

3. 11 触摸屏读取坐标数据

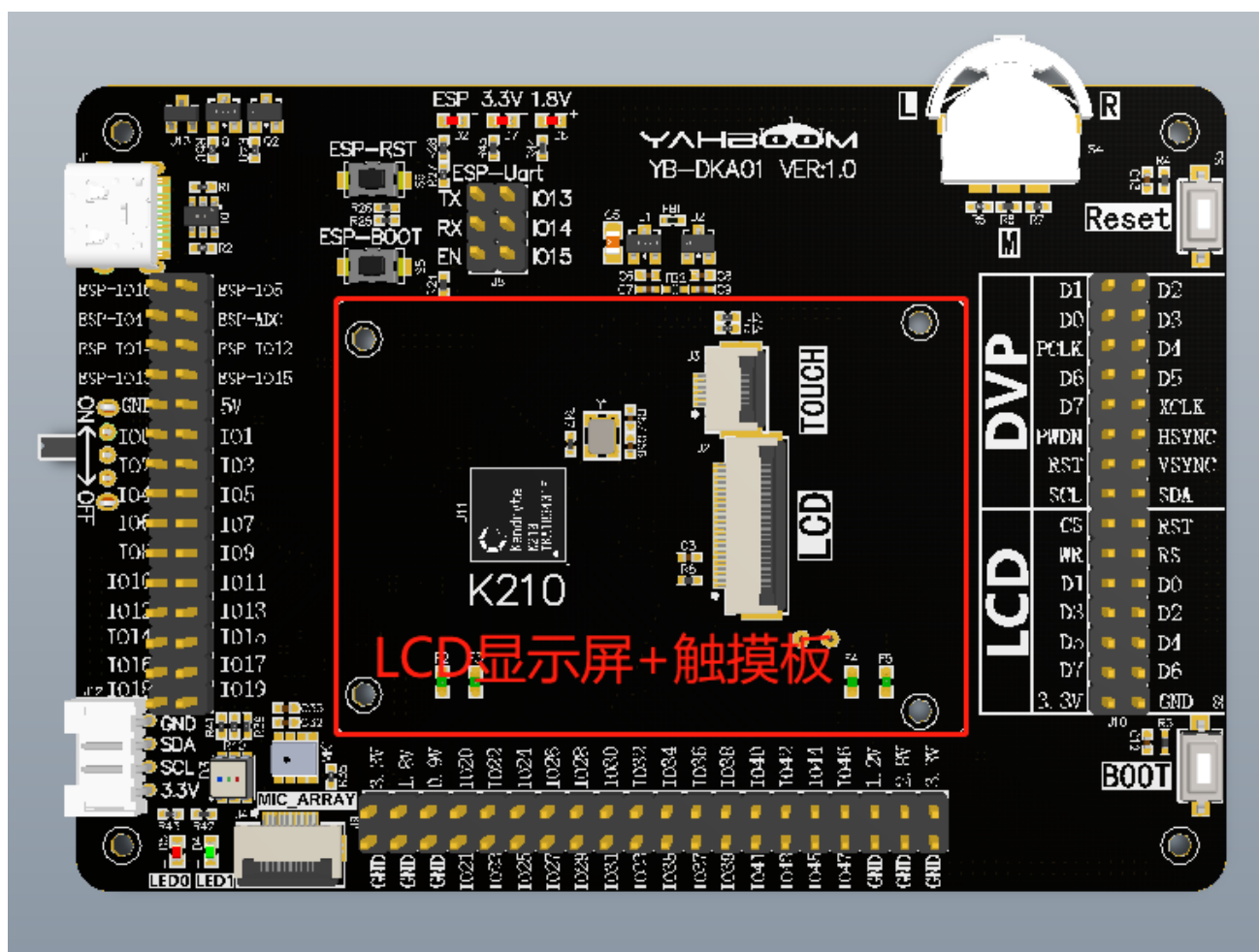
一、实验目的

本节课主要学习 K210 通过 I2C 读取触摸屏的坐标，并打印出来，显示在 LCD 上。

二、实验准备

1. 实验元件

LCD 显示屏+触摸板



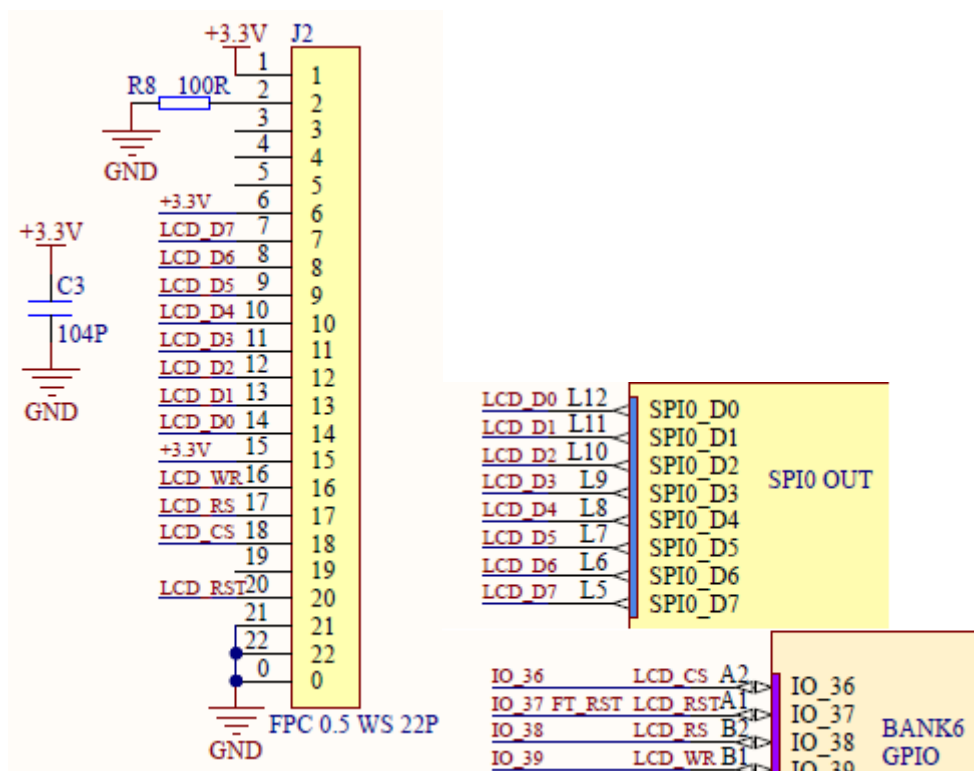
2. 元件特性

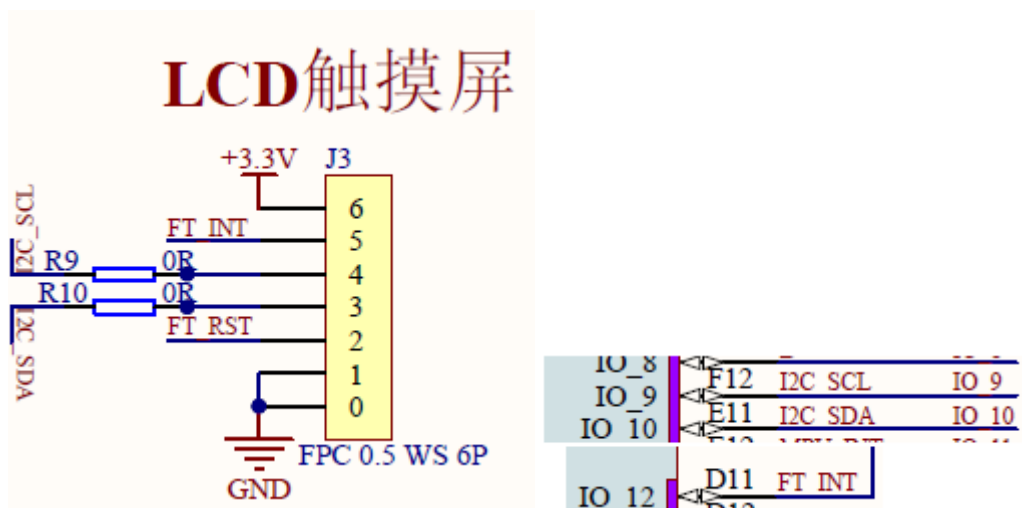
K210 开发板自带 2.0 寸触摸屏，其实是 LCD 显示屏上贴一个触摸板组成，LCD

显示屏上一节课已经学过，而触摸板为电容触摸板，与手机上使用的电容屏原理一样，可以直接用手指触摸，而不像电阻屏一样要用指甲按。电容屏具有相应时间短，精确度高，使用寿命长，操作简单方便等特点。触摸板使用的是 I2C 通讯，能够稳定传输数据，并且支持与其他 I2C 设备并联到同一个 I2C 接口上通讯。

3. 硬件连接

K210 开发板出厂默认已经安装好 LCD 显示屏和触摸板，其中 LCD 显示屏的 LCD_D0~D7 总共八个引脚连接到 SPI0_D0~D7 上，LCD_CS 连接到 IO36 上，LCD_RST 连接到 IO37 上，LCD_RS 连接 IO38 上，LCD_WR 连接 IO39 上；触摸板的 I2C_SCL 连接到 IO9，I2C_SDA 连接到 IO10，FT_INT 连接到 IO12，FT_RST 连接到 IO37，与 LCD 显示屏共用使用一个 RST 接口。





4. SDK 中对应 API 功能

对应的头文件 `i2c.h`

I2C 总线用于和多个外部设备进行通信。多个外部设备可以共用一个 I2C 总线。

I2C 模块具有独立的 I2C 设备封装外设相关参数，自动处理多设备总线争用的功能。

K210 芯片集成电路总线有 3 个 I2C 总线接口，都可以作为 I2C 主机(MASTER)模式或从机 (SLAVE) 模式来使用。

I2C 接口支持标准模式 (0 到 100kb/s)，快速模式 ($\leq 400\text{kb/s}$)，7 位或 10 位寻址模式，批量传输模式，中断或轮询模式操作。

为用户提供以下接口：

- `i2c_init`: 初始化 I2C，配置从机地址、寄存器位宽度和 I2C 速率。
- `i2c_init_as_slave`: 配置 I²C 为从模式。
- `i2c_send_data`: I2C 写数据。
- `i2c_send_data_dma`: I2C 通过 DMA 写数据。
- `i2c_recv_data`: I2C 通过 CPU 读数据。
- `i2c_recv_data_dma`: I2C 通过 dma 读数据。

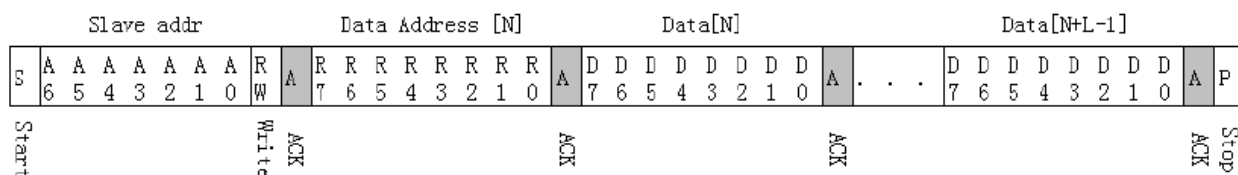
- i2c_handle_data_dma: I2C 使用 dma 传输数据。

三、实验原理

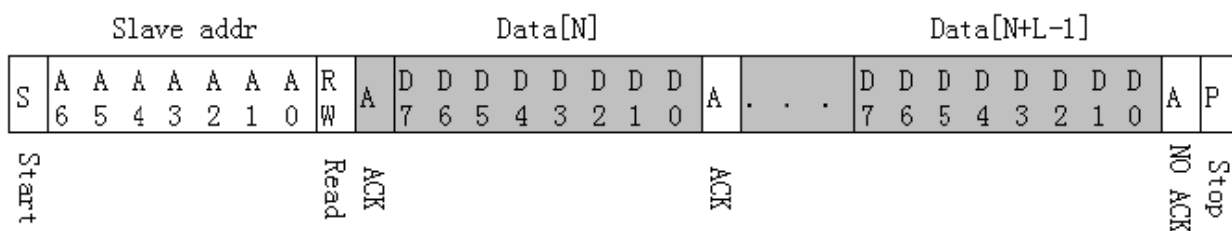
电容式触摸屏技术是利用人体的电流感应进行工作的。电容式触摸屏是一块四层复合玻璃屏，玻璃屏的内表面和夹层各涂有一层 ITO，最外层是一薄层矽土玻璃保护层，夹层 ITO 涂层作为工作面，四个角上引出四个电极，内层 ITO 为屏蔽层以保证良好的工作环境。当手指触摸在金属层上时，由于人体电场，用户和触摸屏表面形成以一个耦合电容，对于高频电流来说，电容是直接导体，于是手指从接触点吸走一个很小的电流。这个电流分别从触摸屏的四角上的电极中流出，并且流经这四个电极的电流与手指到四角的距离成正比，控制器通过对这四个电流比例的精确计算，得出触摸点的位置。

I2C 是一种总线式结构，它只需要 SCL 时钟信号线与 SDA 数据线，两根线就能将连接与总线上的设备实现数据通信，由于它的简便的构造设计，于是成为一种较为常用的通信方式。由于 I2C 采用的是主从式通信方式，所以，通信的过程完全由主设备仲裁。在通信之前，必须由主设备发送一个起始信号，决定数据是否可以开始传送，并且在结束通信时，必须再由主设备发送一个结束信号，以表示通信已经结束。

I2C 向寄存器写入数据的过程：



I2C 从寄存器读取数据的过程：



四、实验过程

1. 首先初始化 K210 的硬件引脚和软件功能使用的是 FPIOA 映射关系。

```

/*****HARDWARE-PIN*****/
// 硬件IO口，与原理图对应
#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*****/
// 软件GPIO口，与程序对应
#define LCD_RST_GPIONUM      (0)
#define LCD_RS_GPIONUM      (1)

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

/*****FUNC-GPIO*****/
// GPIO口的功能，绑定到硬件IO口
#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)

```

```

void hardware_init(void)
{
    /**
     *lcd_cs      36
     *lcd_rst     37
     *lcd_rs      38
     *lcd_wr      39
     */
    fpioa_set_function(PIN_LCD_CS, FUNC_LCD_CS);
    fpioa_set_function(PIN_LCD_RST, FUNC_LCD_RST);
    fpioa_set_function(PIN_LCD_RS, FUNC_LCD_RS);
    fpioa_set_function(PIN_LCD_WR, FUNC_LCD_WR);

    /* 使能SPI0和DVP */
    sysctl_set_spi0_dvp_data(1);

    /* I2C FT6236 */
    // fpioa_set_function(PIN_FT_RST, FUNC_FT_RST);
    fpioa_set_function(PIN_FT_INT, FUNC_FT_INT);
    fpioa_set_function(PIN_FT_SCL, FUNC_FT_SCL);
    fpioa_set_function(PIN_FT_SDA, FUNC_FT_SDA);
}

```

2. 设置 LCD 的 IO 口电平电压为 1.8V。

```

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

```

3. 由于触摸板需要使用中断来判断屏幕是否有触摸，所有需要初始化中断并使能全局中断。

```

/* 系统中断初始化，并使能全局中断 */
plic_init();
sysctl_enable_irq();

```

4. 初始化 LCD 显示屏，并显示图片和字符串欢迎语。

```

/* 初始化LCD */
lcd_init();

/* 显示图片 */
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);

```

5. 初始化触摸板，并通过 LCD 显示和串口打印触摸提示。

```

/* 初始化触摸板 */
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);

```

触摸板 FT6236 的初始化比较简单，分为硬件初始化和软件初始化，软件初始化主要是设置 FT6236 的寄存器，唤醒 FT6236 设置触摸灵敏度和扫描周期。

```

/* 初始化ft6236 */
void ft6236_init(void)
{
    ft6236.touch_state = 0;
    ft6236.touch_x = 0;
    ft6236.touch_y = 0;

    /* 硬件初始化 */
    ft6236_hardware_init();

    /* 软件初始化 */
    i2c_hardware_init(FT6236_I2C_ADDR);
    ft_i2c_write(FT_DEVIDE_MODE, 0x00);
    /* 设置触摸有效值，越小越灵敏，def=0xbb */
    ft_i2c_write(FT_ID_G_THGROUP, 0x12); // 0x22
    /* 工作扫描周期，用于控制报点率，def=0x08, 0x04~0x14 */
    ft_i2c_write(FT_ID_G_PERIODACTIVE, 0x06);
}

```

6. 硬件引脚初始化，主要是修改触摸板复位引脚的电平，这里与屏幕使用同一个复位引脚，所以不需要再复位操作。再设置 FT_INT 中断引脚为输入，中断回调函数为 ft6236_isr_cb，并在中断函数中修改触摸屏的状态。

```

/* FT6236硬件引脚初始化 */
void ft6236_hardware_init(void)
{
    /* 与屏幕使用不同复位引脚时设置为1 */
    #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);
}

```

```

/* 中断回调函数，修改touch_state的状态为有触摸 */
void ft6236_isr_cb(void)
{
    ft6236.touch_state |= TP_COORD_UD;
}

```

7. FT6236通过 I2C 通讯的方式来读和写数据，以下是触摸板 I2C 控制的函数：

```

/* I2C写数据 */
static void ft_i2c_write(uint8_t reg, uint8_t data)
{
    i2c_hd_write(FT6236_I2C_ADDR, reg, data);
}

/* I2C读数据 */
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);
}

```

初始化 I2C，设置从机地址，数据位宽度，I2C 通讯速率等，


```
static uint16_t _current_addr = 0x00;

/* 硬件初始化I2C, 设置从机地址, 数据位宽度, I2C通讯速率 */
void i2c_hardware_init(uint16_t addr)
{
    i2c_init(I2C_DEVICE_0, addr, ADDRESS_WIDTH, I2C_CLK_SPEED);
    _current_addr = addr;
}
```

向寄存器 reg 写入一个数据 data , 写入成功返回 0, 失败则返回非 0,

```
/* 向寄存器reg写入一个数据data , 写入成功返回0, 失败则返回非0*/
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;
}
```

从寄存器 reg 读取 length 个数据保存到 data_buf, 读取成功返回 0, 失败则返回非 0。

```
/* 从寄存器reg读取length个数据保存到data_buf, 读取成功返回0, 失败则返回非0 */
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;
}
```

8. main 函数里最后是一个 while(1) 循环，读取触摸屏的坐标 XY 值，然后通过串口打印出来，再显示到 LCD 显示屏上。每次显示完成后都需要刷新以下显示的位置，否则旧数据会重叠到一起，所以 lcd_clear_coord 函数就是清除坐标的作用。

```
while (1)
{
    /* 刷新数据位，清空上次显示的数据 */
    if (is_refresh)
    {
        lcd_clear_coord();
        is_refresh = 0;
    }

    /* 如果触摸到触摸屏的时候 */
    if (ft6236.touch_state & TP_COORD_UD)
    {
        ft6236.touch_state &= ~TP_COORD_UD;
        /* 扫描触摸屏 */
        ft6236_scan();
        /* 串口打印X Y 坐标 */
        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;
    }
    /* 延迟80毫秒保证屏幕数据正常刷新 */
    msleep(80);
}
```

9. 先使用 lcd_set_area 函数设定显示坐标的区域，然后再写入白色的值 (0xFFFFFFFF)，以此来刷新显示的内容。如果背景图片的坐标区域不是空白的，则无法使用此方法来刷新。

```
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);
}
```

10. 编译调试，烧录运行

把本课程资料中的 touch 复制到 SDK 中的 src 目录下,然后进入 build 目录,运行以下命令编译。

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

```
make
```

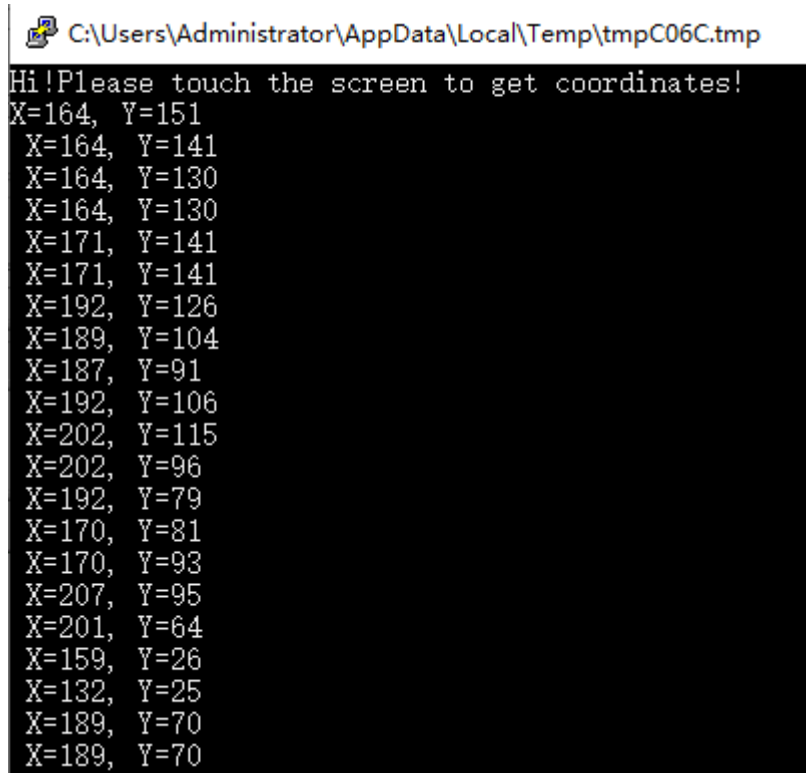
```
[ 89%] Linking C executable touch
Generating .bin file ...
[100%] Built target touch
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build> 
```

编译完成后，在 build 文件夹下会生成 touch.bin 文件。

使用 type-C 数据线连接电脑与 K210 开发板,打开 kflash,选择对应的设备,再将程序固件烧录到 K210 开发板上。

五、实验现象

烧录完成固件后，系统会弹出一个终端界面，如果没有弹出终端界面的可以打开串口助手显示调试内容。



A screenshot of a serial terminal window. The title bar shows the file path: C:\Users\Administrator\AppData\Local\Temp\tmpC06C.tmp. The terminal text is as follows:

```
Hi!Please touch the screen to get coordinates!  
X=164, Y=151  
X=164, Y=141  
X=164, Y=130  
X=164, Y=130  
X=171, Y=141  
X=171, Y=141  
X=192, Y=126  
X=189, Y=104  
X=187, Y=91  
X=192, Y=106  
X=202, Y=115  
X=202, Y=96  
X=192, Y=79  
X=170, Y=81  
X=170, Y=93  
X=207, Y=95  
X=201, Y=64  
X=159, Y=26  
X=132, Y=25  
X=189, Y=70  
X=189, Y=70
```

打开电脑的串口助手，选择对应的 K210 开发板对应的串口号，波特率设置为 115200，然后点击打开串口助手。注意还需要设置一下串口助手的 DTR 和 RTS。在串口助手底部此时的 4. DTR 和 7. RTS 默认是红色的，点击 4. DTR 和 7. RTS，都设置为绿色，然后按一下 K210 开发板的复位键。



LCD 屏幕会显示图片，一秒后打印出“Hello Yahboom!” “Nice to meet you!” 的欢迎语。然后打印提示触摸语 “Please touch the screen to get coord!” 同时串口也会打印出 “Hi!Please touch the screen to get coordinates!” 的提示语。



同时 LCD 显示屏上也会显示当前触摸的坐标，当松开手时，坐标也会消失。每次移动手指，改变触摸的位置，对应的触摸点坐标也会改变。



六、实验总结

1. K210 开发板的触摸屏是电容式触摸屏，可以使用手指直接触摸使用。
2. 触摸板使用的是 I2C 通讯，读取数据后需要转化计算一下才能得到实际的触摸坐标。
3. 触摸板的相关寄存器放在项目的 README.md 文件中，具体寄存器功能请查看硬件相关资料中的触摸屏资料。

附：API 对应的头文件 i2c.h

i2c_init

描述

配置 I²C 器件从地址、寄存器位宽度和 I²C 速率。

函数原型

```
void i2c_init(i2c_device_number_t i2c_num, uint32_t slave_address, uint32_t address_width, uint32_t i2c_clk)
```

参数

参数名称	描述	输入输出
i2c_num	I ² C 号	输入
slave_address	I ² C 器件从地址	输入
address_width	I ² C 器件寄存器宽度(7 或 10)	输入
i2c_clk	I ² C 速率 (Hz)	输入

返回值

无。

i2c_init_as_slave

描述

配置 I²C 为从模式。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
i2c_num	I ² C 号	输入
slave_address	I ² C 从模式的地址	输入
address_width	I ² C 器件寄存器宽度(7 或 10)	输入
handler	I ² C 从模式的中断处理函数	输入

返回值

无。

i2c_send_data

描述

写数据。

函数原型

```
int i2c_send_data(i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t
send_buf_len)
```

参数

参数名称	描述	输入输出
i2c_num	I ² C 号	输入
send_buf	待传输数据	输入
send_buf_len	待传输数据长度	输入

返回值

返回值 描述

0 成功

非 0 失败

i2c_send_data_dma

描述

通过 DMA 写数据。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_channel_num	使用的 dma 通道号	输入
i2c_num	I ² C 号	输入
send_buf	待传输数据	输入
send_buf_len	待传输数据长度	输入

返回值

无

i2c_recv_data

描述

通过 CPU 读数据。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
i2c_num	I ² C 总线号	输入
send_buf	待传输数据，一般是 i2c 外设的寄存器，如果没有设置为 NULL	输入
send_buf_len	待传输数据长度，如果没有则写 0	输入
receive_buf	接收数据内存	输出
receive_buf_len	接收数据的长度	输入

返回值

返回值 描述

0 成功

非 0 失败

i2c_recv_data_dma**描述**

通过 dma 读数据。

函数原型

```
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)
```

参数

参数名称	描述	输 入 输 出
dma_send_channel_num	发送数据使用的 dma 通道	输入
dma_receive_channel_num	接收数据使用的 dma 通道	输入
i2c_num	I ² C 总线号	输入
send_buf	待传输数据, 一般情况是 i2c 外设的寄存器, 如果没有设置为 NULL	输入
send_buf_len	待传输数据长度, 如果没有则写 0	输入
receive_buf	接收数据内存	输出
receive_buf_len	接收数据的长度	输入

返回值

无

i2c_handle_data_dma**描述**

I2C 使用 dma 传输数据。

函数原型

```
void i2c_handle_data_dma(i2c_device_number_t i2c_num, i2c_data_t data,
plic_interrupt_t *cb);
```

参数

参数名称	描述	输入输出
i2c_num	I2C 总线号	输入
data	I2C 数据相关的参数，详见 i2c_data_t 说明	输入
cb	dma 中断回调函数，如果设置为 NULL 则为阻塞模式，直至传输完毕后退函数	输入

返回值

无

举例

```
/* i2c 外设地址是 0x32, 7 位地址, 速率 200K */
i2c_init(I2C_DEVICE_0, 0x32, 7, 200000);
uint8_t reg = 0;
uint8_t data_buf[2] = {0x00, 0x01}
data_buf[0] = reg;
/* 向 0 寄存器写 0x01 */
i2c_send_data(I2C_DEVICE_0, data_buf, 2);
i2c_send_data_dma(DMAC_CHANNEL0, I2C_DEVICE_0, data_buf, 4);
/* 从 0 寄存器读取 1 字节数据 */
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);
```

数据类型

相关数据类型、数据结构定义如下：

- i2c_device_number_t: i2c 号。
- i2c_slave_handler_t: i2c 从模式的中断处理函数句柄
- i2c_data_t: 使用 dma 传输时数据相关的参数。
- i2c_transfer_mode_t: 使用 DMA 传输数据的模式，发送或接收。

i2c_device_number_t

描述

i2c 编号。

定义

```
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

描述

i2c 从模式的中断处理函数句柄。根据不同的中断状态执行相应的函数操作。

定义

```
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;
```

成员

成员名称	描述
I2C_DEVICE_0	I2C 0
I2C_DEVICE_1	I2C 1
I2C_DEVICE_2	I2C 2

i2c_data_t

描述

使用 dma 传输时数据相关的参数。

定义

```
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;
```

成员

成员名称	描述
tx_channel	发送时使用的 DMA 通道号
rx_channel	发送时使用的 DMA 通道号
tx_buf	发送的数据
tx_len	发送数据的长度
rx_buf	接收的数据
rx_len	接收数据长度
transfer_mode	传输模式，发送或接收

i2c_transfer_mode_t

描述

使用 DMA 传输数据的模式，发送或接收。

定义

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

成员

成员名称	描述
I2C_SEND	发送
I2C_RECEIVE	接收