```

4.4 Initialize the I2S device corresponding to the speaker and set the sampling rate to 16000.

```
void init_speaker(void)
{
    /* Initialize I2S, and the third parameter is to set the channel mask, channel 0:
    i2s_init(I2S_DEVICE_2, I2S_TRANSMITTER, 0x03);

    /* Set the channel parameters for I2S to send data*/
    i2s_tx_channel_config(
        I2S_DEVICE_2, /* I2S device number*/
        I2S_CHANNEL_0, /* I2S channle */
        RESOLUTION_16_BIT, /* Number of data received */
        SCLK_CYCLES_32, /* Number of individual data clocks*/
        TRIGGER_LEVEL_4, /* FIFO depth when DMA is triggered */
        RIGHT_JUSTIFYING_MODE); /* Working mode */
    /* Set sampling rate */
    i2s_set_sample_rate(I2S_DEVICE_2, 16000);
}
```

4.5 Initialize the microphone configuration. The microphone uses channel 1 of the I2S0 device, set the sampling rate to 16000. Then, set the DMA interrupt callback function to i2s_receive_dma_cb, and finally start DMA transfer. When DMA data transfer is ending, it will trigger the DM interrupt.

```
void init_mic(void)
{
    /* I2S device 0 is initialized to receive mode */
    i2s_init(I2S_DEVICE_0, I2S_RECEIVER, 0x0C);

    /* set channel */
    i2s_rx_channel_config(
        I2S_DEVICE_0, /* I2S device 0 */
        I2S_CHANNEL_1, /* channel1 */
        RESOLUTION_16_BIT, /* receive data 16bit */
        SCLK_CYCLES_32, /* single data clock is 32 */
        TRIGGER_LEVEL_4, /* FIFO depth is 4 */
        STANDARD_MODE); /* standard mode */

    /* set sample rate */
    i2s_set_sample_rate(I2S_DEVICE_0, 16000);

    /* Set the DMA interrupt callback*/
    dmac_set_irq(DMAC_CHANNEL1, i2s_receive_dma_cb, NULL, 4);

    /* I2S receives data through DMA and saves it to RX_BUF*/
    i2s_receive_data_dma(I2S_DEVICE_0, &rx_buf[g_index], FRAME_LEN * 2, DMAC_CHANNEL1);
}
```

4.6 Receive the original data of the microphone in the DMA interrupt function and save it in g_rx_dma_buf. We only take the data of the right channel and save it in rx_buf, where MIC_GAIN is the microphone gain.

```
/* Microphone gain value, you can increa
#define MIC_GAIN      1
```

```c
int i2s_receive_dma_cb(void *ctx)
{
    uint32_t i;

    if(g_index)
    {
        /* receive DMA data */
        i2s_receive_data_dma(I2S_DEVICE_0, &g_rx_dma_buf[g_index], FRAME_LEN * 2, DMAC_CHANNEL1);
        g_index = 0;
        for(i = 0; i < FRAME_LEN; i++)
        {
            /* save data */
            rx_buf[2 * i] = (int16_t)((g_rx_dma_buf[2 * i + 1] * MIC_GAIN) & 0xffff);
            rx_buf[2 * i + 1] = (int16_t)((g_rx_dma_buf[2 * i + 1] * MIC_GAIN) & 0xffff);
        }
        i2s_rec_flag = 1;
    }
    else
    {
        i2s_receive_data_dma(I2S_DEVICE_0, &g_rx_dma_buf[0], FRAME_LEN * 2, DMAC_CHANNEL1);
        g_index = FRAME_LEN * 2;
        for(i = FRAME_LEN; i < FRAME_LEN * 2; i++)
        {
            rx_buf[2 * i] = (int16_t)((g_rx_dma_buf[2 * i + 1] * MIC_GAIN) & 0xffff);
            rx_buf[2 * i + 1] = (int16_t)((g_rx_dma_buf[2 * i + 1] * MIC_GAIN) & 0xffff);
        }
        i2s_rec_flag = 2;
    }
    return 0;
}
```

4.7 Compile and debug, burn and run

Copy the mic to the src directory in the SDK.

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

cmake .. -DPROJ=mic -G "MinGW Makefiles"

make

```
[100%] Linking C executable mic
Generating .bin file ...
[100%] Built target mic
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **mic.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 **mic.bin** file to the K210 development board.

5. Experimental phenomenon

The microphone will collect the sound of the current environment and play it through the speaker.

!Note: If the gain of the microphone is set too large, it will cause the microphone and the speaker to resonate, resulting in a lot of noise.

6. Experiment summary

6.1 Both the microphone and the speaker use I2S to transmit data. The microphone uses the input mode, and the speaker uses the output mode.

6.2 The data buffered by the microphone is directly transmitted to the DAC component connected to the speaker through the DMA channel.

6.3 The microphone is a sound-sensitive component, it can directly convert sound into electrical energy signals.

Appendix -- API

i2s_init

Description: Initialize 2S

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 receive 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 bits	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 bits	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	Single sending quantity	Input
bits_per_sample	Single sampling bit width	Input
track_num	Number of channels	Input

Return value: No

i2s_set_sample_rate
Description: Set sample 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	Sample rate	Input

Return value: 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: Get the dma_divide_16 value, which be used to determine whether to set dma_divide_16.

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 transmission mode.
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	Send mode
I2S_RECEIVER	Receive mode

i2s_work_mode_t

Description: I2S work 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 transmission 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 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	DMA channel number used when sending
rx_channel	DMA channel number used when receiving
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