ESP8266 SPI透传协议(双线)

Version 0.1

Espressif Systems IOT Team

Table of Contents

1. 功能综述 4
2. ESP8266 SPI从机协议格式 4
   2.1. SPI从机时钟极性配置要求 4
   2.2. SPI从机支持的通信格式 4
3. 数据流控制线功能说明 5
   3.1. GPIO0主机发送从机接收缓存状态 5
   3.2. GPIO2主机接收从机发送缓存状态 5
   3.3. 主机通信逻辑实现 5
4. ESP8266 SPI从机API函数说明 7

1. 功能综述

该协议使用ESP8266的从机模式与其他的处理器的SPI主机进行通信，连线上需要6路信号线实现该协议，除了标准SPI所需要的4路信号线外，还需要额外的2路信号用于告知主机当前从机的接收与发送缓存状态以实现数据流控制。

2. ESP8266 SPI从机协议格式

2.1. SPI从机时钟极性配置要求

与ESP8266 SPI从机通信的主机设备时钟极性需配置为：空闲低电平，上升沿采样，下降沿变换数据。并且在一次34字节读/写过程中，务必保持片选信号CS的低电平，如果在发送过程中CS被拉高，从机内部状态将会重置。

2.2. SPI从机支持的通信格式

ESP8266SPI从机通信格式与主机模式基本相同为命令+地址+读/写数据，具体为：

(1) 命令：长度，8bits；主机输出从机输入（MOSI）。

其中0x02为主机发送从机接收数据，主机通过MOSI将32bytes写入从机数据缓存对应寄存器SPI_W0至SPI_W7；

而0x03为主机接收从机发送数据，将从机缓存对应寄存器SPI_W8至SPI_W15中的32bytes数据通过MISO发送到主机。

注意：其余数值用于读写SPI从机的状态寄存器SPI_STATUS，由于其通信格式与读写数据缓存不同，会造成从机读写错误，请勿使用。

(2) 地址：长度，8bits；主机输出从机输入（MOSI）。地址内容必须为0。

(3) 读/写数据：长度，256bits(32bytes)；主机输出从机输入（MOSI）对应0x02 命令或主机输入从机输出（MISO）对应0x03命令。

3. 数据流控制线功能说明

EPS8266使用两个GPIO输出从机接收和发送缓存的状态。

3.1. GPIO0主机发送从机接收缓存状态

GPIO0在进入从机接收中断后中断程序会：将SPI从机恢复到可通信状态准备下一次通信；然后把GPIO0写为低电平；再处理所接收到的数据；最后将GPIO0写为高电平退出中断程序。因此：

(1)在主机启动一次SPI写入通信到GPIO0产生下降沿期间，再次启动任何其他的SPI通信都会出错。

(2)在GPIO0低电平期间，主机启动任何SPI写入（0x02命令）操作会覆盖从机接收寄存器SPI_W0至SPI_W7。然而，如果此时从机发送寄存器准备好了有效数据（参见GPIO2说明），GPIO0低电平期间可以启动主机读取操作（0x03命令），将从机发送寄存器SPI_W8至SPI_W15中的数据读出。

(3)在GPIO0低电平跳转到高电平后，表示从机已经将接收寄存器SPI_W0至SPI_W7中的数据处理完毕，主机就可以再一次启动写入操作（0x02命令）。

3.2. GPIO2主机接收从机发送缓存状态

GPIO2与GPIO0动作略有区别。在进入从机发送中断后，中断程序会：将SPI从机恢复到可通信状态准备下一次通信；然后把GPIO0写为低电平；最后将退出中断程序。假如之后WIFI端有数据传入ESP8266并要求从SPI转发，则8266软件会写入SPI_W8至SPI_W15并将GPIO2置为高电平，因此：

(1)在主机启动一次SPI读取通信到GPIO2产生下降沿期间，再次启动任何其他的SPI通信都会出错。

(2)在GPIO2低电平期间，主机启动任何SPI读取（0x03命令）操作一般只能读出与上一次相同数据或写入不完整的数据。然而，如果此时从机接收寄存器数据处理完毕（参见GPIO0说明），GPIO2低电平期间可以启动主机写入操作（0x02命令）。

(3)在GPIO2低电平跳转到高电平后，表示从机已经将发送寄存器SPI_W8至SPI_W15中的数据更新，主机就可以再一次启动读取操作（0x03命令）。

3.3. 主机通信逻辑实现

下面以不完整C代码简要说明通信逻辑：

```
//wr_rdy变量表示可以进行下一次SPI写通信
//rd_rdy变量表示可以进行下一次SPI读通信
unsigned char wr_rdy=1,rd_rdy=0;
void spi_read_func(....)
```

```
{
        //在启动读传输之前，要判断从机是否有新数据可以读（即rd_rdy不为0）；
        //此外还要判断上次写入传输是否完毕：完毕正在处理数据(信号GPIO0为0)或者从机可以接受下一次写入 （wr_rdy不为0）
        if(rd_rdy&&((GPIO0= =0)||wr_rdy)){
         rd_rdy=0;       //清零可读标识 rd_rdy
        spi_transmit(0x03,0,*read_buff);//启动SPI传输,命令3+地址0+加32字节数据
        ….
        }
}
void spi_write_func(...)
{
        //在启动写传输之前，要判断从机是否可以接收新数据（即rd_rdy不为0）；
        //此外还要判断上次读取传输是否完毕：完毕无新数据需要读取(信号GPIO2为0)或者从机有新数据需要读取 （rd_rdy不为0）
        if(wr_rdy&&((GPIO2= =0)||rd_rdy)){
        wr_rdy=0;       //清零可写标识 rd_rdy
        spi_transmit(0x02,0,*write_buff);//启动SPI传输,命令2+地址0+加32字节数据
        ...
        }
}

GPIO0_Raising_Edge_ISR() //与8266的GPIO0相连的上升沿中断程序
{
         wr_rdy=1;      //主机发送数据处理完毕可以进行下一次写入
}

GPIO2_Raising_Edge_ISR()//与8266的GPIO2相连的上升沿中断程序
{
         rd_rdy=1; //从机更新发送缓冲，主机可以读取
}
```

4. ESP8266 SPI从机API函数说明

(1) void spi_slave_init(uint8 spi_no)

功能：SPI从机模式初始化，将IO口配置为SPI模式，启用SPI传输中断，并注册 spi_slave_isr_handler函数。通信格式设定为 8bits命令+8bit地址+256bits(32bytes)读/写数据。

参数：

spi_no: SPI模块的序号，ESP8266处理器有两组功能相同的SPI模块，分别为SPI和HSPI

可选配的值: SPI或HSPI

(2) spi_slave_isr_handler(void *para)

功能与触发条件：spi中断处理函数，主机如正确进行了传输操作（读或写从机），中断就会触发。用户可以修改中断服务程序实现所需通信功能，代码如下：

代码：

```
    uint32 regvalue;
    static uint32 t1 =0;
    static uint32 t2 =0;
    t1=system_get_time();

    if(READ_PERI_REG(0x3ff00020)&BIT4){                //bit4表示SPI中断
  //following 3 lines is to clear isr signal
    CLEAR_PERI_REG_MASK(SPI_SLAVE(SPI), 0x3ff);
    }else if(READ_PERI_REG(0x3ff00020)&BIT7){ //bit7 表示HSPI中断,
    regvalue=READ_PERI_REG(SPI_SLAVE(HSPI));       // 记录中断类型
        // 关闭SPI中断使能
    CLEAR_PERI_REG_MASK(SPI_SLAVE(HSPI),
                                                    SPI_TRANS_DONE_EN|
                                                    SPI_SLV_WR_STA_DONE_EN|
                                                    SPI_SLV_RD_STA_DONE_EN|
                                                    SPI_SLV_WR_BUF_DONE_EN|
                                                    SPI_SLV_RD_BUF_DONE_EN);
        // 将SPI从机恢复到可通信状态，准备下一次通信
    SET_PERI_REG_MASK(SPI_SLAVE(HSPI), SPI_SYNC_RESET);
        // 清除中断标志
    CLEAR_PERI_REG_MASK(SPI_SLAVE(HSPI),
                                                    SPI_TRANS_DONE|
                                                    SPI_SLV_WR_STA_DONE|
```

```
                                                    SPI_SLV_RD_STA_DONE|
                                                    SPI_SLV_WR_BUF_DONE|
                                                    SPI_SLV_RD_BUF_DONE);
            // 打开SPI中断使能
            SET_PERI_REG_MASK(SPI_SLAVE(HSPI),
                                                    SPI_TRANS_DONE_EN|
                                                    SPI_SLV_WR_STA_DONE_EN|
                                                    SPI_SLV_RD_STA_DONE_EN|
                                                    SPI_SLV_WR_BUF_DONE_EN|
                                                    SPI_SLV_RD_BUF_DONE_EN);
            //主机写入，从机接收处理程序
             if(regvalue&SPI_SLV_WR_BUF_DONE){
            GPIO_OUTPUT_SET(0, 0);     //GPIO0清0
            idx=0;
            //读取接收数据
                    while(idx<8){
                    recv_data=READ_PERI_REG(SPI_W0(HSPI)+4*idx);
                    //os_printf("rcv data : 0x%x \n\r",recv_data);
                    spi_data[4*idx+0] = recv_data&0xff;
                    spi_data[4*idx+1] = (recv_data>>8)&0xff;
                    spi_data[4*idx+2] = (recv_data>>16)&0xff;
                    spi_data[4*idx+3] = (recv_data>>24)&0xff;
                    idx++;
                    }
            system_os_post(USER_TASK_PRIO_1,MOSI,0);//投送接收完成消息
            GPIO_OUTPUT_SET(0, 1);                                  //GPIO0置1
            SET_PERI_REG_MASK(SPI_SLAVE(HSPI),  SPI_SLV_WR_BUF_DONE_EN);
            //主机读取，从机发送处理程序
         if(regvalue&SPI_SLV_RD_BUF_DONE){
                    GPIO_OUTPUT_SET(2, 0);     //GPIO2清0
        }
    }else if(READ_PERI_REG(0x3ff00020)&BIT9){ //bit7 表示I2S中断

    }
```