1 IIC 实验

1.1 实验目的

使用 STM32F429 的普通 IO 口，用软件模拟 IIC 时序，实现和 24C02 之间的双向通信（读写），并将结果显示在 LCD 模块上。

1.2 实验原理

IIC(Inter－Integrated Circuit)总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。它是由数据线 SDA 和时钟 SCL 构成的串行总线，可发送和接收数据。在 CPU 与被控 IC 之间、IC 与 IC 之间进行双向传送，高速 IIC 总线一般可达 400kbps 以上。

I2C 总线在传送数据过程中共有三种类型信号。

- 开始信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。
- 结束信号：SCL 为高电平时，SDA 由低电平向高电平跳变，结束传送数据。
- 应答信号：接收数据的 IC 在接收到 8bit 数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据。CPU 向受控单元发出一个信号后，等待受控单元发出一个应答信号，CPU 接收到应答信号后，根据实际情况作出是否继续传递信号的判断。若未收到应答信号，由判断为受控单元出现故障。

这些信号中，起始信号是必需的，结束信号和应答信号是可选的。IIC 总线时序图如图所示：

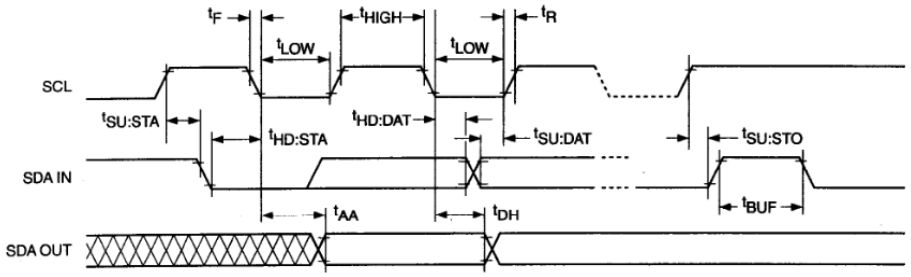

图 **1** **IIC** 总线时序图

目前大部分 MCU 都带有 IIC 总线接口，STM32F4 也不例外。但是这里我们不使用 STM32F4 的硬件 IIC 来读写 24C02，而是通过软件模拟。软件模拟最大的好处就是方便移植，同一个代码兼容所有 MCU，任何一个单片机。只要有 IO 口，就可以很快的移植过去，而且不需要特定的 IO 口。而硬件 IIC，则换一款 MCU，基本上就得重新搞一次，移植是比较麻烦的。

1.3 代码修改

未经修改的代码能够显示在按下不同按键时，通过 IIC 总线向 24C02 写入一段固定的字符串，并且将数据读出，显示在 LCD 模块上。

我们小组修改了程序逻辑，使其能够判断当前按下的按键，并将按键的值通过 IIC 总线读写 24C02，并显示在 LCD 模块上。

main.c 片段

```c
void itoa_1(u8 i, u8 *str) { str[0] = i + '0'; } // 一位整数转字符串
int main(void) {
  u8 key, datatemp[1], buffer[1];
  HAL_Init();                          // 初始化 HAL 库
  Stm32_Clock_Init(360, 25, 2, 8); // 设置时钟,180Mhz
  delay_init(180);                     // 初始化延时函数
  uart_init(115200);                   // 初始化 USART
  KEY_Init();                          // 初始化按键
  SDRAM_Init();                        // 初始化 SDRAM
  LCD_Init();                          // 初始化 LCD
  AT24CXX_Init();                      // 初始化 IIC
  POINT_COLOR = RED;
  LCD_ShowString(30, 130, 200, 16, 16, "KEYUP:Read  other:Write");
  POINT_COLOR = BLUE;
  while (1) {
    delay_ms(10);
    key = KEY_Scan(0);
    switch (key) {
    case 0:
      break;
    case WKUP_PRES:                          // WKUP 按下,读取字符串并显示
      LCD_Fill(0, 170, 239, 319, WHITE); // 清除半屏
      AT24CXX_Read(0, datatemp, sizeof(buffer)); // 读取 buffer 的值并显示
      LCD_ShowString(30, 190, 200, 16, 16, "The Data Readed Is: ");
      LCD_ShowString(30, 210, 200, 16, 16, buffer);
      break;
    default:
      LCD_Fill(0, 170, 239, 319, WHITE); // 清除半屏
      LCD_ShowString(30, 170, 200, 16, 16, "Start Write 24C02....");
      itoa_1(key, buffer); // 将 key 的值写入 buffer
      AT24CXX_Write(0, (u8 *)buffer, sizeof(buffer)); // 将 buffer 写入
      LCD_ShowString(30, 190, 200, 16, 16, "Write Finished!"); // 提示传送完成
      break;
    }
  }
}
```

函数 `itoa_1` 将一位整数转为字符，写入 buffer。这一位整数即为 `key.h` 中定义的按键值，例如 `#define KEY0_PRES 1`。

1.4 实验结果

按下 `WKUP_PRES` 按键，观察到开发板在 LCD 模块上显示按键值，初始时该值并不存在，因此不显示。而按下其他按键后，再次按下 `WKUP_PRES` 按键，观察到 LCD 模块上显示按键值，并且该值与按下按键的值一致，证明按键值写入与读取成功。

断电重启开发板，按下 `WKUP_PRES` 按键，观察到 LCD 上不显示按键值，证明 `24C02` 中的数据被清除，是易失性存储。

1.5 心得体会

这次实验，我们小组修改了 IIC 总线读写 24C02 的程序，并且成功将按键值写入 24C02，成功读出，并且亲身验证了其数据易失性，启发了我们在实际应用时需要做好断电预防措施。

2 SPI 实验

2.1 实验目的

使用 STM32F429 自带的 SPI 来实现对外部 FLASH（W25Q256）的读写，并将结果显示在 LCD 模块上，并通过一些方法验证数据切实写入了 Flash 中。

2.2 实验原理

SPI 是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便，正是出于这种简单易用的特性，现在越来越多的芯片集成了这种通信协议，STM32F4 也有 SPI 接口。SPI 的内部简图如下：

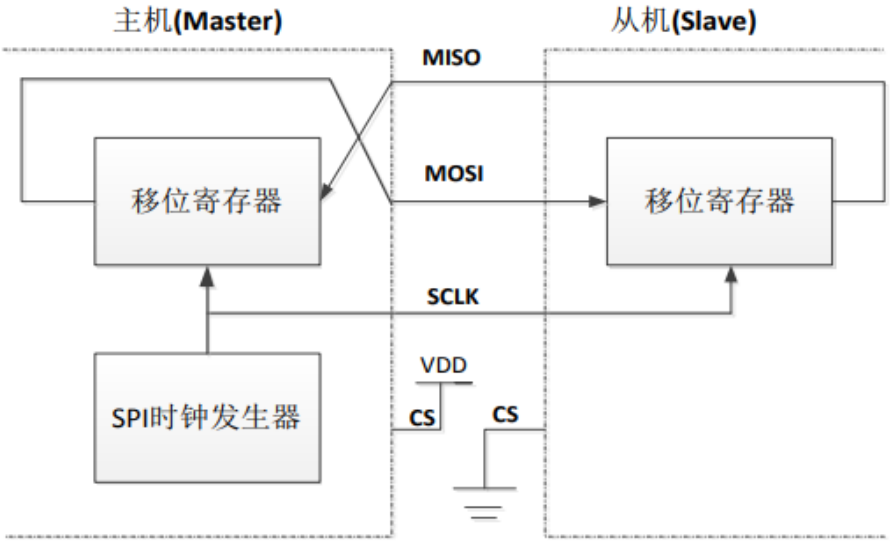

图 **2** **SPI** 内部结构简明图

其中，MISO 主设备数据输入，从设备数据输出。MOSI 主设备数据输出，从设备数据输入。SCLK 时钟信号，由主设备产生。CS 从设备片选信号，由主设备控制。STM32F429 的 SPI 功能很强大，SPI 时钟最高可以到 45Mhz，支持 DMA，可以配置为 SPI 协议或者 I2S 协议（支持全双工 I2S）。本次实验中，使用 STM32F429 的 SPI 来读取外部 SPI FLASH 芯片（W25Q256），使用了 STM32F429 的 SPI5 的主模式。

2.3 修改后的代码

本次实验的修改代码流程与 IIC 实验基本一致，我们小组也使用 `WKUP_PRES` 按键来控制 SPI 读操作，显示其数据在 LCD 模块上。其他按键进行写操作，将按键写入 SPI FLASH 芯片。

main.c 片段

```c
void itoa_1(u8 i, u8 *str) { str[0] = i + '0'; } // 一位整数转字符串
int main(void) {
  u8 key, datatemp[1];
  u32 FLASH_SIZE = 32 * 1024 * 1024; // FLASH 大小为 32M 字节
  HAL_Init();                        // 初始化 HAL 库
  Stm32_Clock_Init(360, 25, 2, 8);   // 设置时钟,180Mhz
  delay_init(180);                   // 初始化延时函数
  uart_init(115200);                 // 初始化 USART
  KEY_Init();                        // 初始化按键
  SDRAM_Init();                      // 初始化 SDRAM
  LCD_Init();                        // 初始化 LCD
  W25QXX_Init();                     // W25QXX 初始化
  POINT_COLOR = RED;
  LCD_ShowString(30, 130, 200, 16, 16, "KEYUP:Read  other:Write");
  POINT_COLOR = BLUE;
  while (1) {
    delay_ms(10);
    key = KEY_Scan(0);
    switch (key) {
    case 0:
      break;
    case WKUP_PRES:                        // WKUP 按下,读取字符串并显示
      LCD_Fill(0, 170, 239, 319, WHITE); // 清除半屏
      W25QXX_Read(datatemp, FLASH_SIZE - 100, sizeof(datatemp));
      LCD_ShowString(30, 190, 200, 16, 16, "The Data Readed Is: ");
      LCD_ShowString(30, 210, 200, 16, 16, datatemp);
      break;
    default:
      LCD_Fill(0, 170, 239, 319, WHITE); // 清除半屏
      LCD_ShowString(30, 170, 200, 16, 16, "Start Write....");
      itoa_1(key, datatemp); // 将 key 的值写入 datatemp
      W25QXX_Write((u8 *)datatemp, FLASH_SIZE - 100, sizeof(datatemp));
      LCD_ShowString(30, 190, 200, 16, 16, "Write Finished!"); // 提示传送完成
      break;
    }
  }
}
```

代码与 IIC 实验的代码主要不同在初始化阶段，使用 `W25QXX_Init` 初始化了不同的模块，并且在读写操作上使用了不同的函数。`W25QXX_Write` 比起 `AT24CXX_Write` 的参数发生了一些变化，需要指定从 FLASH 上读取的起始地址。

2.4 实验结果

按下 `WKUP_PRES` 按键，观察到开发板在 LCD 模块上显示按键值。而按下其他按键后，再次按下 `WKUP_PRES` 按键，观察到 LCD 模块上显示按键值，并且该值与按下按键的值一致，证明按键值写入与读取成功。

断电重启开发板，按下 `WKUP_PRES` 按键，观察到 LCD 上显示断电前的按键值，证明数据确实写入了 FLASH 芯片，属于非易失性存储。

2.5 心得体会

实验修改了 SPI 实验的代码，并且成功将按键值写入 SPI FLASH 芯片并读出，验证了其数据非易失性。

FLASH 在实际使用中有寿命的限制，W25Q128 的擦写周期大概有 10W 次，具有 20 年的数据保存期限。因此实际设计时需要考虑均匀在 SPI FLASH 芯片上写入数据，并且注意坏块检测与处理，以延长 FLASH 的使用寿命。