3.10LCD 显示图片

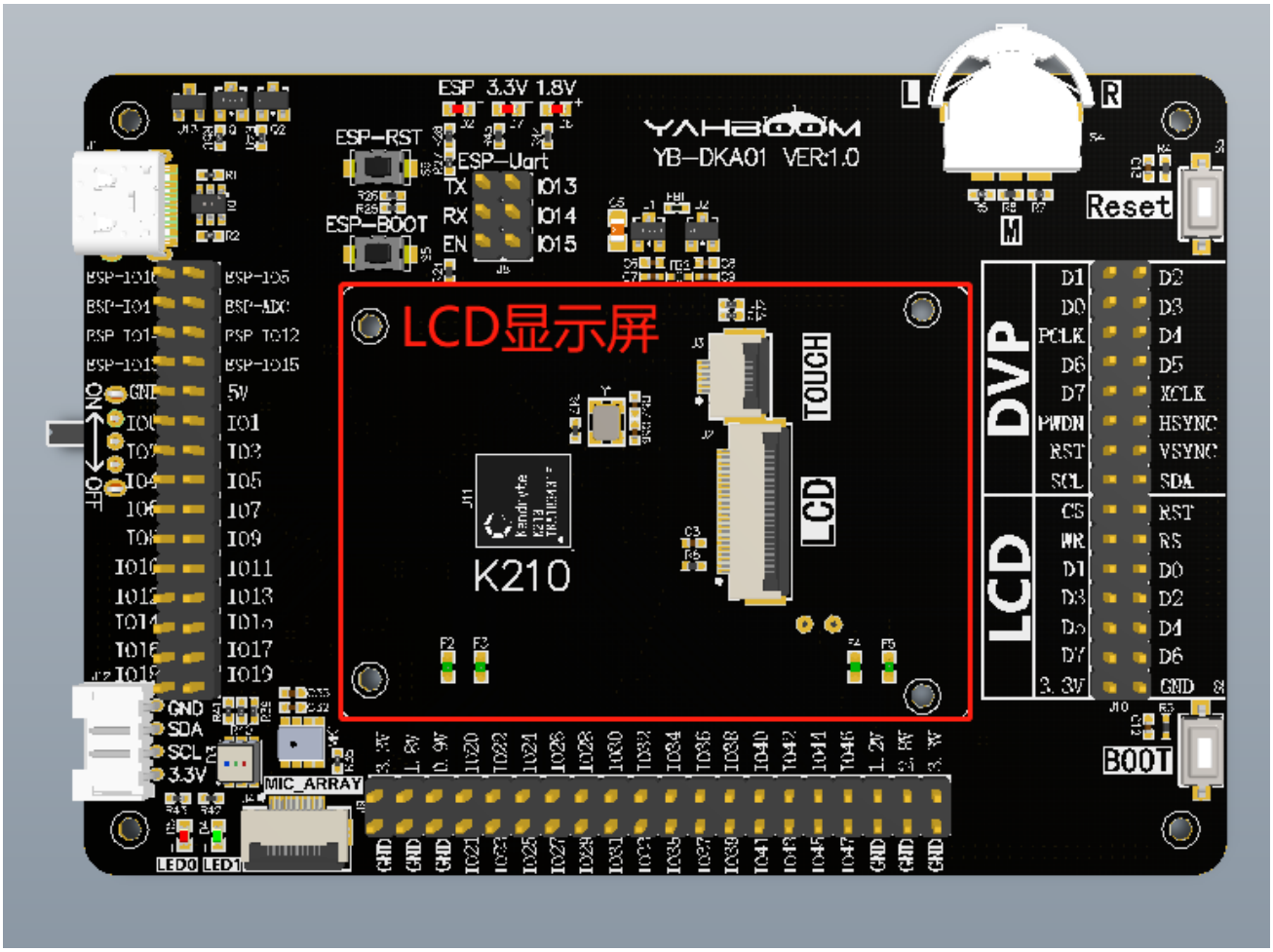
一、实验目的

本节课主要学习 K210 驱动 LCD 显示图片和字符串。

二、实验准备

1.实验元件

LCD 显示屏



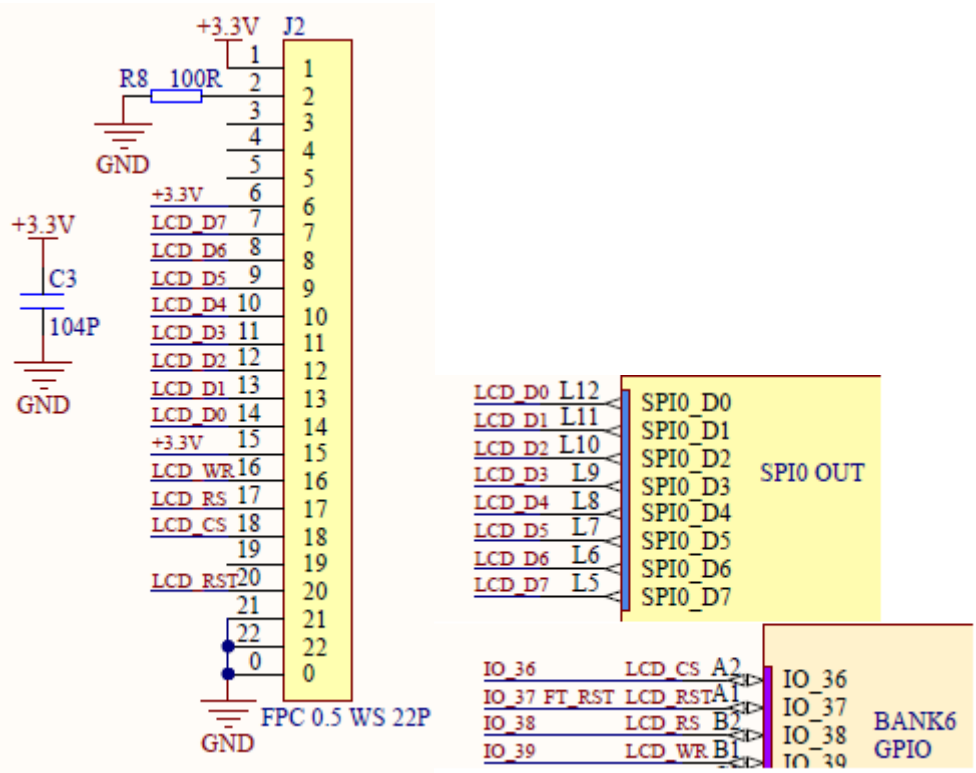
2.元件特性

LCD 显示为 2.0 寸，分辨率为 320*240，驱动芯片是 st7789，体积小，厚度薄，耗能低，工作电压为 3.3V，显示屏的材料为 TFT。

TFT（Thin Film Transistor）是指薄膜晶体管，使每个液晶像素点都是由集成在像素点后面的薄膜晶体管来驱动。从而可以做到高速度、高亮度、高对比度显示屏幕信息。

3.硬件连接

K210 开发板出厂默认已经安装好 LCD 显示屏，其中 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 IO3 上，LCD_WR 连接 IO39 上。



4.SDK 中对应 API 功能

对应的头文件 spi.h

SPI 是一种高速的，全双工，同步的通信总线。

SPI 模块具有以下功能：

独立的 SPI 设备封装外设相关参数，自动处理多设备总线争用，支持标准、双线、四线、八线模式，支持先写后读和全双工读写，支持发送一串相同的数据帧，常见与清屏、填充储存扇区等场景。

为用户提供以下接口：

- spi_init：设置 SPI 工作模式、多线模式和位宽。
- spi_init_non_standard：多线模式下设置指令长度、地址长度、等待时钟数、指令地址传输模式。
- spi_send_data_standard：SPI 标准模式传输数据。
- spi_send_data_standard_dma：SPI 标准模式下使用 DMA 传输数据。
- spi_receive_data_standard：标准模式下接收数据。
- spi_receive_data_standard_dma：标准模式下通过 DMA 接收数据。
- spi_send_data_multiple：多线模式发送数据。
- spi_send_data_multiple_dma：多线模式使用 DMA 发送数据。
- spi_receive_data_multiple：多线模式接收数据。
- spi_receive_data_multiple_dma：多线模式通过 DMA 接收。
- spi_fill_data_dma：通过 DMA 始终发送同一个数据，可以用于刷新数据。
- spi_send_data_normal_dma：通过 DMA 发送数据。不用设置指令地址。
- spi_set_clk_rate：设置 SPI 的时钟频率。
- spi_handle_data_dma：SPI 通过 DMA 传输数据。

三、实验原理

LCD 显示的基本原理是将液晶至于两片导电玻璃基板之间，在上下玻璃基板的两个电极作用下，引起液晶分子扭曲变形，改变通过液晶盒光束的偏振状态，实现对背光源光束的开关控制。

如果液晶盒上不施加外电场，由于 TN 型液晶显示器件中液晶分子在盒中的扭曲螺距远比可见光波长大得多，所以当入射直线偏振光的偏振方向与玻璃表面液晶分子的排列方向一致时，其偏光方向在通过整个液晶层后会随着液晶分子扭曲变形而被扭曲 90。由另一侧射出，呈透光状态。如果这时在液晶盒上施加一个

电压并达到一定值后，液晶分子长轴将开始沿电场方向倾斜，除电极表面的液晶分子外，所有液晶盒内两电极之间的液晶分子都变成沿电场方向的再排列。液晶显示屏技术是根据电压的大小来改变亮度，每个液晶显示屏的子图元显示的颜色取决于色彩筛检程序。

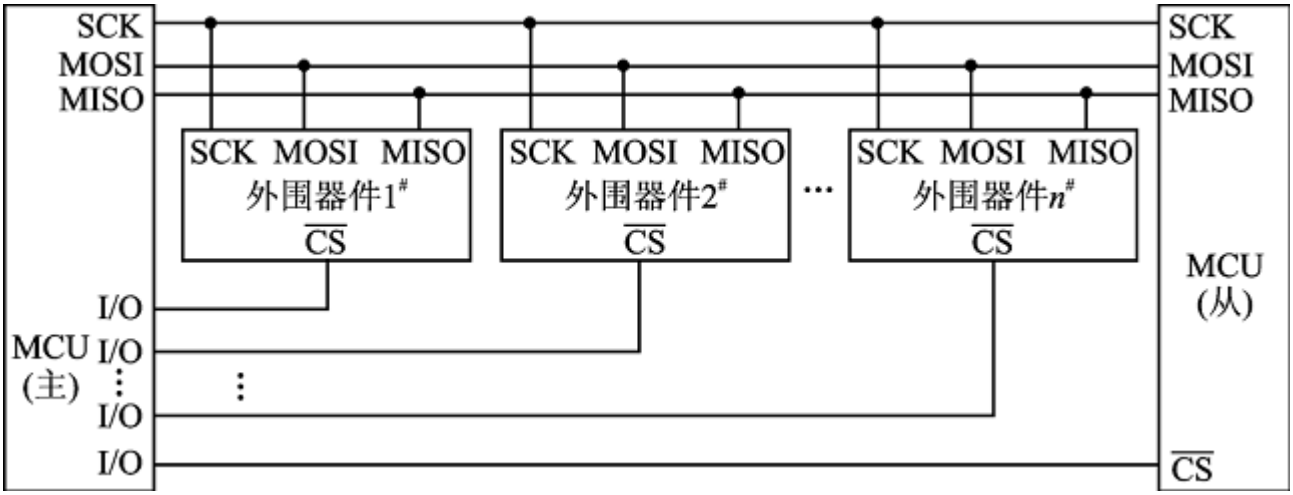
SPI 是一种高速、高效率的串行接口技术。通常由一个主模块和一个或多个从模块组成，主模块选择一个从模块进行同步通信，从而完成数据的交换。SPI 是一个环形结构，通信时需要至少 4 根线（事实上在单向传输时 3 根线也可以），它们是 MISO（主设备数据输入）、MOSI（主设备数据输出）、SCLK（时钟）、CS（片选）。

（1）MISO - Master Input Slave Output,主设备数据输入，从设备数据输出；

（2）MOSI - Master Output Slave Input，主设备数据输出，从设备数据输入；

（3）SCLK - Serial Clock，时钟信号，由主设备产生；

（4）CS - Chip Select，从设备使能信号，由主设备控制。当有多个从设备的时候，因为每个从设备上都有一个片选引脚接入到主设备机中，当我们的主设备和某个从设备通信时将需要将从设备对应的片选引脚电平拉低或者是拉高。



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)

/*****************************SOFTWARE-GPIO************************
// 软件GPIO口，与程序对应
#define LCD_RST_GPIONUM         (0)
#define LCD_RS_GPIONUM          (1)


/*****************************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)
```

```c
void hardware_init(void)
{
    /**
    *PIN_LCD_CS      36
    *PIN_LCD_RST     37
    *PIN_LCD_RS      38
    *PIN_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数据 */
    sysctl_set_spi0_dvp_data(1);

}
```

2. 设置 LCD 的 IO 口电平电压为 1.8V。

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

3. 初始化 LCD，主要的功能就是激活 LCD 显示屏，设置显示方向和显示颜色，并使能显示和清空显示屏。关于详细的 ST7789 的初始化寄存器对应和功能可以查看硬件资料中的 LCD 资料，查看 ST7789 开发手册。

```c
/* 初始化LCD，设置显示方向和启动显示 */
void lcd_init(void)
{
    uint8_t data = 0;
    /* 硬件初始化 */
    tft_hard_init();
    /* 重置LCD */
    tft_write_command(SOFTWARE_RESET);
    usleep(100000);
    /* 关闭睡眠模式 */
    tft_write_command(SLEEP_OFF);
    usleep(100000);
    /* 设置像素格式: 65K, 16bit/pixel */
    tft_write_command(PIXEL_FORMAT_SET);
    data = 0x55;  /* 0101 0101*/
    tft_write_byte(&data, 1);
    /* 打开显示反转 */
    tft_write_command(INVERSION_DISPLAY_ON);
    /* 设置LCD显示方向 */
    lcd_set_direction(DIR_YX_LRUD);

    /* 使能显示 */
    tft_write_command(DISPLAY_ON);
    /* 清空显示 */
    lcd_clear(WHITE);
}
```

4. 显示图片，其中 x1 和 y1 是起点位置，width 为图片的宽度（最大 320），height 为图片的高度（最大 240），ptr 指针指向要显示的图片。

```c
/* LCD画图片 */
void lcd_draw_picture_half(uint16_t x1, uint16_t y1, uint16_t width, uint16_t height, uint16_t *ptr)
{
    lcd_set_area(x1, y1, x1 + width - 1, y1 + height - 1);
    tft_write_half(ptr, width * height);
}
```

5. 显示字符串，其中 x 和 y 为起始坐标，str 指针指向要显示的字符串，color 为字体的颜色。

```c
/* LCD显示字符串 */
void lcd_draw_string(uint16_t x, uint16_t y, char *str, uint16_t color)
{
    while (*str)
    {
        lcd_draw_char(x, y, *str, color);
        str++;
        x += 8;
    }
}
```

```c
/* LCD显示字符 */
void lcd_draw_char(uint16_t x, uint16_t y, char c, uint16_t color)
{
    uint8_t i = 0;
    uint8_t j = 0;
    uint8_t data = 0;

    for (i = 0; i < 16; i++)
    {
        data = ascii0816[c * 16 + i];
        for (j = 0; j < 8; j++)
        {
            if (data & 0x80)
                lcd_draw_point(x + j, y, color);
            data <<= 1;
        }
        y++;
    }
}
```

```c
/* 设置显示某个点的颜色 */
void lcd_draw_point(uint16_t x, uint16_t y, uint16_t color)
{
    lcd_set_area(x, y, x, y);
    tft_write_half(&color, 1);
}
```

6. ST7789 底层 SPI 驱动程序：

开始传输命令、数据

```c
/* 开始传输命令 */
static void set_start_cmd(void)
{
    gpiohs_set_pin(LCD_RS_GPIONUM, GPIO_PV_LOW);
}

/* 开始传输数据 */
static void set_start_data(void)
{
    gpiohs_set_pin(LCD_RS_GPIONUM, GPIO_PV_HIGH);
}
```

SPI 写命令、数据

```c
/* SPI写命令 */
void tft_write_command(uint8_t cmd)
{
    set_start_cmd();
    spi_init(SPI_CHANNEL, SPI_WORK_MODE_0, SPI_FF_OCTAL, 8, 0);
    spi_init_non_standard(SPI_CHANNEL, 8 /*instrction length*/, 0 /*address length*/, 0 /*wait cycles*/,
                          SPI_AITM_AS_FRAME_FORMAT /*spi address trans mode*/);
    spi_send_data_normal_dma(DMAC_CHANNEL0, SPI_CHANNEL, SPI_SLAVE_SELECT, (uint8_t *)(&cmd), 1, SPI_TRANS_CHAR);
}

/* SPI写数据（uint8_t类型） */
void tft_write_byte(uint8_t *data_buf, uint32_t length)
{
    set_start_data();
    spi_init(SPI_CHANNEL, SPI_WORK_MODE_0, SPI_FF_OCTAL, 8, 0);
    spi_init_non_standard(SPI_CHANNEL, 8 /*instrction length*/, 0 /*address length*/, 0 /*wait cycles*/,
                          SPI_AITM_AS_FRAME_FORMAT /*spi address trans mode*/);
    spi_send_data_normal_dma(DMAC_CHANNEL0, SPI_CHANNEL, SPI_SLAVE_SELECT, data_buf, length, SPI_TRANS_CHAR);
}
```

7.main 函数里最后是一个 while(1)循环。

```c
int main(void)
{
    /* 硬件引脚初始化 */
    hardware_init();

    /* 设置IO口电压 */
    io_set_power();

    /* 初始化LCD */
    lcd_init();

    /* 显示图片 */
    lcd_draw_picture_half(0, 0, 320, 240, gImage_logo);
    sleep(1);

    /* 显示字符 */
    lcd_draw_string(16, 40, "Hello Yahboom!", RED);
    lcd_draw_string(16, 60, "Nice to meet you!", BLUE);

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

8. 关于图片是如何转化成数据格式的方法：安装并打开图片转化工具 Image2Lcd，参考以下图片的配置以及操作步骤，再把保存得到的.c 文件复制到项目目录下。

然后再通过 extern 关键字在要使用的地方引用。

```
extern const unsigned char gImage_logo[153608];
```

9. 编译调试，烧录运行

把本课程资料中的 lcd 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=lcd -G "MinGW Makefiles"
make
```

```
[ 93%] Linking C executable lcd
Generating .bin file ...
[100%] Built target lcd
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

编译完成后，在 build 文件夹下会生成 lcd.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

LCD 屏幕会显示图片，一秒后打印出“Hello Yahboom!”“Nice to meet you!”的欢迎语。

六、实验总结

1.LCD 显示屏的分辨率是 320*240 的，显示图片前需要把图片转化成 320*240 分辨率，然后通过图片转化工具把图片转成.c 文件，再引用图片的变量即可。

注意：图片转化工具自动生成的.c 文件如果显示异常，可以屏蔽数组中的部分数据。

```
const unsigned char gImage_logo[153608] = { /*0X00,0X10,0X40,0X01,0XF0,0X00,0X01,0X1B,*/
0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XBE,0XF7,0XBE,0XF7,0X9E,0XEF,0X9E,0XE7,0X7E,0XD7,0XDB,0XB6,
0X19,0X96,0X11,0X4C,0X52,0X4C,0X36,0X75,0X7B,0XA6,0X3E,0XCF,0X5F,0XDF,0X9F,0XEF,
0X9F,0XEF,0XBF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,
0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,
0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,
0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBF,0XF7,0XBE,0XF7,0XBE,0XF7,0XBE,0XF7,0XBE,0XF7,
0X9E,0XF7,0X9E,0XEF,0X9E,0XEF,0X9E,0XEF,0XBE,0XEF,0XBE,0XEF,0XBE,0XEF,0XBE,0XEF,
0XBF,0XEF,0XBF,0XF7,0XBF,0XEF,0XBF,0XEF,0X7E,0XE7,0XFC,0XCE,0X9B,0XC6,0X9B,0XC6,
0X1D,0XD7,0X9E,0XE7,0XBF,0XEF,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XBF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XBF,0XF7,0XBF,0XEF,0X9F,0XE7,0XFC,0XCE,0X7B,0XBE,0XDC,0XCE,
0X5E,0XE7,0XBF,0XEF,0XDF,0XF7,0XDF,0XF7,0XFF,0XF7,0XFF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,
0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,
0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,
0XDF,0XF7,0XDF,0XF7,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,
0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,
0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,0XDF,0XFF,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,0XDF,0XF7,
0XBF,0XF7,0X9F,0XEF,0X9E,0XEF,0X7E,0XEF,0X9F,0XEF,0XBF,0XEF,0X9F,0XEF,0X9F,0XE7,
0X7F,0XE7,0X3E,0XDF,0XFD,0XCE,0XFD,0XCE,0X1D,0XD7,0X7E,0XDF,0XBF,0XEF,0XDF,0XF7,
```

2. LCD 是基于 SPI 通讯的，传输速度快并且稳定。

3. 显示屏在显示前需要配置显示的方向以及显示的格式等参数。

附：API 对应的头文件 spi.h

spi_init

描述

设置 SPI 工作模式、多线模式和位宽。

函数原型

```
void spi_init(spi_device_num_t spi_num, spi_work_mode_t work_mode,
spi_frame_format_t frame_format, size_t data_bit_length, uint32_t endian)
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
work_mode	极性相位的四种模式	输入
frame_format	多线模式	输入
data_bit_length	单次传输的数据的位宽	输入
endian	大小端 0：小端 1：大端	输入

返回值

无。

spi_config_non_standard

描述

多线模式下设置指令长度、地址长度、等待时钟数、指令地址传输模式。

函数原型

```
void spi_init_non_standard(spi_device_num_t spi_num, uint32_t
instruction_length, uint32_t address_length, uint32_t wait_cycles,
spi_instruction_address_trans_mode_t instruction_address_trans_mode)
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
instruction_length	发送指令的位数	输入
address_length	发送地址的位数	输入
wait_cycles	等待时钟个数	输入
instruction_address_trans_mode	指令地址传输的方式	输入

返回值

无

spi_send_data_standard

描述

SPI 标准模式传输数据。

函数原型

```
void spi_send_data_standard(spi_device_num_t spi_num, spi_chip_select_t
chip_select, const uint8_t *cmd_buff, size_t cmd_len, const uint8_t *tx_buff,
size_t tx_len)
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
tx_buff	发送的数据	输入
tx_len	发送数据的长度	输入

返回值

无

spi_send_data_standard_dma

描述

SPI 标准模式下使用 DMA 传输数据。

函数原型

```
void spi_send_data_standard_dma(dmac_channel_number_t channel_num,
spi_device_num_t spi_num, spi_chip_select_t chip_select, const uint8_t
*cmd_buff, size_t cmd_len, const uint8_t *tx_buff, size_t tx_len)
```

参数

参数名称	描述	输入输出
channel_num	DMA 通道号	输入
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
tx_buff	发送的数据	输入

参数名称	描述	输入输出
tx_len	发送数据的长度	输入

返回值

无

spi_receive_data_standard

描述

标准模式下接收数据。

函数原型

```
void spi_receive_data_standard(spi_device_num_t spi_num, spi_chip_select_t chip_select, const uint8_t *cmd_buff, size_t cmd_len, uint8_t *rx_buff, size_t rx_len)
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
rx_buff	接收的数据	输出
rx_len	接收数据的长度	输入

返回值

无

spi_receive_data_standard_dma

描述

标准模式下通过 DMA 接收数据。

函数原型

```
void spi_receive_data_standard_dma(dmac_channel_number_t dma_send_channel_num, dmac_channel_number_t dma_receive_channel_num,
```

```
spi_device_num_t spi_num, spi_chip_select_t chip_select, const uint8_t
*cmd_buff, size_t cmd_len, uint8_t *rx_buff, size_t rx_len)
```

参数

参数名称	描述	输入输出
dma_send_channel_num	发送指令地址使用的 DMA 通道号	输入
dma_receive_channel_num	接收数据使用的 DMA 通道号	输入
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
rx_buff	接收的数据	输出
rx_len	接收数据的长度	输入

返回值

无

spi_send_data_multiple

描述

多线模式发送数据。

函数原型

```
void spi_send_data_multiple(spi_device_num_t spi_num, spi_chip_select_t
chip_select, const uint32_t *cmd_buff, size_t cmd_len, uint8_t *tx_buff, size_t
tx_len)
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
tx_buff	发送的数据	输入
tx_len	发送数据的长度	输入

返回值

无

spi_send_data_multiple_dma

描述

多线模式使用 DMA 发送数据。

函数原型

```
void spi_send_data_multiple_dma(dmac_channel_number_t
channel_num,spi_device_num_t spi_num, spi_chip_select_t chip_select, const
uint32_t *cmd_buff, size_t cmd_len, const uint8_t *tx_buff, size_t tx_len)
```

参数

参数名称	描述	输入输出
channel_num	DMA 通道号	输入
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
tx_buff	发送的数据	输入
tx_len	发送数据的长度	输入

返回值

无

spi_receive_data_multiple

描述

多线模式接收数据。

函数原型

```
void spi_receive_data_multiple(spi_device_num_t spi_num, spi_chip_select_t
chip_select, const uint32_t *cmd_buff, size_t cmd_len, uint8_t *rx_buff, size_t
rx_len)
```

参数

参数名称	描述	输入输出

参数名称	描述	输入输出
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
rx_buff	接收的数据	输出
rx_len	接收数据的长度	输入

返回值

无

spi_receive_data_multiple_dma

描述

多线模式通过 DMA 接收。

函数原型

```
void spi_receive_data_multiple_dma(dmac_channel_number_t
dma_send_channel_num, dmac_channel_number_t dma_receive_channel_num,
spi_device_num_t spi_num, spi_chip_select_t chip_select, uint32_t const
*cmd_buff, size_t cmd_len, uint8_t *rx_buff, size_t rx_len);
```

参数

参数名称	描述	输入输出
dma_send_channel_num	发送指令地址使用的 DMA 通道号	输入
dma_receive_channel_num	接收数据使用的 DMA 通道号	输入
spi_num	SPI 号	输入
chip_select	片选信号	输入
cmd_buff	外设指令地址数据，没有则设为 NULL	输入
cmd_len	外设指令地址数据长度，没有则设为 0	输入
rx_buff	接收的数据	输出
rx_len	接收数据的长度	输入

返回值

无

spi_fill_data_dma

描述

通过 DMA 始终发送同一个数据，可以用于刷新数据。

函数原型

```
void spi_fill_data_dma(dmac_channel_number_t channel_num,spi_device_num_t
spi_num, spi_chip_select_t chip_select, const uint32_t *tx_buff, size_t
tx_len);
```

参数

参数名称	描述	输入输出
channel_num	DMA 通道号	输入
spi_num	SPI 号	输入
chip_select	片选信号	输入
tx_buff	发送的数据,仅发送 tx_buff 这一个数据，不会自动增加	输入
tx_len	发送数据的长度	输入

返回值

无

spi_send_data_normal_dma

描述

通过 DMA 发送数据。不用设置指令地址。

函数原型

```
void spi_send_data_normal_dma(dmac_channel_number_t channel_num,
spi_device_num_t spi_num, spi_chip_select_t chip_select, const void *tx_buff,
size_t tx_len, spi_transfer_width_t spi_transfer_width)
```

参数

参数名称	描述	输入输出
channel_num	DMA 通道号	输入
spi_num	SPI 号	输入
chip_select	片选信号	输入
tx_buff	发送的数据,仅发送 tx_buff 这一个数据，不会自动增加	输入
tx_len	发送数据的长度	输入

参数名称	描述	输入输出
spi_transfer_width	发送数据的位宽	输入

返回值

无

spi_handle_data_dma

描述

SPI 通过 DMA 传输数据。

函数原型

```
void spi_handle_data_dma(spi_device_num_t spi_num, spi_chip_select_t
chip_select, spi_data_t data, plic_interrupt_t *cb)
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
data	SPI 数据相关的参数，详见 spi_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，直至传输完毕后退出函数	输入

返回值

无

spi_slave_config

描述

SPI slave 配置，K210 的 SPI slave 是一条数据线复用为 MOSI 和 MISO 功能。请参考 demo 文件 spi_slave，以及 README.md 的硬件连接。

函数原型

```
void spi_slave_config(uint8_t int_pin, uint8_t ready_pin,
dmac_channel_number_t dmac_channel, size_t data_bit_length, uint8_t *data,
uint32_t len, spi_slave_receive_callback_t callback);
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
ready_pin	spi slave 准备好后会拉高改 pin，master 端触发中断。	输入
dmac_channel	传输数据时用到的 dma 通道	输入
data_bit_length	设置 spi 传输数据时的数据宽度	输入
data	spi slave 接收数据的 buffer	输入
len	spi slave 接收数据的 buffer 的大小	输入
callback	spi slave 接收数据后的回调函数	输入

返回值

无

spi_slave_dual_config

描述

SPI slave 配置，驱动通过 iomux 内部切换 MOSI 和 MISO 功能。

函数原型

```
void spi_slave_dual_config(uint8_t int_pin,
                           uint8_t ready_pin,
                           uint8_t mosi_pin,
                           uint8_t miso_pin,
                           dmac_channel_number_t dmac_channel,
                           size_t data_bit_length,
                           uint8_t *data,
                           uint32_t len,
                           spi_slave_receive_callback_t callback);
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
ready_pin	spi slave 准备好后会拉高改 pin，master 端触发中断。	输入
mosi_pin	MOSI 的 IO 号	输入
miso_pin	MISO 的 IO 号	输入
dmac_channel	传输数据时用到的 dma 通道	输入
data_bit_length	设置 spi 传输数据时的数据宽度	输入
data	spi slave 接收数据的 buffer	输入
len	spi slave 接收数据的 buffer 的大小	输入

参数名称	描述	输入输出
callback	spi slave 接收数据后的回调函数	输入

返回值

无

举例

```
/* SPI0 工作在 MODE0 模式 标准 SPI 模式 单次发送 8 位数据 */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_0, SPI_FF_STANDARD, 8, 0);
uint8_t cmd[4];
cmd[0] = 0x06;
cmd[1] = 0x01;
cmd[2] = 0x02;
cmd[3] = 0x04;
uint8_t data_buf[4] = {0,1,2,3};
/* SPI0 使用片选 0 发送指令 0x06 向地址 0x010204 发送 0，1，2，3 四个字节数据 */
spi_send_data_standard(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf, 4);
/* SPI0 使用片选 0 发送指令 0x06 地址 0x010204 接收 4 个字节的数据 */
spi_receive_data_standard(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf,
4);

/* SPI0 工作在 MODE0 模式 四线 SPI 模式 单次发送 8 位数据 */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_0, SPI_FF_QUAD, 8, 0);
/* 8 位指令长度 32 位地址长度 发送指令地址后等待 4 个 clk，指令通过标准 SPI 方式发送，地址通
过四线方式发送 */
spi_init_non_standard(SPI_DEVICE_0, 8, 32, 4, SPI_AITM_ADDR_STANDARD);
uint32 cmd[2];
cmd[0] = 0x06;
cmd[1] = 0x010204;
uint8_t data_buf[4] = {0,1,2,3};
/* SPI0 使用片选 0 发送指令 0x06 向地址 0x010204 发送 0，1，2，3 四个字节数据 */
spi_send_data_multiple(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf, 4);
/* SPI0 使用片选 0 发送指令 0x06 地址 0x010204 接收 4 个字节的数据 */
spi_receive_data_multiple(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf,
4);

/* SPI0 工作在 MODE2 模式 八线 SPI 模式 单次发送 32 位数据 */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_2, SPI_FF_OCTAL, 32, 0);
/* 无指令 32 位地址长度 发送指令地址后等待 0 个 clk，指令地址通过 8 线发送 */
spi_init_non_standard(SPI_DEVICE_0, 0, 32, 0, SPI_AITM_AS_FRAME_FORMAT);
uint32_t data_buf[256] = {0};
```

```
/* 使用 DMA 通道 0 片选 0 发送 256 个 int 数据*/
spi_send_data_normal_dma(DMAC_CHANNEL0, SPI_DEVICE_0, SPI_CHIP_SELECT_0,
data_buf, 256, SPI_TRANS_INT);
uint32_t data = 0x55AA55AA;
/* 使用 DMA 通道 0 片选 0 连续发送 256 个 0x55AA55AA*/
spi_fill_data_dma(DMAC_CHANNEL0, SPI_DEVICE_0, SPI_CHIP_SELECT_0,&data, 256);
```

spi_set_clk_rate

描述

设置 SPI 的时钟频率

函数原型

```
uint32_t spi_set_clk_rate(spi_device_num_t spi_num, uint32_t spi_clk)
```

参数

参数名称	描述	输入输出
spi_num	SPI 号	输入
spi_clk	目标 SPI 设备的时钟频率	输入

返回值

设置完后的 SPI 设备的时钟频率

数据类型

相关数据类型、数据结构定义如下：

- spi_device_num_t：SPI 编号。
- spi_mode_t：SPI 模式。
- spi_frame_format_t：SPI 帧格式。
- spi_instruction_address_trans_mode_t：SPI 指令和地址的传输模式。
- spi_data_t：使用 dma 传输时数据相关的参数。
- spi_transfer_mode_t：SPI 传输的方式。

spi_device_num_t

描述

SPI 编号。

定义

```
typedef enum _spi_device_num
{
    SPI_DEVICE_0,
    SPI_DEVICE_1,
    SPI_DEVICE_2,
    SPI_DEVICE_3,
    SPI_DEVICE_MAX,
} spi_device_num_t;
```

成员

成员名称	描述
SPI_DEVICE_0	SPI 0 做为主设备
SPI_DEVICE_1	SPI 1 做为主设备
SPI_DEVICE_2	SPI 2 做为从设备
SPI_DEVICE_3	SPI 3 做为主设备

spi_mode_t

描述

SPI 模式。

定义

```
typedef enum _spi_mode
{
    SPI_WORK_MODE_0,
    SPI_WORK_MODE_1,
    SPI_WORK_MODE_2,
    SPI_WORK_MODE_3,
} spi_mode_t;
```

成员

成员名称	描述
SPI_WORK_MODE_0	SPI 模式 0
SPI_WORK_MODE_1	SPI 模式 1
SPI_WORK_MODE_2	SPI 模式 2
SPI_WORK_MODE_3	SPI 模式 3

spi_frame_format_t

描述

SPI 帧格式。

定义

```
typedef enum _spi_frame_format
{
    SPI_FF_STANDARD,
    SPI_FF_DUAL,
    SPI_FF_QUAD,
    SPI_FF_OCTAL
} spi_frame_format_t;
```

成员

成员名称	描述
SPI_FF_STANDARD	标准
SPI_FF_DUAL	双线
SPI_FF_QUAD	四线
SPI_FF_OCTAL	八线（SPI3 不支持）

spi_instruction_address_trans_mode_t

描述

SPI 指令和地址的传输模式。

定义

```
typedef enum _spi_instruction_address_trans_mode
{
    SPI_AITM_STANDARD,
    SPI_AITM_ADDR_STANDARD,
    SPI_AITM_AS_FRAME_FORMAT
} spi_instruction_address_trans_mode_t;
```

成员

成员名称	描述
SPI_AITM_STANDARD	均使用标准帧格式
SPI_AITM_ADDR_STANDARD	指令使用配置的值，地址使用标准帧格式

成员名称	描述
SPI_AITM_AS_FRAME_FORMAT	均使用配置的值

spi_data_t

描述

使用 dma 传输时数据相关的参数。

定义

```
typedef struct _spi_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;
    spi_transfer_mode_t transfer_mode;
    bool fill_mode;
} spi_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
fill_mode	是否以填充方式传输数据，此种情况下 DMA 传输的源地址地址不会自增加

spi_transfer_mode_t

描述

SPI 传输的方式。

定义

```
typedef enum _spi_transfer_mode
{
    SPI_TMOD_TRANS_RECV,
    SPI_TMOD_TRANS,
    SPI_TMOD_RECV,
    SPI_TMOD_EEROM
} spi_transfer_mode_t;
```

成员

成员名称	描述
SPI_TMOD_TRANS_RECV	全双工，边发边收
SPI_TMOD_TRANS	只发送
SPI_TMOD_RECV	只接收
SPI_TMOD_EEROM	先发送后接收