



5.1 点亮 LED 灯

在嵌入式开发中,I/O 口的高低电平控制是最简单的外设控制之一。本节通过一个典型的点亮 LED 灯实验,带大家开启 OpenHarmony 基础外设开发之旅。通过本节的学习,可以了解 RK2206 芯片的 I/O 口控制使用方法。

RK2206 芯片可以提供多达 32 个双向 GPIO 口,它们分别分布在 PA~PD 这 4 个端口中,每个端口有 8 个 GPIO,每个 GPIO 口都可以承受最大 3.3V 的压降。通过 RK2206 芯片寄存器配置,可以将 GPIO 口配置成想要的工作模式。

5.1.1 硬件电路设计

模块硬件电路如图 5.1.1 所示,可以看到 LED 灯引脚连接到 RK2206 芯片的 GPIO0_D3。

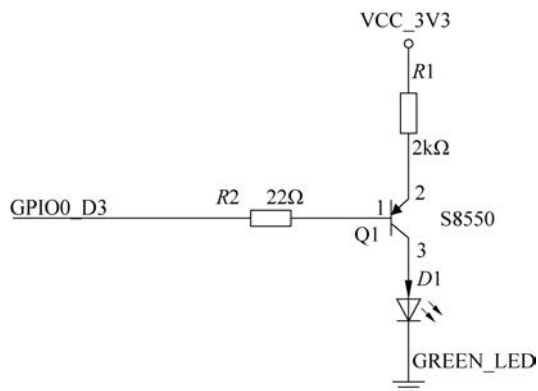


图 5.1.1 硬件电路图

5.1.2 程序设计

通过初始化 LED 灯对应的 GPIO 口,然后每隔 1s 控制 GPIO 输出电平点亮或熄灭

LED 灯。

1. 主程序设计

如图 5.1.2 所示为点亮 LED 灯的主程序流程图。LiteOS 系统初始化后,再初始化 LED 灯的 GPIO 口,并使变量 i 为 0,最后进入死循环。死循环中根据变量 i 控制 GPIO 口输出电平,如果 $i=0$,则输出低电平,设置 $i=1$; 如果 $i=1$,则输出高电平,设置 $i=0$; 最后睡眠 1s。

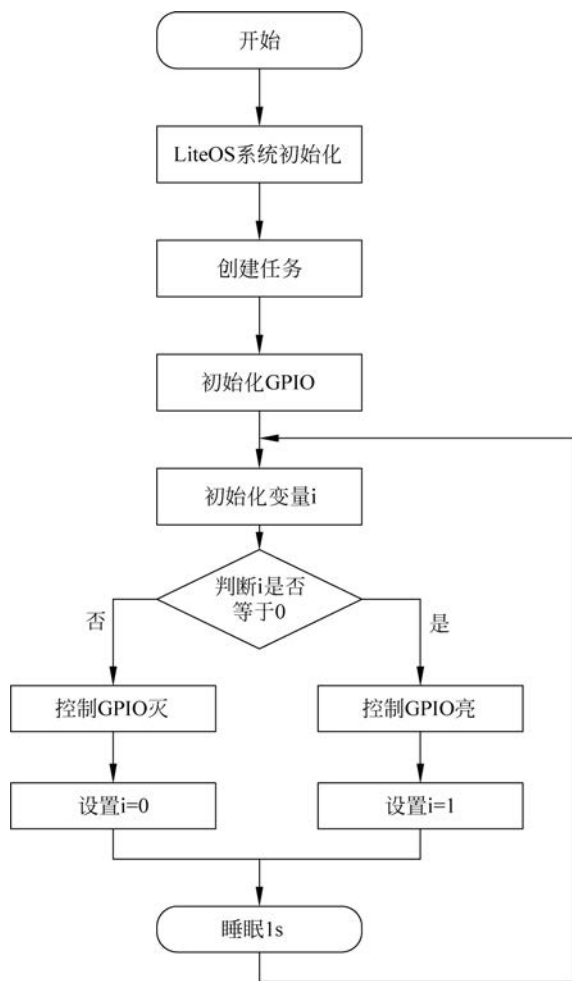


图 5.1.2 主程序流程图

2. GPIO 初始化程序设计

GPIO 初始化程序主要分为 I/O 口初始化和控制 I/O 口输出高电平两部分。

```

void led_init()
{
    /* 配置 GPIO0_PD3 的复用功能寄存器为 GPIO */
}
  
```

```

    PinctrlSet(GPIO0_PD3, MUX_FUNC0, PULL_KEEP, DRIVE_KEEP);
    /* 初始化 GPIO0_PD3 */
    LzGpioInit(GPIO0_PD3);
    /* 设置 GPIO0_PD3 为输出模式 */
    LzGpioSetDir(GPIO0_PD3, LZGPIO_DIR_OUT);
    /* 设置 GPIO0_PD3 输出低电平 */
    LzGpioSetVal(GPIO0_PD3, LZGPIO_LEVEL_LOW);
}

```

3. GPIO 控制 LED 亮灭程序设计

在死循环中,第 1 秒 LED 灯灭,第 2 秒 LED 灯亮,如此反复。

```

void task_led()
{
    uint8_t i;

    /* 初始化 LED 灯的 GPIO 引脚 */
    led_init();
    i = 0;

    while (1)
    {
        if (i == 0)
        {
            printf("Led Off\n");
            /* 控制 GPIO0_PD3 输出低电平 */
            LzGpioSetVal(GPIO0_PD3, LZGPIO_LEVEL_LOW);
            i = 1;
        }
        else
        {
            printf("Led On\n");
            /* 控制 GPIO0_PD3 输出高电平 */
            LzGpioSetVal(GPIO0_PD3, LZGPIO_LEVEL_HIGH);
            i = 0;
        }

        /* 睡眠 1s。该函数为 OpenHarmony 内核睡眠函数,单位:ms */
        LOS_Msleep(1000);
    }
}

```

5.1.3 实验结果

程序编译烧写到开发板后,按下开发板的 RESET 按键,通过串口软件查看日志,程序代码如下:

Led Off
Led On
Led Off
Led On
...

5.2 ADC 按键

在嵌入式系统产品开发中,按键板的设计是最基本的,也是项目评估阶段必须要考虑的问题。其实现方式有很多种,具体使用哪一种需要结合可用 I/O 数量和成本,做出最终选择。传统的按键检测方法是一个按键对应一个 GPIO 口,进行高低电平输入检测。可是在 GPIO 口紧缺的情况下,就需要一个有效的解决方案,其中 ADC 检测实现按键功能是一种相对有效的解决方案。

ADC 检测实现简单实用的按键方法:仅需要一个 ADC 和若干电阻就可实现多个按键的输入检测。ADC 检测的工作原理:按下按键时,通过电阻分压得到不同的电压值,ADC 采集在各个范围内的值来判定是哪个按键被按下。

5.2.1 硬件电路设计

模块整体硬件电路如图 5.2.1 所示,电路中包含了 1 个 ADC 引脚和 4 个按键,USER_KEY_ADC 引脚连接到 RK2206 芯片的 GPIO0_C5。

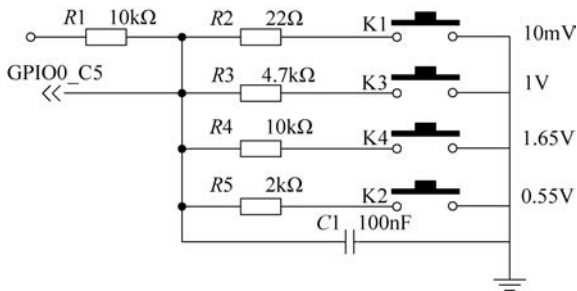


图 5.2.1 硬件电路图

其中,4 个按键分别连接不同的电阻。当按键被按下时,USER_KEY_ADC 检测到不同的电压。按键对应电压如表 5.2.1 所示。

表 5.2.1 按键对应电压表

序 号	按 键	电压/V
1	K1	0.01
2	K2	0.55
3	K3	1.00
4	K4	1.65



视频讲解

5.2.2 程序设计

ADC 按键程序每 1s 通过 GPIO0_PC5 读取一次按键电压,通过电压数值判断当前是哪个按键被按下,并打印出该按键名称。

1. 主程序设计

如图 5.2.2 所示为 ADC 按键主程序流程图,开机 LiteOS 系统初始化后,进入主程序,先初始化 ADC 设备。程序进入主循环,1s 获取一次 ADC 采样电压,判断:

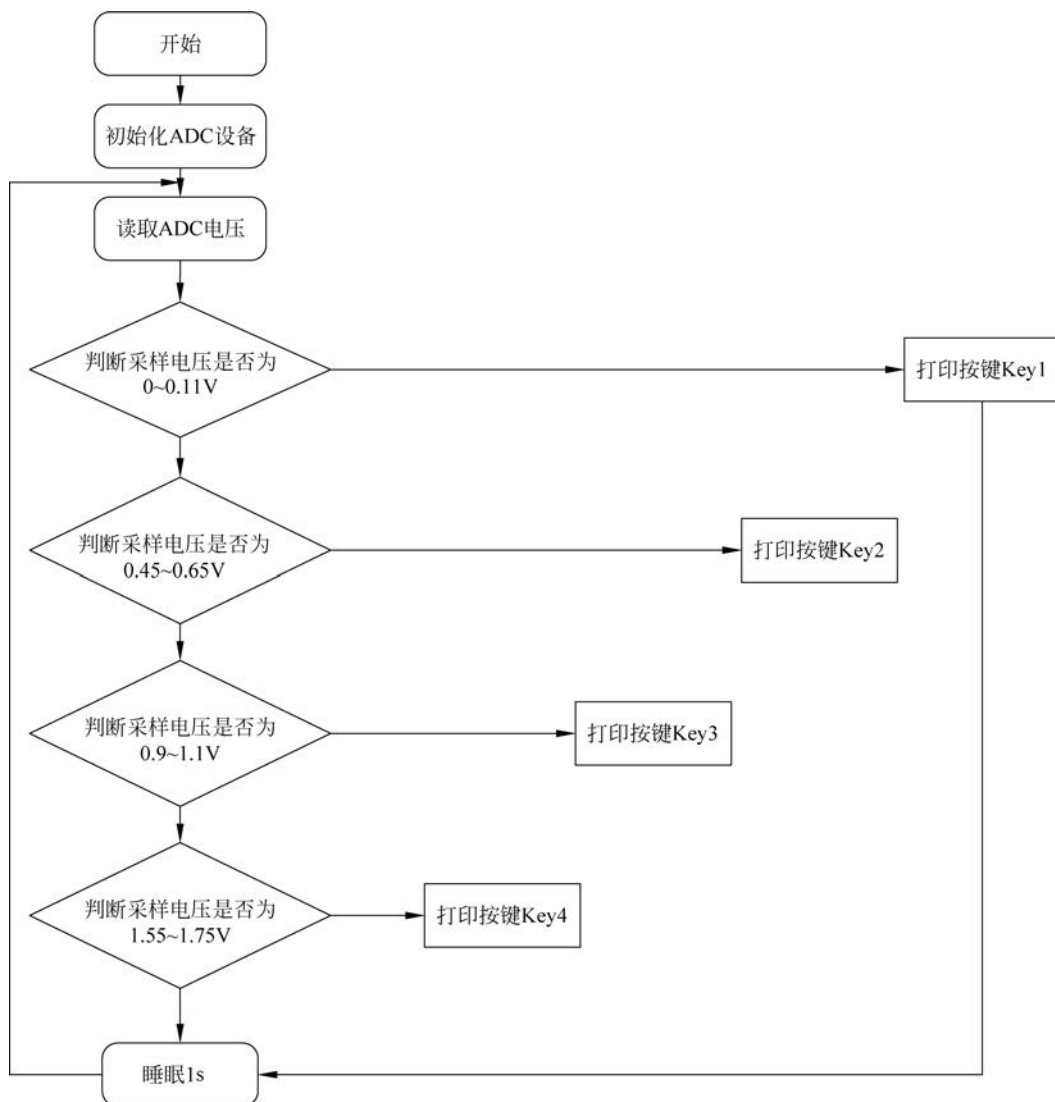


图 5.2.2 主程序流程图

- (1) 若采样电压为 $0\sim 0.11\text{V}$,则当前是按下 K1,打印按键 Key1;
- (2) 若采样电压为 $0.45\sim 0.65\text{V}$,则当前是按下 K2,打印按键 Key2;
- (3) 若采样电压为 $0.9\sim 1.1\text{V}$,则当前是按下 K3,打印按键 Key3;
- (4) 若采样电压为 $1.55\sim 1.75\text{V}$,则当前是按下 K4,打印按键 Key4;
- (5) 当前无按键。

```
void adc_process()
{
    float voltage;

    /* 初始化 ADC 设备 */
    adc_dev_init();

    while (1)
    {
        printf("***** Adc Example ***** \r\n");
        /* 获取电压值 */
        voltage = adc_get_voltage();
        printf("vlt: %.3fV\n", voltage);

        if ((0.11 >= voltage) && (voltage >= 0.00))
        {
            printf("\tKey1\n");
        }
        else if ((0.65 >= voltage) && (voltage >= 0.45))
        {
            printf("\tKey2\n");
        }
        else if ((1.1 >= voltage) && (voltage >= 0.9))
        {
            printf("\tKey3\n");
        }
        else if ((1.75 >= voltage) && (voltage >= 1.55))
        {
            printf("\tKey4\n");
        }

        /* 睡眠 1s */
        LOS_Msleep(1000);
    }
}
```

2. ADC 初始化程序设计

ADC 初始化程序主要分为 ADC 初始化和配置 ADC 参考电压为外部电压两部分。

```
static unsigned int adc_dev_init()
{
    unsigned int ret = 0;
    uint32_t *pGrfSocCon29 = (uint32_t *) (0x41050000U + 0x274U);
    uint32_t ulValue;

    ret = DevIoInit(m_adcKey);
    if (ret != LZ_HARDWARE_SUCCESS)
    {
        printf(" %s, %s, %d: ADC Key IO Init fail\n", __FILE__, __func__, __LINE__);
        return __LINE__;
    }
    ret = LzSaradcInit();
    if (ret != LZ_HARDWARE_SUCCESS) {
        printf(" %s, %s, %d: ADC Init fail\n", __FILE__, __func__, __LINE__);
        return __LINE__;
    }

    /* 设置 saradc 的电压信号, 选择 AVDD */
    ulValue = *pGrfSocCon29;
    ulValue &= ~(0x1 << 4);
    ulValue |= ((0x1 << 4) << 16);
    *pGrfSocCon29 = ulValue;

    return 0;
}
```

3. ADC 读取电压程序设计

RK2206 芯片采用一种逐次逼近寄存器型模数转换器 (Successive-Approximation Analog to Digital Converter), 是一种常用的 A/D 转换结构, 其功耗较低, 转换速率较高, 在有低功耗要求 (可穿戴设备、物联网) 的数据采集场景下应用广泛。该 ADC 采用 10 位采样, 最高电压为 3.3V。简言之, ADC 采样读取的数据, 位 0~位 9 有效, 且最高数值 0x400 (即 1024) 代表实际电压差 3.3V, 也就是说, 1 个数值等于 $3.3\text{V}/1024 \approx 0.003223\text{V}$ 。

```
static float adc_get_voltage()
{
    unsigned int ret = LZ_HARDWARE_SUCCESS;
    unsigned int data = 0;
```

```
ret = LzSaradcReadValue(ADC_CHANNEL, &data);
if (ret != LZ_HARDWARE_SUCCESS)
{
    printf(" %s, %s, %d: ADC Read Fail\n", __FILE__, __func__, __LINE__);
    return 0.0;
}

return (float)(data * 3.3 / 1024.0);
}
```

5.2.3 实验结果

程序编译烧写到开发板后,按下开发板的 RESET 按键,通过串口软件查看日志,程序代码如下:

```
***** Adc Example *****
vlt:3.297V
***** Adc Example *****
vlt:1.67V
Key4
```

5.3 LCD 液晶屏

LCD 的应用很广泛,简单如手表上的液晶显示屏,仪表仪器上的液晶显示器或者是笔记本电脑上的液晶显示器,都使用了 LCD。在一般的办公设备上也很常见,如传真机、复印机,在一些娱乐器材等上也常常见到 LCD 的身影。

本节使用的 LCD 液晶屏采用 ST7789V 驱动器,可单片驱动 262K 色图像的 TFT-LCD,包含 720(240×3 色)×320 线输出,可以直接以 SPI 协议,以及 8 位/9 位/16 位/18 位并行连接外部控制器。ST7789V 显示数据存储于片内 240×320×18 位内存中,显示内存的读写不需要外部时钟驱动。关于 ST7789V 驱动器的详细内容可以查看其芯片手册。

5.3.1 硬件电路设计

模块整体硬件电路如图 5.3.1 所示,电路中包含了电源电路、液晶接口以及小凌派-RK2206 开发板连接的相关引脚。其中,液晶屏 ST7789V 的相关引脚资源如图 5.3.2 所示。

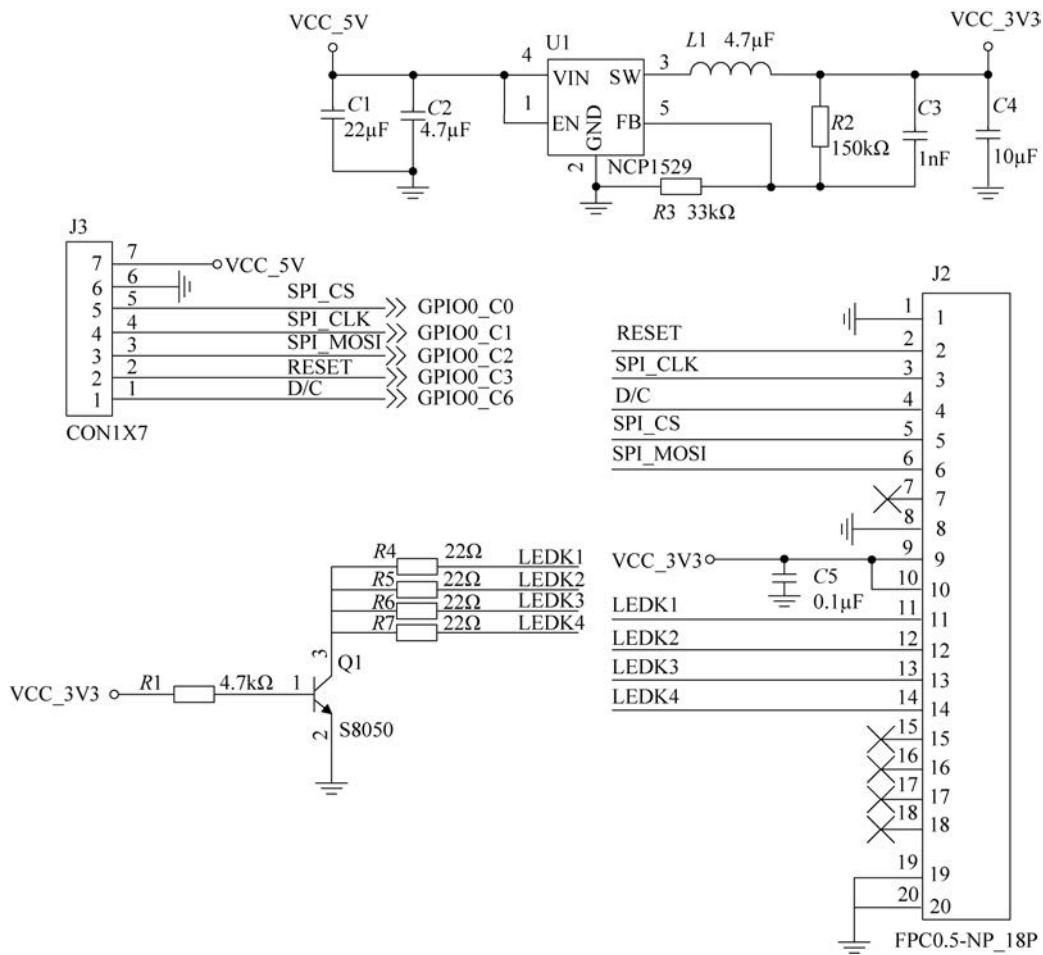


图 5.3.1 硬件电路图

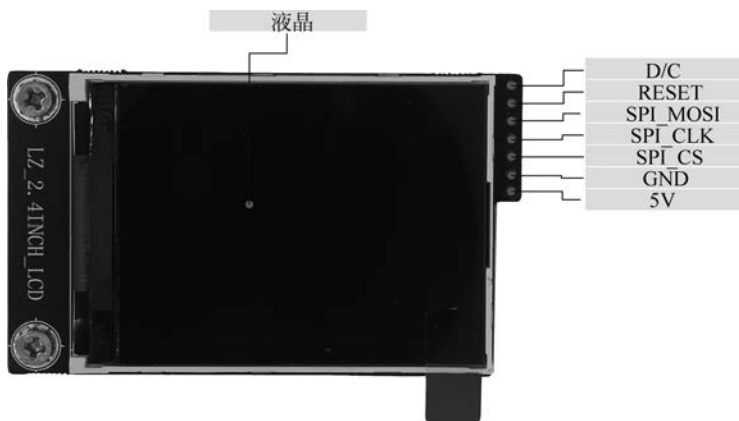


图 5.3.2 液晶屏 ST7789V 硬件资源示意图

LCD 液晶屏引脚功能描述,如表 5.3.1 所示。

表 5.3.1 LCD 液晶屏引脚功能描述

序 号	LCD 引脚	功 能 描 述
1	D/C	指令/数据选择端,L: 指令,H: 数据
2	RESET	复位信号线,低电平有效
3	SPI_MOSI	SPI 数据输入信号线
4	SPI_CLK	SPI 时钟信号线
5	SPI_CS	SPI 片选信号线,低电平有效
6	GND	电源地引脚
7	5V	5V 电源输入引脚

LCD 液晶屏与小凌派-RK2206 开发板连接如图 5.3.3 所示。

5.3.2 程序设计

本节将利用小凌派-RK2206 开发板上的 GPIO 和 SPI 接口方式来点亮 2.4 寸 LCD 液晶屏,并实现 ASCII 字符及汉字的显示。



视频讲解

1. 主程序设计

如图 5.3.4 所示为 LCD 液晶屏主程序流程图,开机 LiteOS 系统初始化后,进入主程序。主程序首先进行 GPIO 和 SPI 总线初始化,然后配置 LCD 液晶屏设备,最后进入循环。在循环中,主程序控制 SPI 对 LCD 液晶屏进行 ASCII 字符和汉字的显示。

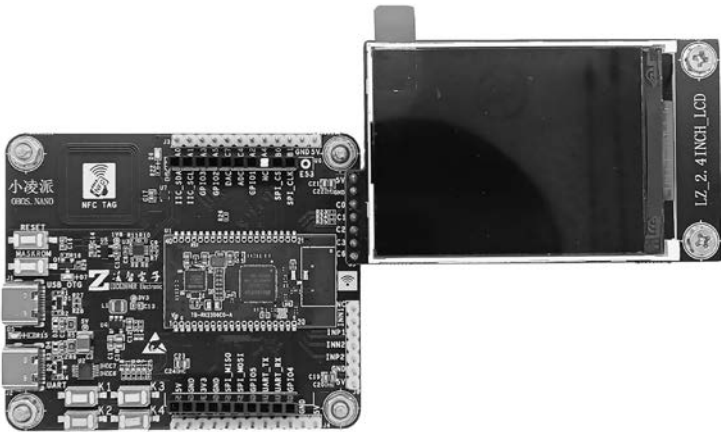


图 5.3.3 2.4 寸 LCD 液晶屏和小凌派-RK2206 开发板连接图

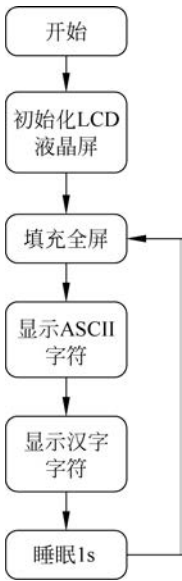


图 5.3.4 主程序流程图

2. LCD 初始化程序设计

LCD 初始化程序主要分为 GPIO 和 SPI 总线初始化及配置 LCD 两部分。

其中,GPIO 初始化首先用 LzGpioInit()函数将 GPIO0_PC3 初始化为 GPIO 引脚,然后用 LzGpioSetDir()将引脚设置为输出模式,最后调用 LzGpioSetVal()输出低电平。

```
/* 初始化 GPIO0_C3 */
LzGpioInit(LCD_PIN_RES);
LzGpioSetDir(LCD_PIN_RES, LZGPIO_DIR_OUT);
LzGpioSetVal(LCD_PIN_RES, LZGPIO_LEVEL_HIGH);

/* 初始化 GPIO0_C6 */
LzGpioInit(LCD_PIN_DC);
LzGpioSetDir(LCD_PIN_DC, LZGPIO_DIR_OUT);
LzGpioSetVal(LCD_PIN_DC, LZGPIO_LEVEL_LOW);
```

SPI 初始化首先用 SpiIoInit()函数将 GPIO0_PC0 复用为 SPI0_CS0n_M1,GPIO0_PC1 复用为 SPI0_CLK_M1,GPIO0_PC2 复用为 SPI0_MOSI_M1。其次调用 LzI2cInit()函数初始化 SPI0 端口。

```
LzSpiDeinit(LCD_SPI_BUS);

if (SpiIoInit(m_spiBus) != LZ_HARDWARE_SUCCESS) {
    printf("%s, %d: SpiIoInit failed!\n", __FILE__, __LINE__);
    return __LINE__;
}
if (LzSpiInit(LCD_SPI_BUS, m_spiConf) != LZ_HARDWARE_SUCCESS) {
    printf("%s, %d: LzSpiInit failed!\n", __FILE__, __LINE__);
    return __LINE__;
}
```

配置 LCD 主要是配置 ST7789V 的工作模式,具体代码如下:

```
/* 重启 LCD */
LCD_RES_Clr();
LOS_Msleep(100);
LCD_RES_Set();
LOS_Msleep(100);
LOS_Msleep(500);
lcd_wr_reg(0x11);
/* 等待 LCD 100ms */
LOS_Msleep(100);
/* 启动 LCD 配置,设置显示和颜色配置 */
lcd_wr_reg(0X36);
if (USE_HORIZONTAL == 0)
{
    lcd_wr_data8(0x00);
}
```

```
else if (USE_HORIZONTAL == 1)
{
    lcd_wr_data8(0xC0);
}
else if (USE_HORIZONTAL == 2)
{
    lcd_wr_data8(0x70);
}
else
{
    lcd_wr_data8(0xA0);
}
lcd_wr_reg(0X3A);
lcd_wr_data8(0X05);
/* ST7789V 帧刷新率设置 */
lcd_wr_reg(0xb2);
lcd_wr_data8(0x0c);
lcd_wr_data8(0x0c);
lcd_wr_data8(0x00);
lcd_wr_data8(0x33);
lcd_wr_data8(0x33);
lcd_wr_reg(0xb7);
lcd_wr_data8(0x35);
/* ST7789V 电源设置 */
lcd_wr_reg(0xbb);
lcd_wr_data8(0x35);
lcd_wr_reg(0xc0);
lcd_wr_data8(0x2c);
lcd_wr_reg(0xc2);
lcd_wr_data8(0x01);
lcd_wr_reg(0xc3);
lcd_wr_data8(0x13);
lcd_wr_reg(0xc4);
lcd_wr_data8(0x20);
lcd_wr_reg(0xc6);
lcd_wr_data8(0x0f);
lcd_wr_reg(0xca);
lcd_wr_data8(0x0f);
lcd_wr_reg(0xc8);
lcd_wr_data8(0x08);
lcd_wr_reg(0x55);
```

```
lcd_wr_data8(0x90);  
lcd_wr_reg(0xd0);  
lcd_wr_data8(0xa4);  
lcd_wr_data8(0xa1);  
/* ST7789V gamma 设置 */  
lcd_wr_reg(0xe0);  
lcd_wr_data8(0xd0);  
lcd_wr_data8(0x00);  
lcd_wr_data8(0x06);  
lcd_wr_data8(0x09);  
lcd_wr_data8(0x0b);  
lcd_wr_data8(0x2a);  
lcd_wr_data8(0x3c);  
lcd_wr_data8(0x55);  
lcd_wr_data8(0x4b);  
lcd_wr_data8(0x08);  
lcd_wr_data8(0x16);  
lcd_wr_data8(0x14);  
lcd_wr_data8(0x19);  
lcd_wr_data8(0x20);  
lcd_wr_reg(0xe1);  
lcd_wr_data8(0xd0);  
lcd_wr_data8(0x00);  
lcd_wr_data8(0x06);  
lcd_wr_data8(0x09);  
lcd_wr_data8(0x0b);  
lcd_wr_data8(0x29);  
lcd_wr_data8(0x36);  
lcd_wr_data8(0x54);  
lcd_wr_data8(0x4b);  
lcd_wr_data8(0x0d);  
lcd_wr_data8(0x16);  
lcd_wr_data8(0x14);  
lcd_wr_data8(0x21);  
lcd_wr_data8(0x20);  
lcd_wr_reg(0x29);
```

3. LCD 的点数据设计

ST7789V 采用 4 线串行 SPI 通信方式,数据位共 16 位,其 RGB 分别是 5 位、6 位和 5 位,也就是共 65K 个颜色,寄存器 3AH 的值设置为 05H。其数据传输时序图如图 5.3.5 所示。

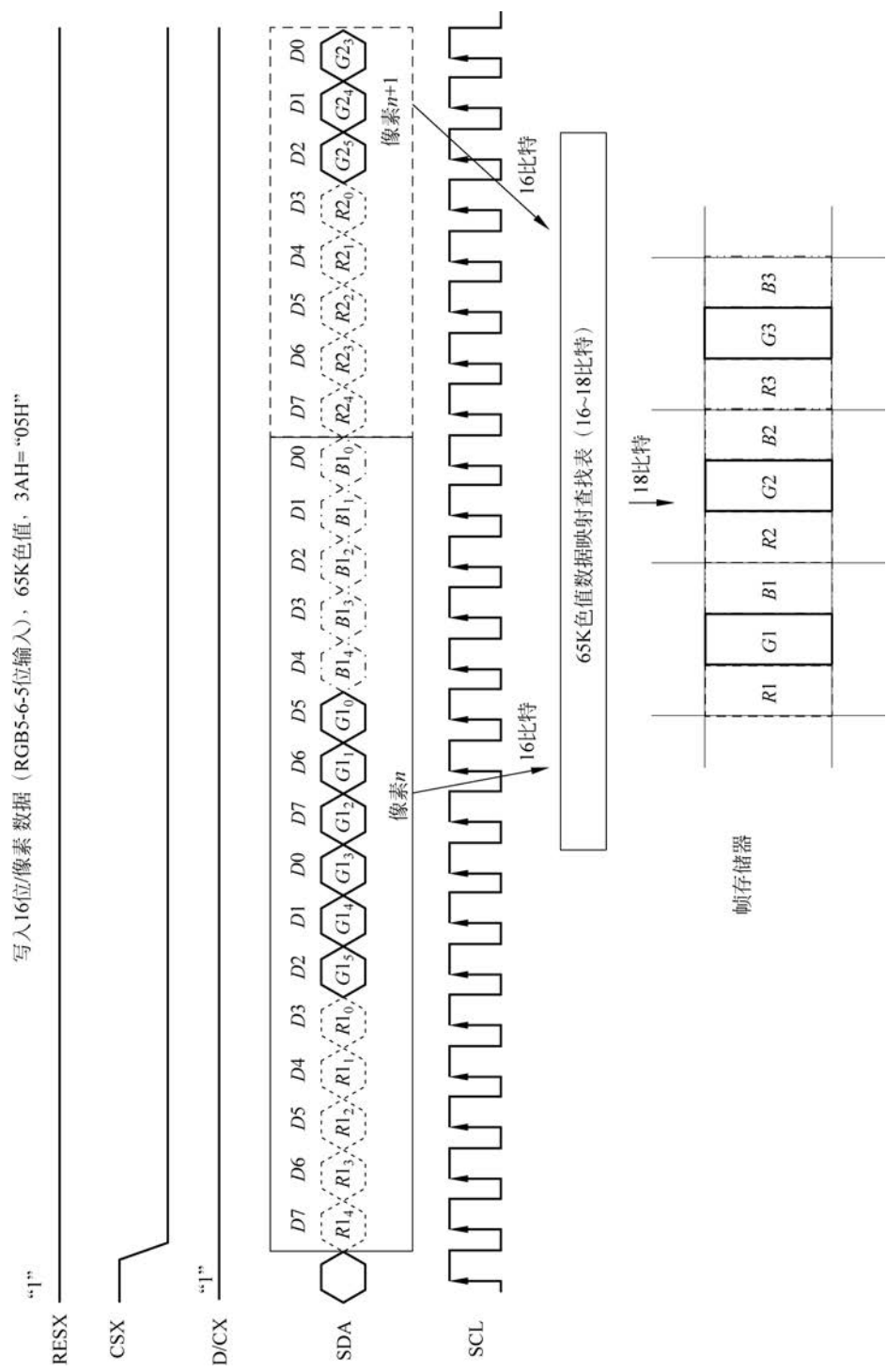


图 5.3.5 ST7789V 液晶屏 SPI 数据传输时序图

也就是一个像素点的 RGB 为 5 位+6 位+5 位,每个像素点需要占用 2 字节存储空间。因此,向 LCD 液晶屏发送某个像素信息的程序代码如下:

```
static void lcd_write_bus(uint8_t dat)
{
    LzSpiWrite(LCD_SPI_BUS, 0, &dat, 1);
}

static void lcd_wr_data(uint16_t dat)
{
    lcd_write_bus(dat >> 8);
    lcd_write_bus(dat);
}

static void lcd_wr_reg(uint8_t dat)
{
    LCD_DC_Clr();
    lcd_write_bus(dat);
    LCD_DC_Set();
}

static void lcd_address_set(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)
{
    /* 列地址设置 */
    lcd_wr_reg(0x2a);
    lcd_wr_data(x1);
    lcd_wr_data(x2);
    /* 行地址设置 */
    lcd_wr_reg(0x2b);
    lcd_wr_data(y1);
    lcd_wr_data(y2);
    /* 写存储器 */
    lcd_wr_reg(0x2c);
}

static void lcd_wr_data(uint16_t dat)
{
    lcd_write_bus(dat >> 8);
    lcd_write_bus(dat);
}

void lcd_draw_point(uint16_t x, uint16_t y, uint16_t color)
{
    /* 设置光标位置 */
    lcd_address_set(x, y, x, y);
    lcd_wr_data(color);
}
```

4. LCD 的 ASCII 字符显示设计

预先将规定字号的 ASCII 字符的 LCD 液晶屏像素信息存放于在 lcd_font.h 源代码文件中。LCD 液晶屏依照 ASCII 的数值来存放像素信息。例如,空格的 ASCII 数值是 0x00,则程序将像素放到第一行像素中,具体代码如下:

[illegible]

当需要将某个字号的 ASCII 字符投射到 LCD 液晶屏时,程序根据字号大小找到对应的 ASCII 字符像素表,然后根据 ASCII 字符的数值找到对应的像素行,最后将该像素行数据依次通过 SPI 总线发送给 LCD 液晶屏,具体代码如下:

```
void lcd_show_char(uint16_t x, uint16_t y, uint8_t num, uint16_t fc, uint16_t bc, uint8_t
sizey, uint8_t mode)
{
    uint8_t temp, size_x, t, m = 0;
    uint16_t i;
    uint16_t TypefaceNum;                //一个字符所占字节大小
    uint16_t x0 = x;

    size_x = sizey/2;
    TypefaceNum = (size_x/8 + ((size_x%8)?1:0)) * sizey;

    /* 得到偏移后的值 */
    num = num - ' ';
    /* 设置光标位置 */
    lcd_address_set(x, y, x + size_x - 1, y + sizey - 1);

    for (i = 0; i < TypefaceNum; i++)
    {
        if (sizey == 12)
        {
            /* 调用 6x12 字体 */
            temp = ascii_1206[num][i];
        }
        else if (sizey == 16)
        {
            /* 调用 8x16 字体 */
            temp = ascii_1608[num][i];
        }
        else if (sizey == 24)
        {
            /* 调用 12x24 字体 */
            temp = ascii_2412[num][i];
        }
        else if (sizey == 32)
        {
            /* 调用 16x32 字体 */
            temp = ascii_3216[num][i];
        }
    }
}
```

```

    }
    else
    {
        return;
    }

    for (t = 0; t < 8; t++)
    {
        if (!mode)
        { /* 非叠加模式 */
            if (temp & (0x01 << t))
            {
                lcd_wr_data(fc);
            }
            else
            {
                lcd_wr_data(bc);
            }

            m++;
            if (m % sizex == 0)
            {
                m = 0;
                break;
            }
        }
        else
        { /* 叠加模式 */
            if (temp & (0x01 << t))
            {
                /* 画一个点 */
                lcd_draw_point(x, y, fc);
            }

            x++;
            if ((x - x0) == sizex)
            {
                x = x0;
                y++;
                break;
            }
        }
    }
}
}
}

```

5. LCD 的汉字显示设计

原理同上,程序将某一个特定字号的汉字信息存放于一个数据结构体数组中。该数据结构体包含字体编码 Index 和像素数据 Msk,具体代码如下:

```
/* 定义中文字符 12 * 12 */
typedef struct
{
    unsigned char Index[2];
    unsigned char Msk[24];
} typFNT_GB12;

/* 定义中文字符 16 * 16 */
typedef struct
{
    unsigned char Index[2];
    unsigned char Msk[32];
} typFNT_GB16;

/* 定义中文字符 24 * 24 */
typedef struct
{
    unsigned char Index[2];
    unsigned char Msk[72];
} typFNT_GB24;

...
```

通过汉字像素软件将对应的汉字和像素存放于 lcd_font.h 文件中,具体代码如下:

```
const typFNT_GB12 tfont12[ ] =
{
    "小", 0x20, 0x00, 0x20, 0x00, 0x20, 0x00, 0x20, 0x00, 0x24, 0x01, 0x24, 0x02, 0x22, 0x02,
    0x22, 0x04, 0x21, 0x04, 0x20, 0x00, 0x20, 0x00, 0x38, 0x00, /* "小" */

    "凌", 0x40, 0x00, 0xF9, 0x03, 0x42, 0x00, 0xFC, 0x07, 0x10, 0x01, 0x28, 0x02, 0xE0, 0x01,
    0x14, 0x01, 0xAA, 0x00, 0x41, 0x00, 0xB0, 0x01, 0x0C, 0x06, /* "凌" */

    "派", 0x00, 0x03, 0xF2, 0x00, 0x14, 0x02, 0xD0, 0x01, 0x51, 0x01, 0x52, 0x05, 0x50, 0x03,
    0x50, 0x01, 0x54, 0x01, 0x52, 0x02, 0xD1, 0x02, 0x48, 0x04, /* "派" */

};

const typFNT_GB16 tfont16[ ] =
{
```

```

        "小", 0x80, 0x00, 0x80, 0x00, 0x80, 0x00, 0x80, 0x00, 0x80, 0x00, 0x88, 0x08, 0x88, 0x10,
        0x88, 0x20, 0x84, 0x20, 0x84, 0x40, 0x82, 0x40, 0x81, 0x40, 0x80, 0x00, 0x80, 0x00, 0xA0,
        0x00, 0x40, 0x00, /* "小", 0 */

        "凌", 0x00, 0x02, 0x02, 0x02, 0xC4, 0x1F, 0x04, 0x02, 0x00, 0x02, 0xE0, 0x7F, 0x88, 0x08,
        0x48, 0x11, 0x24, 0x21, 0x87, 0x0F, 0xC4, 0x08, 0x24, 0x05, 0x04, 0x02, 0x04, 0x05, 0xC4,
        0x08, 0x30, 0x30, /* "凌", 1 */

        "派", 0x00, 0x10, 0x04, 0x3C, 0xE8, 0x03, 0x28, 0x00, 0x21, 0x38, 0xA2, 0x07, 0xA2, 0x04,
        0xA8, 0x44, 0xA8, 0x24, 0xA4, 0x14, 0xA7, 0x08, 0xA4, 0x08, 0xA4, 0x10, 0x94, 0x22, 0x94,
        0x41, 0x88, 0x00, /* "派", 2 */

};
...

```

当程序需要将某个特定字号的汉字投射到 LCD 液晶屏时,程序就根据对应的字号查找对应字号的 tfontXX 数组,并将对应的像素行数据发送给 LCD 液晶屏,具体代码如下:

```

void lcd_show_chinese(uint16_t x, uint16_t y, uint8_t * s, uint16_t fc, uint16_t bc, uint8_t
sizey, uint8_t mode)
{
    uint8_t buffer[128];
    uint32_t buffer_len = 0;
    uint32_t len = strlen(s);

    memset(buffer, 0, sizeof(buffer));
    /* UTF8 格式汉字转化为 ASCII 格式 */
    chinese_utf8_to_ascii(s, strlen(s), buffer, &buffer_len);

    for (uint32_t i = 0; i < buffer_len; i += 2, x += sizey)
    {
        if (sizey == 12)
        {
            lcd_show_chinese_12x12(x, y, &buffer[i], fc, bc, sizey, mode);
        }
        else if (sizey == 16)
        {
            lcd_show_chinese_16x16(x, y, &buffer[i], fc, bc, sizey, mode);
        }
    }
}

```

```

        else if (sizey == 24)
        {
            lcd_show_chinese_24x24(x, y, &buffer[i], fc, bc, sizey, mode);
        }
        else if (sizey == 32)
        {
            lcd_show_chinese_32x32(x, y, &buffer[i], fc, bc, sizey, mode);
        }
        else
        {
            return;
        }
    }
}

```

5.3.3 实验结果

程序编译烧写到开发板后,按下开发板的 RESET 按键,通过串口软件查看日志,程序代码如下:

```

***** Lcd Example *****

***** Lcd Example *****

```

5.4 EEPROM 存储

在实际应用中,保存在 RAM 中的数据掉电后就丢失了,保存在 Flash 中的数据又不能随意改变,也就是不能用它来记录变化的数值。在某些特定场合,需要记录下某些数据,并且它们时常会改变或更新,掉电之后数据还不能丢失。例如,家用电表度数、电视机的频道记忆,一般都是使用 EEPROM(Electrically Erasable Programmable Read Only Memory)来保存数据的,其特点就是掉电后存储的数据不丢失。

EEPROM 是指电可擦可编程只读存储器,是一种掉电后数据不丢失的存储芯片。可以通过计算机或专用设备擦除 EEPROM 的已有信息,重新编程。一般情况下,EEPROM 拥有 30 万~100 万次的寿命,也就是它可以反复写入 30 万~100 万次,而读取次数是无限的。

5.4.1 硬件电路设计

以人体感应模块为例,整体硬件电路如图 5.4.1 所示,电路中包含了 E53 接口连接器和 EEPROM 存储器。

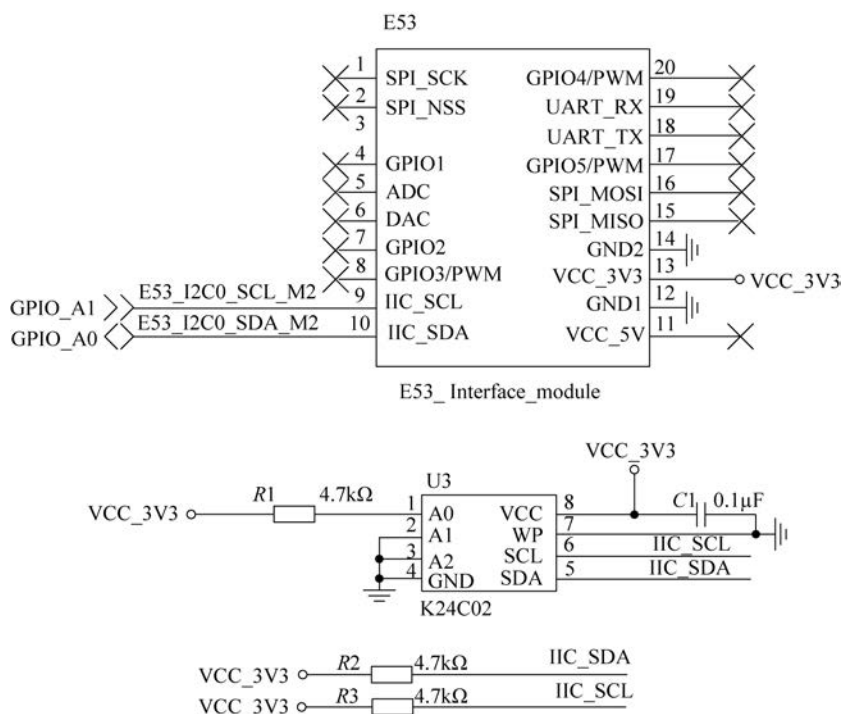


图 5.4.1 硬件电路图

设计使用的 EEPROM 型号是 K24C02,它是一个常用的基于 I²C 通信协议的 EEPROM 元件,例如,ATMEL 公司的 AT24C02、CATALYST 公司的 CAT24C02 和 ST 公司的 ST24C02 等芯片。I²C 是一个通信协议,它拥有严密的通信时序逻辑要求,而 EEPROM 是一个元件,只是这个元件采用了 I²C 协议的接口与单片机相连,二者并没有必然的联系,EEPROM 可以用其他接口,I²C 也可以用在其他很多元件上。根据 K24C02 芯片手册,可获取如下信息。

1. K24C02 芯片的从设备地址

因其存储容量为 2Kb,所以该芯片 I²C 从设备读写地址分别为 51H 和 50H,如图 5.4.2 所示。

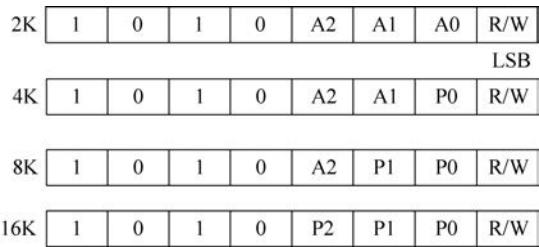


图 5.4.2 K24C02 的从设备地址图

2. K24C02 芯片的读操作

K24C02 芯片的读操作共分为 3 种,分别为当前地址读(Current Address Read)、随机读(Random Read)和连续读(Sequential Read)。

当前地址读(Current Address Read)操作是控制 I²C 总线与 K24C02 芯片通信,通信内容为:从设备地址(1 字节,最低位为 1,表示读)+数据(1 字节,K24C02 发送给 CPU 的存储内容)。该读操作没有附带 EEPROM 的存储地址,存储地址是由上一次存储地址累加而来,如图 5.4.3 所示。

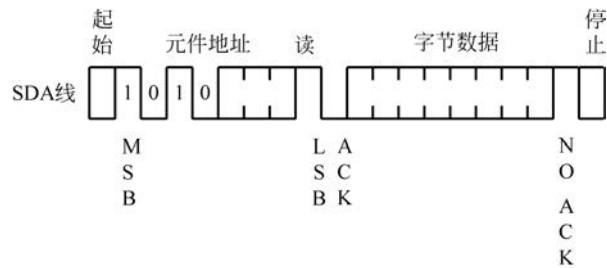


图 5.4.3 K24C02 的当前地址读操作

而随机读操作则控制 I²C 与 K24C02 进行两次通信：

第一次 I²C 通信：从设备地址(1 字节,最低位为 0,表示写)+存储地址(1 字节,CPU 发送给 K24C02 的存储地址)。

第二次 I²C 通信：从设备地址(1 字节,最低位为 1,表示读)+数据(1 字节,K24C02 发送给 CPU 的存储内容)。

具体数据传输如图 5.4.4 所示。

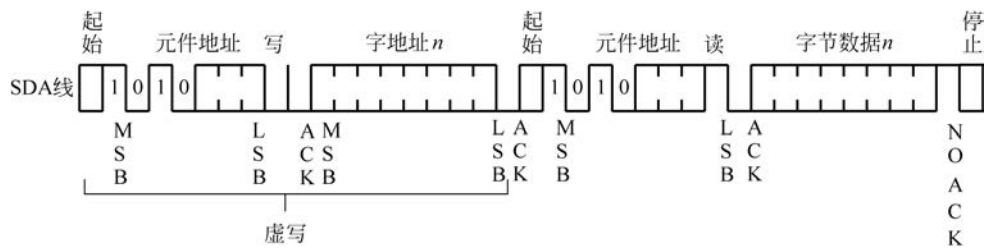


图 5.4.4 K24C02 的随机地址读操作

连续读操作(Sequential Read)则控制 I²C 往 K24C02 发送 n 字节,通信内容为:从设备地址(1 字节,最低位为 1,表示读)+ n 个数据(K24C02 发送给 CPU)。具体数据传输如图 5.4.5 所示。

3. K24C02 芯片的写操作

K24C02 芯片写数据操作可分为两种,分别为字节写操作(Byte Write)和页写操作(Page Write)。

其中,字节写操作(Byte Write)控制 I²C 与 K24C02 通信,通信内容为:从设备地址(1

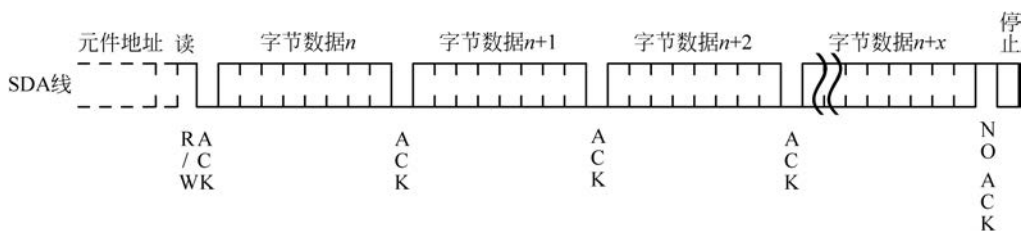


图 5.4.5 K24C02 的连续读操作

字节,最低位为 0,表示写)+存储地址(1 字节)+数据(1 字节,CPU 发送给 K24C0 的存储内容)。具体数据传输如图 5.4.6 所示。

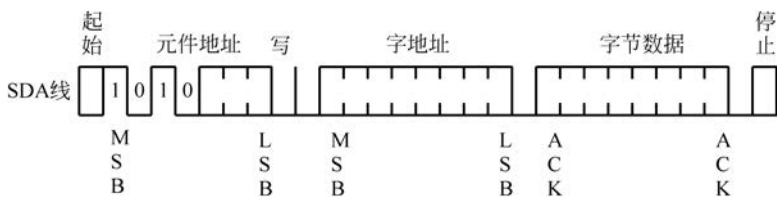


图 5.4.6 K24C02 的字节写操作

页写操作(Page Write)则控制 I²C 与 K24C02 通信,通信内容为:从设备地址(1 字节,最低位为 0,表示写)+存储地址(1 字节)+数据(n 字节,CPU 发送给 K24C0 的存储内容)。其中,存储数据的 n 字节, n 不能超过页大小(K24C02 的页大小为 8 字节)。

小凌派-RK2206 开发板与人体感应模块均带有防呆设计,故很容易区分安装方向,直接将模块插入开发板的 E53 母座接口上即可,如图 5.4.7 所示。

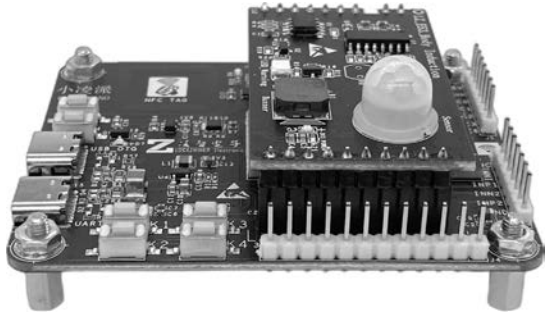


图 5.4.7 硬件连接图

5.4.2 程序设计

可通过程序控制 RK2206 芯片的 I²C 与 K24C02 芯片通信,每 5s 向某一块存储空间(该存储空间地址依次累加)写入不同数据,然后再读取出来。

1. 主程序设计

如图 5.4.8 所示为 EEPROM 存储主程序流程图。主程序首先初始化 I²C 总线，接着程序进入主循环，每 5s 将不同的数据写入一块存储空间，然后再读取出来。其中，存储空间地址每次循环都累加 32，数据也随着循环而累加 1。

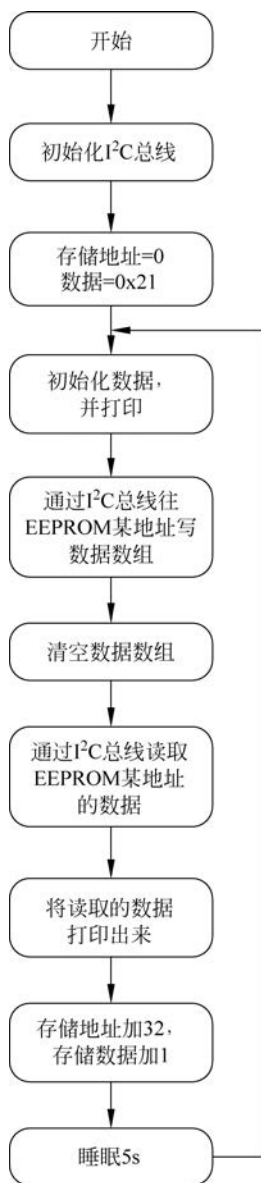


图 5.4.8 主程序流程图

```

while (1)
{
    printf("***** Eeprom Process ***** \n");
    printf("BlockSize = 0x%x\n", eeprom_get_blocksize());

    /* 写 EEPROM */
    memset(buffer, 0, sizeof(buffer));
    for (unsigned int i = 0; i < FOR_CHAR; i++)
    {
        buffer[i] = data_offset + i;
        printf("Write Byte: %d = %c\n", addr_offset + i, buffer[i]);
    }
    ret = eeprom_write(addr_offset, buffer, FOR_CHAR);
    if (ret != FOR_CHAR)
    {
        printf("EepromWrite failed( %d)\n", ret);
    }

    /* 读 EEPROM */
    memset(buffer, 0, sizeof(buffer));
    ret = eeprom_read(addr_offset, buffer, FOR_CHAR);
    if (ret != FOR_CHAR)
    {
        printf("Read Bytes: failed!\n");
    }
    else
    {
        for (unsigned int i = 0; i < FOR_CHAR; i++)
        {
            printf("Read Byte: %d = %c\n", addr_offset + i, buffer[i]);
        }
    }

    data_offset++;
    if (data_offset >= CHAR_END)
    {
        data_offset = CHAR_START;
    }

    addr_offset += FOR_ADDRESS;
    if (addr_offset >= 200)
    {
        addr_offset = 0;
    }
    printf("\n");

    LOS_Msleep(5000);
}

```

2. EEPROM 初始化程序设计

主程序通过控制 RK2206 芯片的接口对 I²C 总线进行初始化。

```
#define EEPROM_I2C_BUS      0
#define EEPROM_I2C_ADDRESS  0x51

static I2cBusIo m_i2cBus = {
    .scl = {.gpio = GPIO0_PA1, .func = MUX_FUNC3, .type = PULL_NONE, .drv = DRIVE_KEEP,
    .dir = LZGPIO_DIR_KEEP, .val = LZGPIO_LEVEL_KEEP},
    .sda = {.gpio = GPIO0_PA0, .func = MUX_FUNC3, .type = PULL_NONE, .drv = DRIVE_KEEP,
    .dir = LZGPIO_DIR_KEEP, .val = LZGPIO_LEVEL_KEEP},
    .id = FUNC_ID_I2C0,
    .mode = FUNC_MODE_M2,
};

static unsigned int m_i2c_freq = 100000;

unsigned int eeprom_init()
{
    if (I2cIoInit(m_i2cBus) != LZ_HARDWARE_SUCCESS) {
        printf(" %s, %d: I2cIoInit failed!\n", __FILE__, __LINE__);
        return __LINE__;
    }

    if (LzI2cInit(EEPROM_I2C_BUS, m_i2c_freq) != LZ_HARDWARE_SUCCESS) {
        printf(" %s, %d: I2cInit failed!\n", __FILE__, __LINE__);
        return __LINE__;
    }

    /* GPIO0_A0 => I2C1_SDA_M1 */
    PinctrlSet(GPIO0_PA0, MUX_FUNC3, PULL_NONE, DRIVE_KEEP);
    /* GPIO0_A1 => I2C1_SCL_M1 */
    PinctrlSet(GPIO0_PA1, MUX_FUNC3, PULL_NONE, DRIVE_KEEP);

    return 0;
}
```

3. EEPROM 读操作程序设计

蜂鸣器控制程序当开启蜂鸣器时,打开蜂鸣器,并且设置 PWM 波的周期为 100ms,其中占空比为 50%;当关闭蜂鸣器时,停止蜂鸣器。

```

#define EEPROM_I2C_BUS                0
#define EEPROM_I2C_ADDRESS            0x51

/* EEPROM 型号:K24C02,2Kb(256B),32 页,每页 8 字节 */
#define EEPROM_ADDRESS_MAX            256
#define EEPROM_PAGE                    8

unsigned int eeprom_readbyte(unsigned int addr, unsigned char * data)
{
    unsigned int ret = 0;
    unsigned char buffer[1];
    LzI2cMsg msgs[2];

    /* K24C02 的存储地址是 0~255 */
    if (addr >= EEPROM_ADDRESS_MAX) {
        printf(" %s, %s, %d: addr(0x%x) >= EEPROM_ADDRESS_MAX(0x%x)\n", __FILE__,
__func__, __LINE__, addr, EEPROM_ADDRESS_MAX);
        return 0;
    }

    buffer[0] = (unsigned char)addr;
    msgs[0].addr = EEPROM_I2C_ADDRESS;
    msgs[0].flags = 0;
    msgs[0].buf = &buffer[0];
    msgs[0].len = 1;
    msgs[1].addr = EEPROM_I2C_ADDRESS;
    msgs[1].flags = I2C_M_RD;
    msgs[1].buf = data;
    msgs[1].len = 1;
    ret = LzI2cTransfer(EEPROM_I2C_BUS, msgs, 2);
    if (ret != LZ_HARDWARE_SUCCESS) {
        printf(" %s, %s, %d: LzI2cTransfer failed(%d)!\n", __FILE__, __func__, __LINE__,
ret);
        return 0;
    }

    return 1;
}

unsigned int eeprom_read(unsigned int addr, unsigned char * data, unsigned int data_len)
{
    unsigned int ret = 0;

    if (addr >= EEPROM_ADDRESS_MAX) {
        printf(" %s, %s, %d: addr(0x%x) >= EEPROM_ADDRESS_MAX(0x%x)\n", __FILE__,
__func__, __LINE__, addr, EEPROM_ADDRESS_MAX);

```

```

        return 0;
    }

    if ((addr + data_len) > EEPROM_ADDRESS_MAX) {
        printf(" %s, %s, %d: addr + len(0x%x) > EEPROM_ADDRESS_MAX(0x%x)\n", __FILE__,
            __func__, __LINE__, addr + data_len, EEPROM_ADDRESS_MAX);
        return 0;
    }

    ret = eeprom_readbyte(addr, data);
    if (ret != 1) {
        printf(" %s, %s, %d: EepromReadByte failed( %d)\n", __FILE__, __func__, __LINE__,
            ret);
        return 0;
    }

    if (data_len > 1) {
        ret = LzI2cRead(EEPROM_I2C_BUS, EEPROM_I2C_ADDRESS, &data[1], data_len - 1);
        if (ret < 0) {
            printf(" %s, %s, %d: LzI2cRead failed( %d)!\n", __FILE__, __func__, __LINE__,
                ret);
            return 0;
        }
    }

    return data_len;
}

```

4. EEPROM 写操作程序设计

主程序根据存储地址、存储数据和数据长度的不同,选用字节写操作或页写操作,具体代码如下:

```

#define EEPROM_I2C_BUS        0
#define EEPROM_I2C_ADDRESS    0x51

/* EEPROM 型号:K24C02,2Kb(256B),32 页,每页 8 字节 */
#define EEPROM_ADDRESS_MAX    256
#define EEPROM_PAGE            8

unsigned int eeprom_writebyte(unsigned int addr, unsigned char data)
{
    unsigned int ret = 0;
    LzI2cMsg msgs[1];
    unsigned char buffer[2];

```

```

/* K24C02 的存储地址是 0~255 */
if (addr >= EEPROM_ADDRESS_MAX) {
    printf(" %s, %s, %d: addr(0x%x) >= EEPROM_ADDRESS_MAX(0x%x)\n", __FILE__,
__func__, __LINE__, addr, EEPROM_ADDRESS_MAX);
    return 0;
}

buffer[0] = (unsigned char)(addr & 0xFF);
buffer[1] = data;
msgs[0].addr = EEPROM_I2C_ADDRESS;
msgs[0].flags = 0;
msgs[0].buf = &buffer[0];
msgs[0].len = 2;
ret = LzI2cTransfer(EEPROM_I2C_BUS, msgs, 1);
if (ret != LZ_HARDWARE_SUCCESS) {
    printf(" %s, %s, %d: LzI2cTransfer failed(%d)!\n", __FILE__, __func__, __LINE__,
ret);
    return 0;
}

/* K24C02 芯片需要时间完成写操作,在此之前不响应其他操作 */
eeprog_delay_usec(1000);
return 1;
}

unsigned int eeprom_writepage(unsigned int addr, unsigned char *data, unsigned int data_len)
{
    unsigned int ret = 0;
    LzI2cMsg msgs[1];
    unsigned char buffer[EEPROM_PAGE + 1];

    /* K24C02 的存储地址是 0~255 */
    if (addr >= EEPROM_ADDRESS_MAX) {
        printf(" %s, %s, %d: addr(0x%x) >= EEPROM_ADDRESS_MAX(0x%x)\n", __FILE__,
__func__, __LINE__, addr, EEPROM_ADDRESS_MAX);
        return 0;
    }

    if ((addr % EEPROM_PAGE) != 0) {
        printf(" %s, %s, %d: addr(0x%x) is not page addr(0x%x)\n", __FILE__, __func__,
__LINE__, addr, EEPROM_PAGE);
        return 0;
    }

    if ((addr + data_len) > EEPROM_ADDRESS_MAX) {

```

```

        printf(" %s, %s, %d: addr + data_len(0x%x) > EEPROM_ADDRESS_MAX(0x%x)\n",
__FILE__, __func__, __LINE__, addr + data_len, EEPROM_ADDRESS_MAX);
        return 0;
    }

    if (data_len > EEPROM_PAGE) {
        printf(" %s, %s, %d: data_len(%d) > EEPROM_PAGE(%d)\n", __FILE__, __func__,
__LINE__, data_len, EEPROM_PAGE);
        return 0;
    }

    buffer[0] = addr;
    memcpy(&buffer[1], data, data_len);
    msgs[0].addr = EEPROM_I2C_ADDRESS;
    msgs[0].flags = 0;
    msgs[0].buf = &buffer[0];
    msgs[0].len = 1 + data_len;
    ret = LzI2cTransfer(EEPROM_I2C_BUS, msgs, 1);
    if (ret != LZ_HARDWARE_SUCCESS) {
        printf(" %s, %s, %d: LzI2cTransfer failed(%d)!\n", __FILE__, __func__, __LINE__,
ret);
        return 0;
    }

    /* K24C02 芯片需要时间完成写操作,在此之前不响应其他操作 */
    eeprog_delay_usec(1000);
    return data_len;
}

unsigned int eeprom_write(unsigned int addr, unsigned char *data, unsigned int data_len)
{
    unsigned int ret = 0;
    unsigned int offset_current = 0;
    unsigned int page_start, page_end;
    unsigned char is_data_front = 0;
    unsigned char is_data_back = 0;
    unsigned int len;

    if (addr >= EEPROM_ADDRESS_MAX) {
        printf(" %s, %s, %d: addr(0x%x) >= EEPROM_ADDRESS_MAX(0x%x)\n", __FILE__,
__func__, __LINE__, addr, EEPROM_ADDRESS_MAX);
        return 0;
    }

    if ((addr + data_len) > EEPROM_ADDRESS_MAX) {
        printf(" %s, %s, %d: addr + len(0x%x) > EEPROM_ADDRESS_MAX(0x%x)\n", __FILE__,
__func__, __LINE__, addr + data_len, EEPROM_ADDRESS_MAX);
        return 0;
    }

```

```

}

/* 判断 addr 是否为页地址 */
page_start = addr / EEPROM_PAGE;
if ((addr % EEPROM_PAGE) != 0) {
    page_start += 1;
    is_data_front = 1;
}

/* 判断 addr + data_len 是否为页地址 */
page_end = (addr + data_len) / EEPROM_PAGE;
if ((addr + data_len) % EEPROM_PAGE != 0) {
    page_end += 1;
    is_data_back = 1;
}

offset_current = 0;

/* 处理前面非页地址的数据,如果是页地址则不执行 */
for (unsigned int i = addr; i < (page_start * EEPROM_PAGE); i++) {
    ret = eeprom_writebyte(i, data[offset_current]);
    if (ret != 1) {
        printf("%s, %s, %d: EepromWriteByte failed(%d)\n", __FILE__, __func__,
__LINE__, ret);
        return offset_current;
    }
    offset_current++;
}

/* 处理后续的数据,如果数据长度不足一页,则不执行 */
for (unsigned int page = page_start; page < page_end; page++) {
    len = EEPROM_PAGE;
    if ((page == (page_end - 1)) && (is_data_back)) {
        len = (addr + data_len) % EEPROM_PAGE;
    }

    ret = eeprom_writepage(page * EEPROM_PAGE, &data[offset_current], len);
    if (ret != len) {
        printf("%s, %s, %d: EepromWritePage failed(%d)\n", __FILE__, __func__,
__LINE__, ret);
        return offset_current;
    }
    offset_current += EEPROM_PAGE;
}

return data_len;
}

```

5.4.3 实验结果

程序编译烧写到开发板后,按下开发板的 RESET 按键,通过串口软件查看日志,具体内容如下:

```
***** Eeprom Process *****  
BlockSize = 0x8  
Write Byte: 3 = !  
Write Byte: 4 = "  
Write Byte: 5 = #  
Write Byte: 6 = $  
Write Byte: 7 = %  
Write Byte: 8 = &  
Write Byte: 9 = '  
Write Byte: 10 = (  
Write Byte: 11 = )  
Write Byte: 12 = *  
Write Byte: 13 = +  
Write Byte: 14 = ,  
Write Byte: 15 = -  
Write Byte: 16 = .  
Write Byte: 17 = /  
Write Byte: 18 = 0  
Write Byte: 19 = 1  
Write Byte: 20 = 2  
Write Byte: 21 = 3  
Write Byte: 22 = 4  
Write Byte: 23 = 5  
Write Byte: 24 = 6  
Write Byte: 25 = 7  
Write Byte: 26 = 8  
Write Byte: 27 = 9  
Write Byte: 28 = :  
Write Byte: 29 = ;  
Write Byte: 30 = <  
Write Byte: 31 = =  
Write Byte: 32 = >  
Read Byte: 3 = !  
Read Byte: 4 = "  
Read Byte: 5 = #  
Read Byte: 6 = $  
Read Byte: 7 = %  
Read Byte: 8 = &  
Read Byte: 9 = '  
Read Byte: 10 = (  
Read Byte: 11 = )  
Read Byte: 12 = *
```

```
Read Byte: 13 = +
Read Byte: 14 = ,
Read Byte: 15 = -
Read Byte: 16 = .
Read Byte: 17 = /
Read Byte: 18 = 0
Read Byte: 19 = 1
Read Byte: 20 = 2
Read Byte: 21 = 3
Read Byte: 22 = 4
Read Byte: 23 = 5
Read Byte: 24 = 6
Read Byte: 25 = 7
Read Byte: 26 = 8
Read Byte: 27 = 9
Read Byte: 28 = :
Read Byte: 29 = ;
Read Byte: 30 = <
Read Byte: 31 = =
Read Byte: 32 = >
...
```

5.5 NFC 碰一碰

NFC(Near Field Communication, 近距离无线通信)是由飞利浦公司发起,由诺基亚、索尼等著名厂商联合主推的一项无线技术。NFC 由非接触式射频识别(Radio Frequency Identification, RFID)及互联互通技术整合演变而来,在单一芯片上结合感应式读卡器、感应式卡片和点对点的功能,能在短距离内与兼容设备进行识别和数据交换。这项技术最初只是 RFID 技术和网络技术的简单合并,现在已经演变成一种短距离无线通信技术,发展相当迅速。与 RFID 不同的是, NFC 具有双向连接和识别的特点,工作于 13.56MHz 频率,作用距离为 10cm 左右。NFC 技术在 ISO 18092、ECMA 340 和 ETSI TS 102 190 框架下推动标准化,同时也兼容应用广泛的 ISO 14443 TYPE-A、TYPE-B 以及 Felica 标准非接触式智能卡的基础架构。

使用 NFC 技术的设备(如智能手机)可以在彼此靠近的情况下进行数据交换,是通过在单一芯片上集成感应式读卡器、感应式卡片和点对点通信,实现移动终端移动支付、门禁、移动身份识别等功能。

5.5.1 硬件电路设计

硬件电路如图 5.5.1 所示, NT3H1201 是一款简单、低成本的 NFC 芯片,通过 I²C 接口

和微控制器通信。芯片通过 PCB 电路板上的射频天线从接触的有源 NFC 设备获取能量，并完成数据交互。交互的数据被写入片上的 EEPROM 存储器，以便掉电后的再次读写。

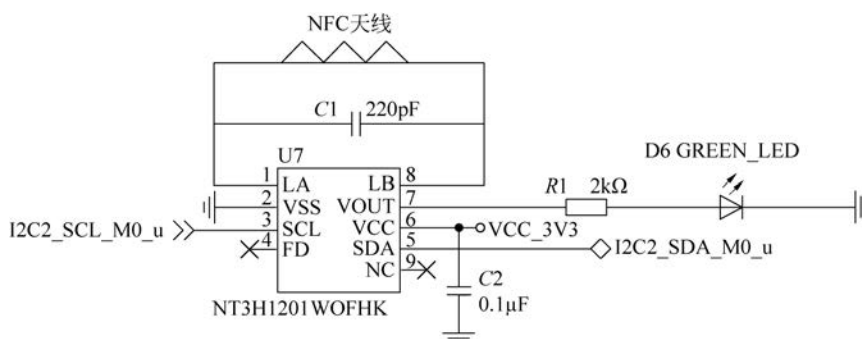


图 5.5.1 硬件电路图



视频讲解

5.5.2 程序设计

与以往设备配网技术相比，NFC“碰一碰”方案可以支持 NFC 功能的安卓手机和 iOS 13.0 以上系统的 iPhone 使用，从而为消费者提供高效便捷的智慧生活无缝体验。

1. 主程序设计

如图 5.5.2 所示为 NFC 碰一碰主程序流程图，首先初始化 I²C 总线，然后控制 I²C 总线向 NFC 写入一段文本信息和一段网址信息，最后使用支持 NFC 功能的安卓手机或 iOS 13.0 以上系统的 iPhone 靠近小凌派-RK2206 开发板，就可以识别出一段文本信息和一个网址。



图 5.5.2 主程序流程图

程序代码如下：

```
void nfc_process(void)
{
    unsigned int ret = 0;

    /* 初始化 NFC 设备 */
    nfc_init();

    ret = nfc_store_text(NDEFFirstPos, (uint8_t *)TEXT);
    if (ret != 1) {
        printf("NFC Write Text Failed: %d\n", ret);
    }

    ret = nfc_store_uri_http(NDEFLastPos, (uint8_t *)WEB);
    if (ret != 1) {
        printf("NFC Write Url Failed: %d\n", ret);
    }

    while (1) {
        printf("===== NFC Example ===== \r\n");
        printf("Please use the mobile phone with NFC function close to the development board!\r\n");
        printf("\n\n");
        LOS_Msleep(1000);
    }
}
```

2. NFC 初始化程序设计

NFC 碰一碰初始化主要包括 I²C 总线初始化。

```
/* NFC 使用 I2C 的总线 ID */
static unsigned int NFC_I2C_PORT = 2;

/* I2C 配置 */
static I2cBusIo m_i2c2m0 =
{
    .scl = {.gpio = GPIO0_PD6, .func = MUX_FUNC1, .type = PULL_NONE, .drv = DRIVE_KEEP,
    .dir = LZGPIO_DIR_KEEP, .val = LZGPIO_LEVEL_KEEP},
    .sda = {.gpio = GPIO0_PD5, .func = MUX_FUNC1, .type = PULL_NONE, .drv = DRIVE_KEEP,
    .dir = LZGPIO_DIR_KEEP, .val = LZGPIO_LEVEL_KEEP},
    .id = FUNC_ID_I2C2,
    .mode = FUNC_MODE_M0,
};
/* I2C 的时钟频率 */
```

```

static unsigned int m_i2c2_freq = 400000;

unsigned int NT3HI2cInit()
{
    uint32_t *pGrf = (uint32_t *)0x41050000U;
    uint32_t ulValue;

    ulValue = pGrf[7];
    ulValue &= ~(0x7 << 8) | (0x7 << 4);
    ulValue |= ((0x1 << 8) | (0x1 << 4));
    pGrf[7] = ulValue | (0xFFFF << 16);
    printf("%s, %d: GRF_GPIO0D_IOMUX_H(0x%x) = 0x%x\n", __func__, __LINE__, &pGrf[7],
pGrf[7]);

    if (I2cIoInit(m_i2c2m0) != LZ_HARDWARE_SUCCESS)
    {
        printf("%s, %s, %d: I2cIoInit failed!\n", __FILE__, __func__, __LINE__);
        return __LINE__;
    }
    if (LzI2cInit(NFC_I2C_PORT, m_i2c2_freq) != LZ_HARDWARE_SUCCESS)
    {
        printf("%s, %s, %d: LzI2cInit failed!\n", __FILE__, __func__, __LINE__);
        return __LINE__;
    }

    return 0;
}

unsigned int nfc_init(void)
{
    unsigned int ret = 0;
    uint32_t *pGrf = (uint32_t *)0x41050000U;
    uint32_t ulValue;

    if (m_nfc_is_init == 1)
    {
        printf("%s, %s, %d: Nfc readily init!\n", __FILE__, __func__, __LINE__);
        return __LINE__;
    }

    ret = NT3HI2cInit();
    if (ret != 0)
    {
        printf("%s, %s, %d: NT3HI2cInit failed!\n", __FILE__, __func__, __LINE__);
        return __LINE__;
    }

    m_nfc_is_init = 1;
    return 0;
}

```

3. NFC 写入数据程序设计

下面实现向 NFC 芯片写入 NDEF 数据包的程序。其中,NDEF 数据包可包含多个 Record 信息段;每个 Record 信息段可分为两大数据部分,分别为头部信息(即 Header)和主体信息(即 Payload,也就是传输信息内容);头部信息又可分为 3 个数据部分,分别为标识符信息(即 Identifier)、长度(即 Record 的大小信息)和类型信息,如图 5.5.3 所示。

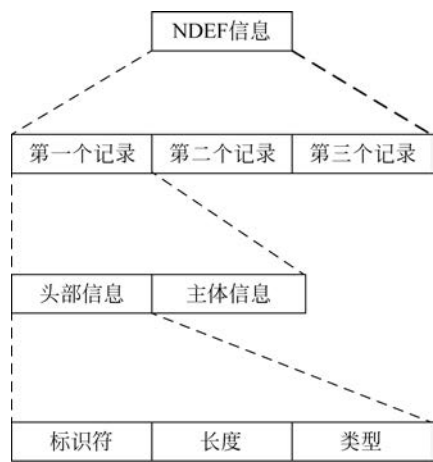


图 5.5.3 NDEF 协议格式

```
ret = nfc_store_text(NDEFFirstPos, (uint8_t *)TEXT);
if (ret != 1) {
    printf("NFC Write Text Failed: %d\n", ret);
}

ret = nfc_store_uri_http(NDEFLastPos, (uint8_t *)WEB);
if (ret != 1) {
    printf("NFC Write Url Failed: %d\n", ret);
}
```

其中,nfc_store_text()和 nfc_store_uri_http()两个函数首先按照 rtdText.h 和 rtdUri.h 中 RTD 协议进行处理,然后使用 ndef.h 中的 NT3HwriteRecord()进行记录写入。

```
bool nfc_store_text(RecordPosEnu position, uint8_t *text)
{
    NDEFDataStr data;

    if (m_nfc_is_init == 0)
    {
        printf(" %s, %s, %d: NFC is not init!\n", __FILE__, __func__, __LINE__);
    }
}
```

```

        return 0;
    }

    prepareText(&data, position, text);
    return NT3HwriteRecord(&data);
}

bool nfc_store_uri_http(RecordPosEnu position, uint8_t * http)
{
    NDEFDataStr data;

    if (m_nfc_is_init == 0)
    {
        printf(" %s, %s, %d: NFC is not init!\n", __FILE__, __func__, __LINE__);
        return 0;
    }

    prepareUrihttp(&data, position, http);
    return NT3HwriteRecord(&data);
}

```

NT3HwriteRecord()负责将需要下发的信息打包成 NDEF 协议报文,最后由 I²C 总线将 NDEF 协议报文发送给 NFC 设备。

```

bool NT3HwriteRecord(const NDEFDataStr * data)
{
    uint8_t recordLength = 0, mbMe;
    UncompletePageStr addPage;
    addPage.page = 0;

    // calculate the last used page
    if (data->ndefPosition != NDEFFirstPos )
    {
        NT3HReadHeaderNfc(&recordLength, &mbMe);
        addPage.page = (recordLength + sizeof(NDEFHeaderStr) + 1) / NFC_PAGE_SIZE;

        addPage.usedBytes = (recordLength + sizeof(NDEFHeaderStr) + 1) % NFC_PAGE_SIZE - 1;
    }

    int16_t payloadPtr = addFuncnt[ data -> ndefPosition ] (&addPage, data, data ->
ndefPosition);
    if (payloadPtr == -1)
    {
        errNo = NT3HERROR_TYPE_NOT_SUPPORTED;
    }
}

```

```
        return false;
    }

    return writeUserPayload(payloadPtr, data, &addPage);
}
```

5.5.3 实验结果

程序编译烧写到开发板后,按下开发板的 RESET 按键,通过串口软件查看日志,具体内容如下:

```
===== NFC Example =====
Please use the mobile phone with NFC function close to the development board!
===== NFC Example =====
Please use the mobile phone with NFC function close to the development board!
...
```