3.11 Touch screen read coordinate data

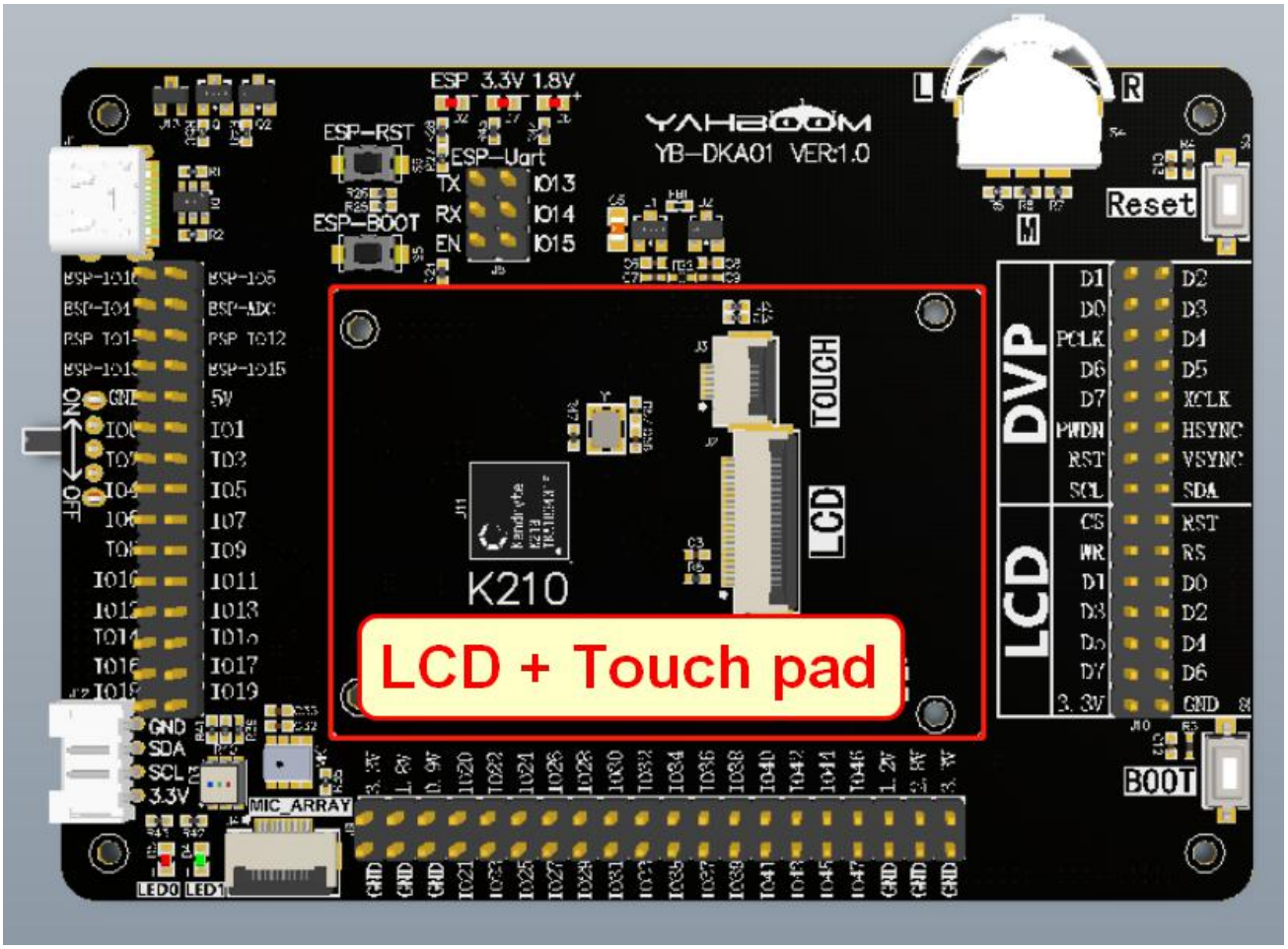
1. Experiment purpose

In this lesson, we mainly learns how to read the coordinates of the touch screen through I2C, print them out, and display them on the LCD.

2.Experiment preparation

2.1 components

LCD + Touch pad



2.2 Component characteristics

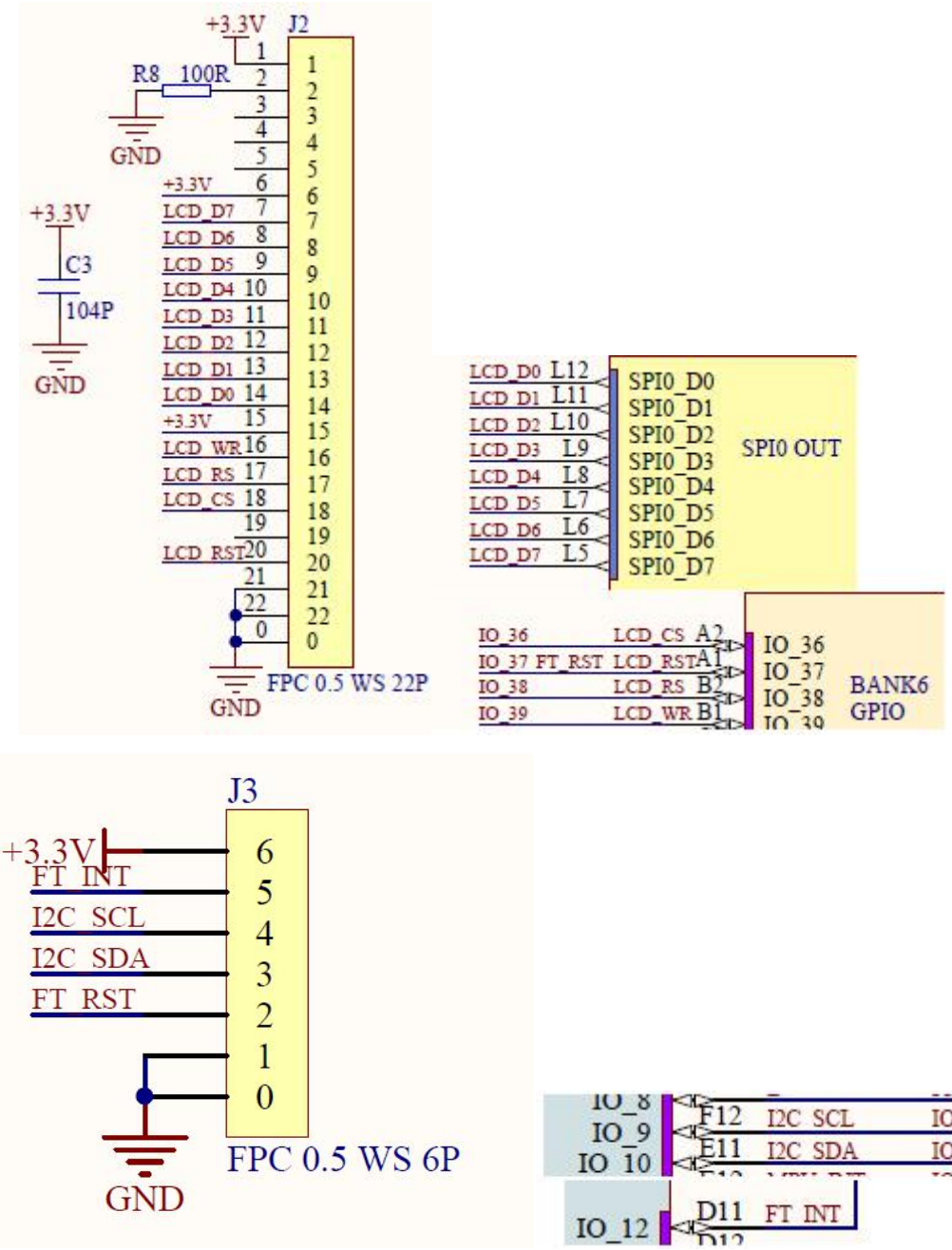
The K210 development board comes with a 2.0-inch capacitive touch screen.

The touch panel uses I2C communication, which can transmit data stably. It supports parallel communication with other I2C devices to the same I2C interface.

2.3 Hardware connection

The K210 development board is shipped with a LCD installed by default.

A total of eight pins of LCD_D0~D7 are connected to SPI0_D0~D7, LCD_CS is connected to IO36, LCD_RST is connected to IO37, LCD_RS is connected to IO3, and LCD_WR is connected to IO39.



2.4 SDK API function

The header file is i2c.h

The I2C bus is used to communicate with multiple external devices. Multiple external devices can share an I2C bus.

The K210 chip integrated circuit bus has 3 I2C bus interfaces, they can be used as I2C master (MASTER) mode or slave (SLAVE) mode.

The I2C interface supports standard mode (0 to 100kb/s), fast mode (<=400kb/s), 7-bit or 10-bit addressing mode, bulk transfer mode, interrupt or polling mode operation.

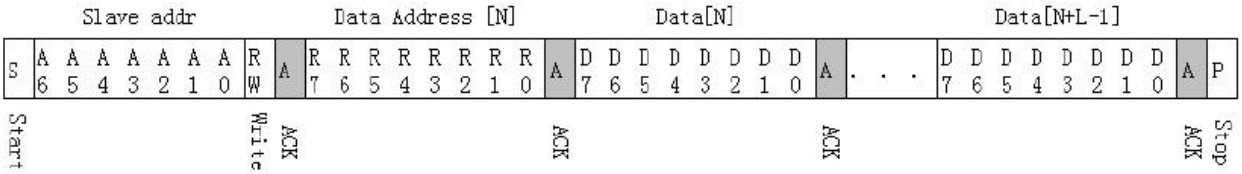
We will provide following interfaces to users.

• i2c_init: Initialize I2C, configure the slave address, register bit width and I2C rate.
• i2c_init_as_slave: Configure I²C as slave mode.
• i2c_send_data: I2C write data.
• i2c_send_data_dma: I2C writes data through DMA.
• i2c_recv_data: I2C reads data through the CPU.
• i2c_recv_data_dma: I2C reads data through the dma.
• i2c_handle_data_dma: I2C uses dma to transmit data.

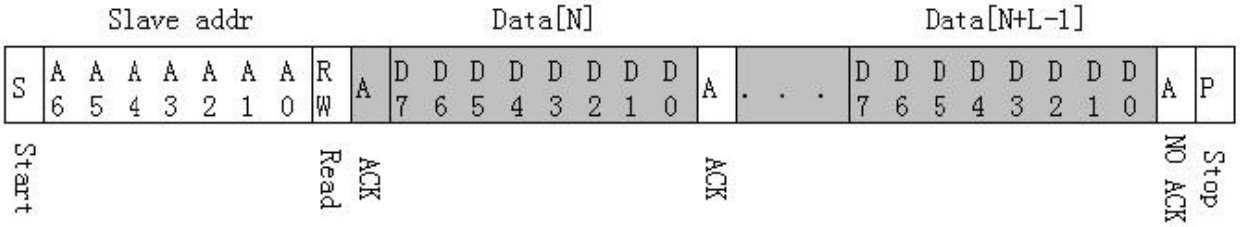
3. Experimental principle

Capacitive touch screen technology uses the current induction of the human body to work.
I2C is a bus structure, it only needs the SCL clock signal line and the SDA data line.
Before communication, a staring signal must be sent by the master device to determine whether the data can be transferred. When the communication ends, a ending signal must be sent by the master device, which indicate that the communication has ended.

The process of I2C writing data to the register:



The process of I2C reading data from the register:



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_LCD_CS              (36)
#define PIN_LCD_RST             (37)
#define PIN_LCD_RS              (38)
#define PIN_LCD_WR              (39)

#define PIN_FT_RST              (37)
#define PIN_FT_INT              (12)
#define PIN_FT_SCL              (9)
#define PIN_FT_SDA              (10)

/*****************************SOFTWARE-GPIO********************************/
// Software GPIO port, corresponding program
#define LCD_RST_GPIONUM         (0)
#define LCD_RS_GPIONUM          (1)

#define FT_INT_GPIONUM          (2)
#define FT_RST_GPIONUM          (3)

/*****************************FUNC-GPIO************************************/
// Function of GPIO port, bound to hardware IO port
#define FUNC_LCD_CS             (FUNC_SPI0_SS3)
#define FUNC_LCD_RST            (FUNC_GPIOHS0 + LCD_RST_GPIONUM)
#define FUNC_LCD_RS             (FUNC_GPIOHS0 + LCD_RS_GPIONUM)
#define FUNC_LCD_WR             (FUNC_SPI0_SCLK)

#define FUNC_FT_RST             (FUNC_GPIOHS0 + FT_RST_GPIONUM)
#define FUNC_FT_INT             (FUNC_GPIOHS0 + FT_INT_GPIONUM)
#define FUNC_FT_SCL             (FUNC_I2C0_SCLK)
#define FUNC_FT_SDA             (FUNC_I2C0_SDA)
```

4.2 Set the IO port level voltage of the LCD to 1.8V.

```
void io_set_power(void)
{
    sysctl_set_power_mode(SYSCTL_POWER_BANK6, SYSCTL_POWER_V18);
}
```

4.3 Since the touchpad needs to use interrupts to determine whether the screen is touched, all need to initialize interrupts and enable global interrupts.

```
/* System interrupt initialization, and enable global interrupt */
plic_init();
sysctl_enable_irq();
```

4.4 Initialize the LCD display, and display the picture and the string.

```
/* Initialize LCD*/
lcd_init();

/* Display image */
uint16_t * img = &gImage_logo;
lcd_draw_picture_half(0, 0, 320, 240, img);
sleep(1);
lcd_draw_string(16, 40, "Hello Yahboom", RED);
lcd_draw_string(16, 60, "Nice to meet you!", BLUE);
```

4.5 Initialize the touch panel, and print touch prompts through LCD display and serial port.

```
/* Initialize Touchpad */
ft6236_init();
printf("Hi!Please touch the screen to get coordinates!\n");
lcd_draw_string(16, 180, "Please touch the screen to get coord!", RED);
```

The initialization of the touch panel FT6236 is divided into hardware initialization and software initialization. The software initialization is mainly to set the registers of the FT6236 and wake up the FT6236 to set the touch sensitivity and scan period.

```
/* ft6236 initialization*/
void ft6236_init(void)
{
    ft6236.touch_state = 0;
    ft6236.touch_x = 0;
    ft6236.touch_y = 0;

    /* Hardware initialization */
    ft6236_hardware_init();

    /* Software initializatio */
    i2c_hardware_init(FT6236_I2C_ADDR);
    ft_i2c_write(FT_DEVIDE_MODE, 0x00);

    ft_i2c_write(FT_ID_G_THGROUP, 0x12);    // 0x22

    ft_i2c_write(FT_ID_G_PERIODACTIVE, 0x06);
}
```

4.6 Hardware pin initialization.

```c
/* FT6236 Hardware pin initialization */
void ft6236_hardware_init(void)
{
    /* Set to 1 when using a reset pin different from the screen*/
    #if (0)
    {
        gpiohs_set_drive_mode(FT_RST_GOIONUM, GPIO_DM_OUTPUT);
        ft6236_reset_pin(LEVEL_LOW);
        msleep(50);
        ft6236_reset_pin(LEVEL_HIGH);
        msleep(120);
    }
    #endif

    gpiohs_set_drive_mode(FT_INT_GPIONUM, GPIO_DM_INPUT);
    gpiohs_set_pin_edge(FT_INT_GPIONUM, GPIO_PE_RISING);
    gpiohs_irq_register(FT_INT_GPIONUM, FT6236_IRQ_LEVEL, ft6236_isr_cb, NULL);
    msleep(5);

}
```

```c
/* Interrupt the callback function, modify the state of touch_state to touch */
void ft6236_isr_cb(void)
{
    ft6236.touch_state |= TP_COORD_UD;
}
```

4.7 FT6236 reads and writes data through I2C communication. The following is the function of touch panel I2C control.

```c
/* I2C write data */
static void ft_i2c_write(uint8_t reg, uint8_t data)
{
    i2c_hd_write(FT6236_I2C_ADDR, reg, data);
}

/* I2C read data  */
static void ft_i2c_read(uint8_t reg, uint8_t *data_buf, uint16_t length)
{
    i2c_hd_read(FT6236_I2C_ADDR, reg, data_buf, length);
}
```

Initialize I2C, set slave address, data bit width, I2C communication rate, etc.,

```
static uint16_t _current_addr = 0x00;

/* Hardware initialization I2C, set slave address, data bit width, I2C communication rate*/
void i2c_hardware_init(uint16_t addr)
{
    i2c_init(I2C_DEVICE_0, addr, ADDRESS_WIDTH, I2C_CLK_SPEED);
    _current_addr = addr;
}
```

Write data to register reg. When it write successful, it will return 0; When it can't write, it will return non-0.

```
/* Write data to register reg, return 0 on write, return non-0 on failure*/
uint16_t i2c_hd_write(uint8_t addr, uint8_t reg, uint8_t data)
{
    if (_current_addr != addr)
    {
        i2c_hardware_init(addr);
    }
    uint8_t cmd[2];
    cmd[0] = reg;
    cmd[1] = data;
    uint16_t error = 1;
    error = i2c_send_data(I2C_DEVICE_0, cmd, 2);
    return error;
}
```

Read length data from register reg and save it to data_buf. When it write successful, it will return 0; When it can't write, it will return non-0.

```
/* Read length data from register reg and save it to data_buf, return 0 if read successfully, non-zero if failed */
uint16_t i2c_hd_read(uint8_t addr, uint8_t reg, uint8_t *data_buf, uint16_t length)
{
    if (_current_addr != addr)
    {
        i2c_hardware_init(addr);
    }
    uint16_t error = 1;
    error = i2c_recv_data(I2C_DEVICE_0, &reg, 1, data_buf, length);
    return error;
}
```

4.8 The last in the main function is a while(1) loop, which reads the coordinate XY value of the touch screen, then prints it out through the serial port, and displays it on the LCD display.
We need to refresh the displayed position every time the display is completed, otherwise the old data will overlap together, so the lcd_clear_coord() function is to clear the coordinates.

```
    while (1)
    {
        /* Refresh the data bit, clear the last displayed data*/
        if (is_refresh)
        {
            lcd_clear_coord();
            is_refresh = 0;
        }

        /* If you touch the touch screen */
        if (ft6236.touch_state & TP_COORD_UD)
        {
            ft6236.touch_state &= ~TP_COORD_UD;
            /* Scan touch screen */
            ft6236_scan();
            /* Serial print X Y coordinates */
            printf("X=%d, Y=%d \n ", ft6236.touch_x, ft6236.touch_y);
            sprintf(coord, "(%d, %d)", ft6236.touch_x, ft6236.touch_y);

            lcd_draw_string(120, 200, coord, BLUE);
            is_refresh = 1;
        }
        /* Delay 80 milliseconds to ensure normal refresh of screen data */
        msleep(80);
    }

    return 0;
}
```

4.9 First, use the lcd_set_area function to set the area of the display coordinates.
Then, write the white value (0XFFFFFFFF) to refresh the displayed content.

```
void lcd_clear_coord(void)
{
    uint32_t color = 0xFFFFFFFF;
    uint8_t x1 = 120;
    uint8_t y1 = 200;
    uint8_t width = 100;
    uint8_t height = 16;

    lcd_set_area(x1, y1, x1 + width - 1, y1 + height - 1);
    tft_fill_data(&color, width * height / 2);
}
```

4.10 Compile and debug, burn and run
Copy the touch to the src directory in the SDK.
Then, enter the build directory and run the following command to compile.

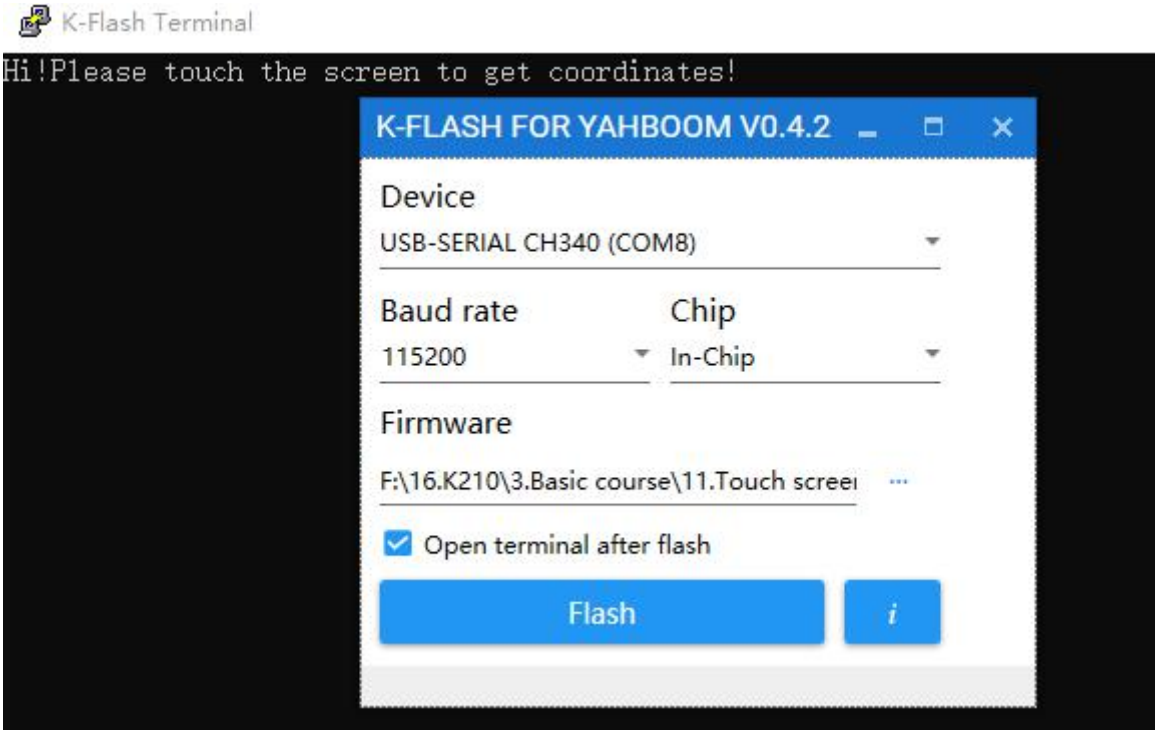
cmake .. -DPROJ=touch -G "MinGW Makefiles"
make

```
[ 89%] Linking C executable touch
Generating .bin file ...
[100%] Built target touch
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

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

5. Experimental phenomenon

After the firmware is write, a terminal interface will pop up. As shown below.



When we touch the screen, the coordinates of the current touch will also be displayed on the LCD screen. At the same time, the terminal will print out the current coordinate value. When you let go, the coordinates will disappear.
Every time you move your finger and change the touch position, the coordinates of the corresponding touch point will also changed.

K-Flash Terminal

```
Hi!Please touch the screen to get coordinates!
X=0,  Y=0
 X=101,  Y=133
 X=101,  Y=133
 X=101,  Y=133
 X=229,  Y=132
 X=229,  Y=132
 X=229,  Y=132
 X=142,  Y=214
 X=142,  Y=214
 X=142,  Y=214
 X=116,  Y=191
 X=116,  Y=191
 X=116,  Y=191
 X=239,  Y=85
 X=239,  Y=85
 X=239,  Y=85
 X=122,  Y=136
 X=122,  Y=136
 X=122,  Y=136
 X=295,  Y=38
 X=295,  Y=38
 X=192,  Y=225
 X=192,  Y=225
 X=63,  Y=165
 X=63,  Y=165
```

6. Experiment summary

6.1 The touch screen of the K210 development board is a capacitive touch screen, which can be directly touched with a finger.

6.2 The touch pad uses I2C communication.

6.3 The relevant registers of the touch panel are placed in the README.md file of the project. About more detail, please view[8.About Hardware]--[LCD].

Appendix -- API

Header file is i2c.h

i2c_init

Description: Configure the I²C device slave address, register bit width, and I²C rate.

Function prototype: **void i2c_init(i2c_device_number_t i2c_num, uint32_t slave_address, uint32_t address_width, uint32_t i2c_clk)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C number	Input
slave_address	I²C device slave address	Input
address_width	I²C register bit width(7 or 10)	Input
i2c_clk	I²C rate (Hz)	Input

Return value: No

i2c_init_as_slave

Description: Configure the I²C device slave mode

Function prototype: **void i2c_init_as_slave(i2c_device_number_t i2c_num, uint32_t slave_address, uint32_t address_width, const i2c_slave_handler_t *handler)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C number	Input
slave_address	I²C slave mode address	Input
address_width	I²C register bit width(7 or 10)	Input
handler	I²C slave mode interrupt handling function	Input

Return value: No

i2c_send_data

Description: Write data

Function prototype: **int i2c_send_data(i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C number	Input
send_buf	Data waiting to be transferred	Input
send_buf_len	Length of data waiting to be transferred	Input

Return value:

Return value	Description
0	succeed
!0	Failure

i2c_send_data_dma

Description: Write data by DMA

Function prototype: **void i2c_send_data_dma(dmac_channel_number_t dma_channel_num, i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len)**

Parameter:

Parameter name	Description	Input/Output
dma_channel_num	dma channel number	Input
i2c_num	I²C number	Input
send_buf	Data waiting to be transferred	Input
send_buf_len	Length of data waiting to be transferred	Input

Return value: No

i2c_recv_data

Description: Read data by CPU

Function prototype: **int i2c_recv_data(i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len, uint8_t *receive_buf, size_t receive_buf_len)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C bus number	Input
send_buf	The data waiting to be transmitted is generally the register of the i2c peripheral, if it is not set to NULL	Input
send_buf_len	Length of the data waiting to be transmitted, if not, write 0	Input
receive_buf	Receive data memory	Output
receive_buf_len	Length of receive data	Output

Return value:

Return value	Description
0	succeed
!0	Failure

i2c_recv_data_dma

Description: Read data bu dma

Function prototype: **void i2c_recv_data_dma(dmac_channel_number_t dma_send_channel_num, dmac_channel_number_t dma_receive_channel_num, i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len, uint8_t *receive_buf, size_t receive_buf_len)**

Parameter:

Parameter name	Description	Input/Output
dma_send_channel_num	DMA channel used to send data	Input
dma_receive_channel_num	DMA channel used to receive data	Input
i2c_num	I²C bus number	Input
send_buf	The data to be transmitted is generally the register of the i2c peripheral, if it is not set to NULL	Input
send_buf_len	The length of the data to be transmitted, if not, write 0	Input
receive_buf	Receive data memory	Output
receive_buf_len	Length of received data	Input

Return value: no

i2c_handle_data_dma

Description: I2C uses dma to transmit data.

Function prototype: **void i2c_handle_data_dma(i2c_device_number_t i2c_num, i2c_data_t data, plic_interrupt_t *cb);**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C bus number	Input
data	I2C 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:

```
/* The i2c peripheral address is 0x32, 7-bit address, and the rate is 200K */
i2c_init(I2C_DEVICE_0, 0x32, 7, 200000);
uint8_t reg = 0;
uint8_t data_buf[2] = {0x00,0x01}
data_buf[0] = reg;

/* Write 0x01 to register 0*/
i2c_send_data(I2C_DEVICE_0, data_buf, 2);
i2c_send_data_dma(DMAC_CHANNEL0, I2C_DEVICE_0, data_buf, 4);

/* Read 1 byte of data from register 0*/
i2c_receive_data(I2C_DEVICE_0, &reg, 1, data_buf, 1);
i2c_receive_data_dma(DMAC_CHANNEL0, DMAC_CHANNEL1, I2C_DEVICE_0,&reg, 1, data_buf, 1);
```

Data type

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

- i2c_device_number_t: i2c number.
- i2c_slave_handler_t: interrupt handler handle of i2c slave mode.
- i2c_data_t: data-related parameters when using dma transmission.
- i2c_transfer_mode_t: The mode of using DMA to transfer data, sending or receiving.

i2c_device_number_t

Description: i2c number.

Define

```
typedef enum _i2c_device_number
{
    I2C_DEVICE_0,
    I2C_DEVICE_1,
    I2C_DEVICE_2,
    I2C_DEVICE_MAX,
} i2c_device_number_t;
```

i2c_slave_handler_t

Description: i2c Slave mode interrupt handler function. According to different interrupt states, perform corresponding function operations

Define

```
typedef struct _i2c_slave_handler
{
    void(*on_receive)(uint32_t data);
    uint32_t(*on_transmit)();
    void(*on_event)(i2c_event_t event);
} i2c_slave_handler_t;
```

member

Member name	Description
I2C_DEVICE_0	I2C 0
I2C_DEVICE_1	I2C 1
I2C_DEVICE_2	I2C 2

i2c_data_t

Description: data-related parameters when using dma transmission.

Define

```
typedef struct _i2c_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;
    i2c_transfer_mode_t transfer_mode;
} i2c_data_t;
```

member

Member name	Description
tx_channel	DMA channel number used when sending
rx_channel	DMA channel number used when receiving
tx_buf	Sent data
tx_len	Length of sent data
rx_buf	Received Data
rx_len	Received Data of received Data
transfer_mode	Transfer_mode, sent or received

i2c_transfer_mode_t

Description: The mode of using DMA to transfer data, send or receive.

Define

```
typedef enum _i2c_transfer_mode
{
    I2C_SEND,
    I2C_RECEIVE,
} i2c_transfer_mode_t;
```

member

Member name	Description
I2C_SEND	send
I2C_RECEIVE	receive