ESP8266 SDIO通信SPI兼容模式

说明手册

Version 0.1

Espressif Systems IOT Team

Table of Contents

1. 功能综述与原理

该协议使用ESP8266的SDIO模块与其他的处理器的SPI主机进行通信。在电气接口方面，协议通过4路信号线实现，包括SPI协议中的SCLK，MOSI，MISO（注意没有CS信号）与1路中断信号。

要使用EPS8266SDIO通信，其程序下载方式与常规情况有所不同。由于ESP8266启动时，默认读取程序的SPI接口与SDIO接口复用相同的芯片管脚，因此要使用SDIO模块通信协议，ESP8266需要以SDIO模式下启动，随后主机通过SDIO下载部分的程序到ESP8266的RAM中以启动芯片，而大部分直接由CPU CACHE调用FLASH的程序则可以事先用烧录工具写入与HSPI接口相连接的FLASH芯片中。

ESP8266SDIO的接收发送的数据直接由内部支持链表检索的DMA模块操作。

8266可以不通过CPU参与，高效的通过内存映射链表的地址完成SDIO数据包的收发。

2. DEMO实现方案

2.1. 平台介绍

通信主机端是以STM32F103ZET6为核心的红龙开发板，软件由IAR平台开发使用FreeRTOS操作系统。从机为ESP_IOT Reference board，基于v0.9.3的SDK开发。

2.2. ESP8266软件编译与下载准备

- 将ESP8266的DEMO工程在SDIO communication demo\ esp_iot_sdk_v0.9.3_sdio_demo \app通过编译器编译并生成下载使用的bin文件。
- 注意SDIO communication demo\esp_iot_sdk_v0.9.3_sdio_demo\lib中的libmain.a与v0.9.3 release版本中的不同，如果使用 release版本的SDK，需要用DEMO中的libmain.a替换掉原来的，修改版的libmain.a会使芯片启动后，将读取FLASH程序的SPI模块与原HSPI的映射管脚互相交换。使用DEMO工程直接编译生成即可。
- 将SDIO communication demo\esp_iot_sdk_v0.9.3_sdio_demo\bin中的eagle.app.v6.irom0text.bin复制到SDIO communication demo\XTCOM_UTIL目录下，eagle.app.v6.irom0text.bin为8266程序中由CPU CACHE直接通过SPI读取FLASH芯片的所有函数。
- 运行SDIO communication demo\中的BinToArray.exe选择SDIO communication demo\esp_iot_sdk_v0.9.3_sdio_demo\bin中的eagle.app.v6.flash.bin将其转换为ANSI C格式的数组。转换后的文件会保存在D:\中，BinToArray.exe的目标路径一定是D盘不能修改……如果没有D盘，只能用有D盘的虚拟机….或连个U盘名叫D….或分一个D盘….或者直接去网上找个bin转数组的工具….
- 假如电脑有D盘，把D:\中的hexarray.c以eagle_fw.h命名，并且把数组名定义为const unsigned char eagle_fw[] =……，替换掉SDIO communication demo\STM32\ Eagle_Wifi_Driver\ egl_drv_simulation\中的eagle_fw.h，（可以先从老的eagle_fw.h中复制出数组名和文件名，修改

hexarray.c后替换掉老的eagle_fw.h）。eagle.app.v6.flash.bin是芯片在启动前先要载入8266内存的部分，这里需要转成数组，通过STM32写入8266。

- 用IAR平台打开SDIO communication demo\STM32\IAR\中的EglWB.ewp.eww，编译工程。

2.3. ESP8266 FLASH端软件下载

- 用串口线连接ESP_IOT Reference board和电脑，连上5V电源。板子上J67连接右边的两针（使能HSPI接口上的FLASH芯片），J66连接左边的两针（断开SPI接口上的FLASH芯片）。 将MTD0,GPIO0,GPIO2 这3个跳线设置 为: 0,0,1 （上，上，下）的UART模式。
- 双击SDIO communication demo\XTCOM_UTIL目录下XTCOM_UTIL.exe, 点击Tools -> Config Device 选择Com口， Baud Rate: 115200. 点击Open，出现open Success. 点击Connect, 然后按H Flash板电源，出现连接成功。
- 点击API TEST(A)->(5) HSpiFlash Image Download, 选择SDIO communication demo\XTCOM_UTIL目录下的eagle.app.v6.irom0text.bin然后Offset: 0x40000, 点击DownLoad，下载完成。

2.4. ESP8266 FLASH端软件下载

使用排线针连接ESP_IOT Reference board和红龙开发板具体连线如下：

红龙开发板JP1中:

ESP_IOT板中J62排针从下往上数

GND	->	1	VSS/GND
SPI_CLK	->	4	SDIO_CLK
SPI_MOSI	->	5	SDIO_CMD
SPI_MISO	->	3	SDIO_DAT0
IRQ	->	2	SDIO_DAT1

ESP_IOT Reference board上:将跳线MTD0换到1(下方两针短接), GPIO0，GPIO2任意（1，x，x为SDIO启动模式） CHIP_PD:ON （开关拨在下方）. 保持跳线J66 连接左边两针， 跳线J67 连接右边两针。

5V电源适配器连接ESP_IOT Reference board和红龙开发板。打开红龙开发板电源，在IAR环境中将之前2.2节中编译完成的工程下载到STM32中。启动STM32程序，打开ESP_IOT Reference board电源。STM32会先向ESP8266写入启动程序，几秒后自动运行SDIO返回测试程序。

3. ESP8266端软件说明

3.1. 协议原理：SDIO中断线行为与SDIO状态寄存器

ESP8266的SD_DATA1管脚在SDIO运行于SPI兼容模式时作为通知SPI主机的中断线，且为低电平有效。当8266中SDIO状态寄存器被软件更新时，中断线由高电平变为低电平，需要主机通过SDIO写入恢复中断线。（具体为：主机需要通过CMD53或52命令向地址为0x30的寄存器写入1来使中断线恢复高电平。）

SDIO状态寄存器为32bits，由8266软件修改，并可以主机由CMD53或52命令读取，地址为0x20-0x23，其数据结构被定义为：

```
struct sdio_slave_status_element
{
        u32 wr_busy:1;
        u32 rd_empty :1;
        u32 comm_cnt :3;
        u32 intr_no :3;
        u32 rx_length:16;
        u32 res:8;
};
```

其中：

(1) wr_busy，bit0：1表示从机写缓存满，8266正在处理主机发送的数据，0表示写缓存空可以进行一次写入操作。

(2) rd_empty，bit1：1表示从机读缓存为空，没有新数据更新，0表示读缓存有新数据需要主机读取。

(3) comm_cnt，bit2-4：读写通信计数。每次8266 SDIO模块完成一次有效读/写数据包操作时，计数值会加1，主机可以由此判断一次读/写数据通信是否已经被8266有效响应。

(4) intr_no，bit5-7：协议最终未使用该变量，保留。

(5) rx_length，bit8-23：读缓存中，所准备数据包的实际长度。

(6) res， bit24-31：保留。

因此，主机通信流程为：

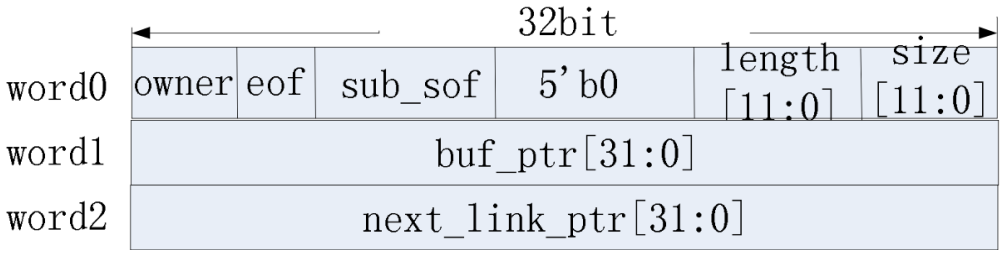
(1) 在收到中断请求后，先读取SDIO状态寄存器，再清除中断，并根据状寄存器来进行读写数据包的操作。

(2) 定时轮询SDIO状态寄存器，根据状态寄存器来进行读写数据包的操作。

3.2. 读写缓存与注册链表的使用说明

ESP8266的SDIO收发数据包直接会被DMA传输到对应的内存。8266软件中会定义链表注册结构体（或数组）以及一个（或多个）缓存空间，本例中只使用一个缓存空间链表也只有一个元素。将缓存的首地址写入链表注册结构体，并完成其他信息，在将链表结构体首地址写入8266对应硬件寄存器，DMA就能自动操作SDIO与缓存空间。

表注册结构体具体为：



(1) owner：1'b0:当前link对应buffer的操作者为SW；当前link对应buffer的操作者。MAC不使用该bit。1'b1:当前link对应buffer的操作者为HW；

(2) eof：帧结束标志（对于AMPDU的子帧结束，该标识是不会置上的）。在mac发送帧时用于指示帧结束link。对于eof位置上的link，它的buffer_length[11:0]必须等于该帧剩下的长度，否者mac会上报error。在mac接收帧时用于指示帧已接收完成。此时该值由硬件置上。

(3) sub_sof：子帧起始link标识，区分AMPDU帧内的不同子帧，仅用于mac发送时。

(4) length[11:0]：buffer实际占用的大小。

(5) size[11:0]：buffer的总大小。

(6) buf_ptr[31:0]：buffer的起始地址。

(7) next_link_ptr[31:0]：下1个discripter的起始地址。在mac接收帧时该值为0，表示已无空buffer用于接收。

3.3. ESP8266 DEMO中提供的API函数

(1) void sdio_slave_init(void)

功能：sdio模块初始化，其中包括状态寄存器初始化，RX和TX注册链表初始化，通信中断线的模式配置，收发中断的配置与注册中断服务程序等。

(2) void sdio_slave_isr(void *para)

功能与触发条件：sdio中断处理函数，sdio正确接收或发送了一个数据包后就会触发该中断程序。在DEMO中，8266所有的测试操作都在中断处理函数中完成。通信过程中所有对注册链表，状态寄存器，数据处理都可以在该函数中找到。

(3) void rx_buff_load_done(uint16 rx_len)

功能：rx_buffer装入新数据包后，必须调用该函数使新数据变为待读取状态。该函数包含了注册链表软硬件相关操作，与 状态寄存器相关操作。DEMO中该函数在中断服务程序中被调用。

参数： rx_len 新数据包实际长度，以字节为单位。

(4) void tx_buff_handle_done(void)

功能：tx_buffer中数据处理完成后，必须调用该函数使sdio变为可发送状态已准备接受下个数据包。该函数包含了注册链表软硬件相关操作，与 状态寄存器相关操作。DEMO中该函数在中断服务程序中被调用。

(5) void rx_buff_read_done(void)

功能：rx_buffe中数据被读取后，必须调用该函数使sdio变为不可读状态。该函数包含了状态寄存器相关操作。函数应在RX_EOF中断服务的一开始调用。

(6) void tx_buff_write_done(void)

功能：tx_buffer收到新数据包时，必须调用该函数sdio变为不可写状态。该函数包含了状态寄存器相关操作。函数应在TX_EOF中断服务的一开始调用。

(7) TRIG_TOHOST_INT()

功能：宏，将通信中断线拉低，通知主机。

其他函数为测试使用。

4. STM32端软件说明

4.1. 主要函数说明

(1) void SdioRW(void *pvParameters)

功能：Sdio测试线程，其中包含了所有读写的操作流程。

位置：egl_thread.c中由同文件中的函数SPITest()中注册。

参数：未使用。

(2) int esp_sdio_probe(void)

功能：8266启动程序下载相关程序。

位置：esp_main_sim.c中，由egl_thread.c中的函数SPITest()调用。

(3) int sif_spi_write_bytes(u32 addr, u8*src,u16 count,u8 func)

功能：sdio byte模式写入API，封装了CMD53 byte模式写入功能，可以对寄存器或数据包进行操作。sdio协议规定最大有效数据长度是512字节。

位置：port_spi.c中，由egl_thread.c中的函数SdioRW 调用。

参数：

src：发送数据包首地址。

count：发送长度。字节为单位

func： 功能号，目前除了使用修改sdio CMD53的block模式中的block_size大小通信用的是0，其余所有通信都使用1。

addr：写入的目标首地址。如果操作寄存器直接输入对应地址，如0x30中断线清除寄存器，0x110修改block_size(func为0)；注意如果操作数据包，要输入0x1f800-tx_length的数值，且tx_length与count相等，如果count>tx_length，SPI主机会发送count长度的数据包，但是从第tx_length+1到count的数据会被8266的sdio模块丢弃。因此，在发送数据包是addr关系到实际传输有效数据的长度。

(4) int sif_spi_read_bytes(u32 addr,u8* dst,u16 count,u8 func)

功能：sdio byte模式读取API，封装了CMD53 byte模式读取功能，可以对寄存器或数据包进行操作。sdio协议规定最大有效数据长度是512字节。

位置：port_spi.c中，由egl_thread.c中的函数SdioRW 调用。

参数：

dst：接收缓存首地址。

count：接收长度。字节为单位

func：功能号，目前除了使用读取sdio CMD53的block模式中的block_size大小通信用的是0，其余所有通信都使用1。

addr： 读取的目标首地址。如果操作寄存器直接输入对应地址，如0x20为SDIO状态寄存器；注意如果操作数据包，要输入0x1f800-tx_length的数值，且tx_length与count相等，如果count>tx_length，SPI主机会读取count长度的数据包，但是从第tx_length+1到count的数据会被8266的sdio模块丢弃读出无效数据。因此，在发送数据包是addr关系到实际传输有效数据的长度。

(5) int sif_spi_write_blocks(u32 addr, u8 * src, u16 count,u16 block_size)

功能：sdio block模式写入API，封装了CMD53 block模式写入功能，只能传输数据包。sdio协议规定最大有效数据长度是512个block。

位置：port_spi.c中，由egl_thread.c中的函数SdioRW 和esp_main_sim.c中程序下载器所使用的函数sif_io_sync调用。

参数：

src：发送数据包首地址。

count：发送长度。以block为单位

block_size：1个block有多少字节，注意必须与func为0 addr为0x110-111中的16bit值相同。一般在sdio初始化时需要配置8266 sdio的block_size。DEMO的初始值为512。在运行过程中，有配置为1024。block_size必须为4的整数倍

addr：写入的目标首地址。与byte模式相同输入0x1f800-tx_length的数值，且tx_length与count相等。

(6) int sif_spi_read_blocks(u32 addr, u8 *dst, u16 count,u16 block_size)

功能：sdio block模式写入API，封装了CMD53 block模式写入功能，只能传输数据包。sdio协议规定最大有效数据长度是512个block。

位置：port_spi.c中，由egl_thread.c中的函数SdioRW 和esp_main_sim.c中程序下载器所使用的函数sif_io_sync调用。

参数：

src：接收缓存首地址。

count：接收长度。以block为单位

block_size： 一个block有多少字节，注意必须与func为0 addr为0x110-111中的16bit值相同。一般在sdio初始化时需要配置8266 sdio的block_size。DEMO的初始值为512。在运行过程中，有配置为1024。block_size必须为4的整数倍

addr： 读取的目标首地址。与byte模式相同输入0x1f800-tx_length的数值，且tx_length与count相等。

(7) void EXTI9_5_IRQHandler(void)

功能：通信中断处理函数为线程函数SdioRW 中的函数egl_arch_sem_wait (&BusIrqReadSem,1000)提供使能信号，使SdioRW线程跳出等待读取sdio状态寄存器。

位置：spi_cfg.c