SPI 驱动编写指导手册

编写：贺敏 2015 年 3 月 23 日
Review：_年_月_日
审阅:罗侍田

1. 贡献者列表

DJYOS 开发团队。

2. 概述

DJYOS 的 DjyBus 总线模型为 IIC、SPI 之类的器件提供统一的访问接口，SPIBUS 模块是 DjyBus 模块的一个子模块，为 SPI 器件提供统一的编程接口，实现通信协议层与器件层的分离。也标准化了 SPI 总线和 Device 驱动接口，本手册指导驱动工程师编写 SPI 的接口程序。

SPI 总线使用手册，请参见《都江堰操作系统用户手册》。

局限性：DJYOSV1.1.1 版本的 SPI 驱动只提供主设备功能。

3. 总线资源结构

SPI 通信协议是一种总线通信方式，这意味着一条总线上可以挂多个符合总线通信协议的设备，DjyBus 资源组织结构就是符合这样一种物理的连接方式。如图 2-1所示资源组织结构图，总线类型“SPI”、第 n 条总线“SPIn”、第 n 条总线上面的设备“Devn”，它们都是 DjyBus 资源树上面的资源节点，每次向总线“SPIn”上面增加一个设备，便向资源树上面增加了一个资源节点，它是“SPIn”的子节点。

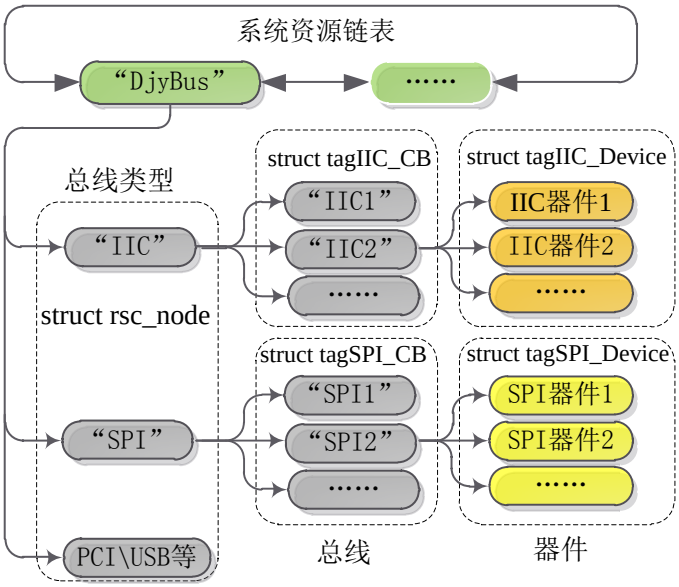

图 2-1 总线资源结构

4. 准备工作

在编写 SPI 器件驱动前，建议完成必要的准备工作，如：

1、 认真阅读器件手册，了解通信协议、参数、操作流程等内容；

2、 熟悉 spibus.h 头文件中提供的 API，懂得参数的使用方法；

3、 阅读 SPI 总线协议文档，熟练掌握 SPI 总线。

5. SPI 总线驱动接口

5.1. 驱动架构

SPIBus 是 DjyBus 模块的一个子模块，其结构如图 4-1所示，它为 SPI 器件提供标准的、一致的应用程序编程接口，并且规范了硬件驱动接口。驱动接口分为总线控制器接口和 SPI 器件接口两部分，驱动的重点是总线控制器，而器件接口实际上就是配置一下该器件的物理参数。

建议文件路径：在 eclipse 工程中的链接目录如下，如果是导入官方提供的 example 工程，那么该目录已经建立，在硬盘中添加文件后，只需要刷新工程即会自动添加进工程中。

src->OS_code->bsp->cpudrv->src->cpu_peri_spi.c。

相应的头文件目录为：

src->OS_code->bsp->cpudrv-> src->cpu_peri_spi.h。

在文件系统（硬盘）中的目录结构是：

djysrc\bsp\cpudrv\cpu_name\src\cpu_peri_spi.c。

djysrc\bsp\cpudrv\cpu_name\include\cpu_peri_spi.h。

根据以上命名，可以在 DJYOS 官方提供的代码中，找到大量范例。

以上文件命名并非绝对，例如 LPC17xx 的 SPI 模块，硬件被官方命名为 SSP 模块，DJYOS 提供的源码中，其文件名就命名为 cpu_peri_ssp.c。

SPI 驱动程序编写重点有：

1、 初始化 SPI 控制器，并且把 SPI 总线添加到 DjyBus 上。

2、 实现图 4-1中的 5 个回调函数（哪些需要实现，参考后续章节）。

3、 如果采用中断方式，须编写中断服务函数（实际上也是为 4 个回调函数服务）

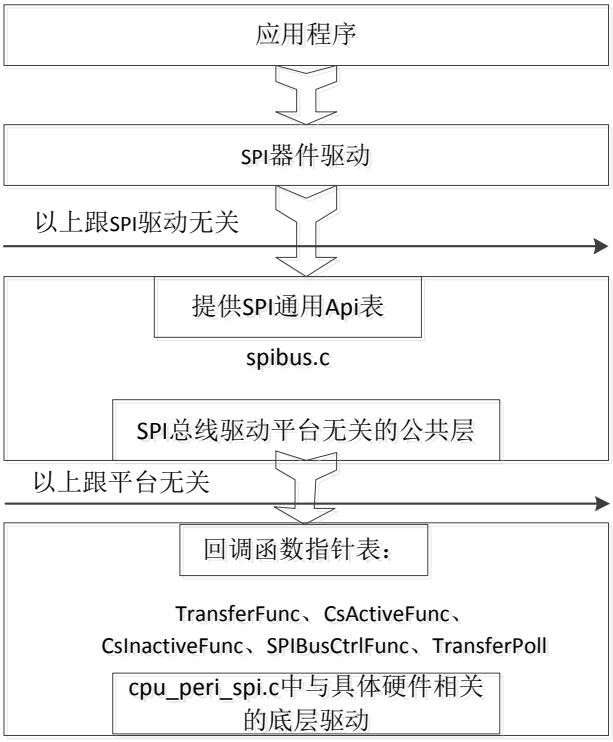

图 4-1 SPIBus 总线驱动架构

5.2. 初始化函数

5.2.1. Step1：初始化硬件

1、 SPI 控制器硬件的初始化，包括传输速度、IO 配置、时钟等；
2、 挂载 SPI 中断到中断系统，并配置中断类型，如配置为异步信号（若只采用轮询方式，则此功能可省略）；

5.2.2. Step2：初始化参数结构体

添加 SPI 总线的参数类型为 struct tagSPI_Param，由函数 SPI_BusAdd 或 SPI_BusAdd_s 完成对 SPI 总线控制块的初始化和添加 SPIn 总线节点到 DjyBus 资源树。

代码 4-1 SPI 参数结构体定义

```
struct tagSPI_Param
{
    char           *BusName;            //总线名称，如SPI1
    u8             *SPIBuf;             //总线缓冲区指针
    u32             SPIBufLen;          //总线缓冲区大小，字节
    ptu32_t         SpecificFlag;       //SPI私有标签，如控制寄存器基址
    bool_t          MultiCSRegFlag;     //SPI控制寄存器是否有多套CS配置寄存器
    TransferFunc    pTransferTxRx;      //发送接收回调函数，中断方式
    TransferPoll    pTransferPoll;      //发送接收回调函数，轮询方式
    CsActiveFunc    pCsActive;          //片选使能
    CsInActiveFunc  pCsInActive;        //片选失能
    SPIBusCtrlFunc  pBusCtrl;           //控制函数
};
```

SPI 主设备同一时刻只能与一个从设备通信，收发同时进行，因此同一个 SPI 控制器中，多个片选可以共用缓冲区。

很多的 SPI 控制器对每个片选都提供一套配置通信参数寄存器，例如，CS0 与从设备通信采用速度 5Mbit/s，字符宽度为 8 比特，MSB，对应的配置片选 CS0 对应的寄存器，而 CS1 的从设备采用速度 10Mbit/s，字符宽度为 16 比特，LSB，对应的配置片选 CS1 对应的寄存器。这种增强型的控制器对于一主多从，参数不一的通信能大大提高通信效率，简化参数配置。

SPI 参数结构体的回调函数参数的原型如代码 4-2所示，其中 PrivateTag 就是结构体中 SPI 的私有标签，即 SPIn 寄存器基址。

代码 4-2 SPI 回调函数类型申明

```
typedef ptu32_t (*TransferFunc)(ptu32_t SpecificFlag,u32 sendlen,u32
                                recvlen,u32 recvoff);
typedef bool_t (*TransferPoll)(ptu32_t SpecificFlag,u8* srcaddr,u32
                               sendlen,u8* destaddr,u32 recvlen,u32 recvoff);
typedef bool_t (*CsActiveFunc)(ptu32_t SpecificFlag, u8 cs);
typedef bool_t (*CsInActiveFunc)(ptu32_t SpecificFlag, u8 cs);
typedef ptu32_t (*SPIBusCtrlFunc)(ptu32_t SpecificFlag,u32 cmd,ptu32_t
                                  data1,ptu32_t data2);
```

5.2.3. Step3：挂载总线

有多少 SPI 总线是由具体的平台决定，因此，增加 SPI 总线到 DjyBus 上是由总线驱动程序员完成，成功添加的“SPIn”节点会成为“SPI”节点的子节点。

增加 SPI 总线的 API 函数可以调用 SPI_BusAdd 函数或 SPI_BusAdd_s 函数，两者的区别在于，SPI_BusAdd 只需调用者提供已初始化好的参数结构体 struct tagSPI_Param，而后者更需要提供 struct tagSPI_CB 结构体控制块（建议定义为静态变量）。

5.3. 回调函数

5.3.1. 轮询函数

如果采用轮询方式收发，5 个回调函数中只需要实现这一个，其他指针置为 NULL 即可。

轮询函数使用场合：

1、 收发方式被设为轮询方式，则总是用轮询函数收发数据。默认值为中断方式，可调用 SPI_BusCtrl 函数设为轮询方式。

2、 在禁止调度（即禁止异步信号中断）期间，强制使用轮询方式。

3、 pTransferTxRx ==NULL，则使用轮询方式收发。

4、 系统初始化未完成，多事件调度尚未启动期间。

如果使用中断方式收发，且不考虑在 2~4 三种情况下收发数据，则无须实现本函数，pTransferPoll 指针设为 NULL 即可。

回调函数说明如下：

```
typedef bool_t (*TransferPoll)(ptu32_t SpecificFlag,u8* srcaddr,u32 sendlen,
                               u8* destaddr,u32 recvlen,u32 recvoff);
```

参数：

SpecificFlag：寄存器基址

srcaddr：发送数据存储地址。

sendlen：发送数据字节数。

destaddr：接收数据存储地址。

recvlen：接收数据字节数。

recvoff：接收偏移字节数，即认为接收到 recvoff 字节后的数据为有效，才存储。

返回：true，执行成功；false，执行失败。

说明：轮询方式主要应用于系统调度未启动或对实时性要求不高的场合，这种方法能够简化编程处理，快速实现通信功能。

5.3.2. 启动收发

启动收发是在使用中断方式收发数据必须实现的回调函数，若使用轮询方式，无须实现本函数，本函数指针设置为 NULL 即可。

回调函数说明如下：

```
typedef ptu32_t (*TransferFunc)(ptu32_t SpecificFlag,u32 sendlen,
                                u32 recvlen,u32 recvoff);
```

参数：

SpecificFlag，寄存器基址。

sendlen，发送数据长度，字节单位。

recvlen，接收数据长度，字节单位。

recvoff，接收偏移，接收到多少个字节后开始保护数据，即有用数据。

返回：true,中断方式启动通信成功，false，失败。

说明：该函数功能是配置 SPI 寄存器，并保存有关参数，使能中断，SPI 总线采用的是四线的收发方式，收、发、时钟和片选。TransferFunc 实质上是实现了相关的寄存器的配置，并中断使能。当然，SPI 总线协议规定，操作设备前必须把对应从设备的 CS 线拉低。

对于 TransferFunc 需要完成的功能作如下说明：

1、 保存静态变量，如发送接收数据长度，接收偏移（从接收到的第几个数据开始保存数据）；

2、 配置 SPI 寄存器，使其处于发送和接收的状态；

3、 配置中断使能，并触发中断，在中断中将数据发送接收完成。

5.3.3. 片选使能

若使用轮询方式，本函数指针设置为NULL即可，但使能片选的功能须实现。

```
typedef bool_t (*CsActiveFunc)(ptu32_t SpecificFlag, u8 cs);
```

功能：片选拉低

参数：

SpecificFlag，寄存器基址

cs, 片选号

返回：是否成功

虽然对于具体的芯片，该函数的实现过程不相同，但是功能是相同的，即拉低片选，选择 CS 对应的 SPI 从器件通信。若控制器具有硬件自动片选，硬件驱动可加以利用，提高效率。

5.3.4. 片选失能

若使用轮询方式，本函数指针设置为NULL即可，但失能片选的功能须实现。

```
typedef bool_t (*CsInActiveFunc)(ptu32_t SpecificFlag, u8 cs);
```

功能：片选拉高

参数：

SpecificFlag，寄存器基址

cs, 片选号

返回：是否成功

5.3.5. 控制函数

目前，控制函数主要实现对总线的配置，如自动片选、传输速度、SPI 时序配置等。应用层将通过调用 SPI_Ctrl 接口函数，传递不同的命令和参数，实现对总线的控制。

如果 SPI 控制器针对每个片选信号都有独立的配置寄存器，则在添加设备时（SPI_DevAddr），配置好每个片选寄存器；若多个片选共用一套配置寄存器，则每次传输都必须重新配置。在结构体 SPI_CB 中，成员 multi_cs_reg 是用来标记 SPI 控制器是否具有多套片选寄存器，在调用 ModuleInstall_SPI 时，硬件驱动会作相应的标记。

从 SPI 协议的时序来讲配置参数会更加的清晰。如图 4-2和图 4-3所示，SPI 通信首先需产生 CS 片选有效，即拉低对应的 CS 片选。时钟信号 SPI_CLK 在未通信状态时的电平状态由 CHOL 决定，为高或者为低。而 CPHA 决定时序的相位，当 CPHA 为 0 时，在 SPI_CLK 的第一个边沿采样，第二个边沿输出数据；当 CPHA 为 1 时，在 SPI_CLK 的第一个边沿输出数据，第二个边沿采样。

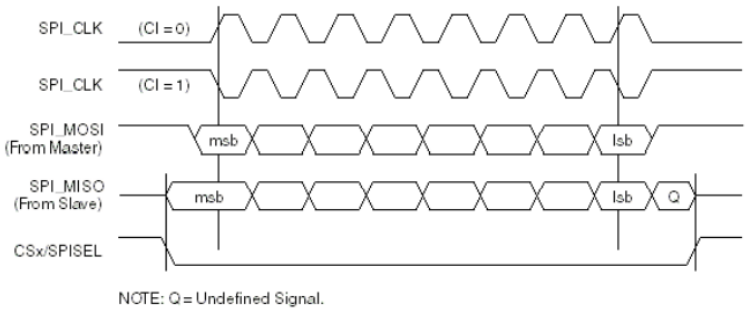

图 4-2 SPI 时序 CPHA=0

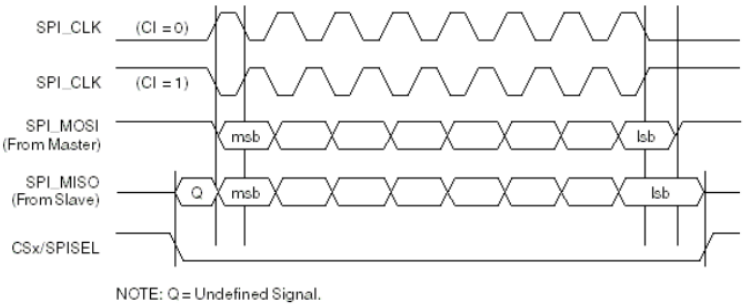

图 4-3 SPI 时序 CPHA=1

CHOL 和 CPHA 两种配置组成了四种模式，分别为模式 0、1、2、3，如表 4-1所示，部分 SPI 从器件只支持部分模式，需根据具体器件配置成要求的模式。

表 4-1 SPI 模式

CHOL	CPHA	MODE
0	0	0
0	1	1
1	0	2
1	1	3

控制命令用于应用层调用 spibus.h 的 API 的控制函数 SPI_Ctrl 实现参数配置或通信设置，与硬件相关的命令一览表如表 4-2所示。

表 4-2 SPI 命令表

命令	参数 1	参数 2	说明
CN_SPI_SET_CLK	速率，bit/s	无意义	设置传输速度
CN_SPI_CS_CONFIG	tagSpiConfig	无意义	配置相关参数
CN_SPI_SET_AUTO_CS_EN	无意义	无意义	自动片选使能
CN_SPI_SET_AUTO_CS_DIS	无意义	无意义	自动片选失能

5.4. 中断服务函数

5.4.1. 功能描述

如果使用轮询方式实现驱动，则无须编写中断服务函数。

SPI 接收和发送使用中断方式的好处在于，将发送任务由 SPI 控制器完成，节省 CPU 的处理负荷，因此提高了程序的运行效率，缺点在于编程相对复杂。现在绝大多数的主流 CPU 的中断系统都支持 SPI 中断，包括发送接收中断等。

SPI 模块要求在中断服务函数内部完成的功能有如下：

1、 清中断标志，处理好接收与发送数据同时进行的硬件机制；

2、 接收数据从接收到 recvoff 字符数据后开始存储，即调用 SPI_PortWrite；

3、 发送数据从 SPI_PortRead 读取，若没有读到数据，则代表数据已经发送完成；若此时接收的数据还未完成，应该继续往寄存器中写数据，直到接收完成；

4、 数据传输完成时，配置相应的寄存器，使其处于初始状态（视控制器而定）；

5.4.2. 中断实现过程

下面以 Atmel 芯片为例，通过流程图的方式简要说明 SPI 中断服务函数的数据处理流程图。值得注意的是，中断服务函数中有些变量是通过__SPI_TransferTxRx 传递参数到底层硬件驱动，底层驱动通过静态变量存储，并在中断服务函数中使用。如发送接收数据大小，信号量等。

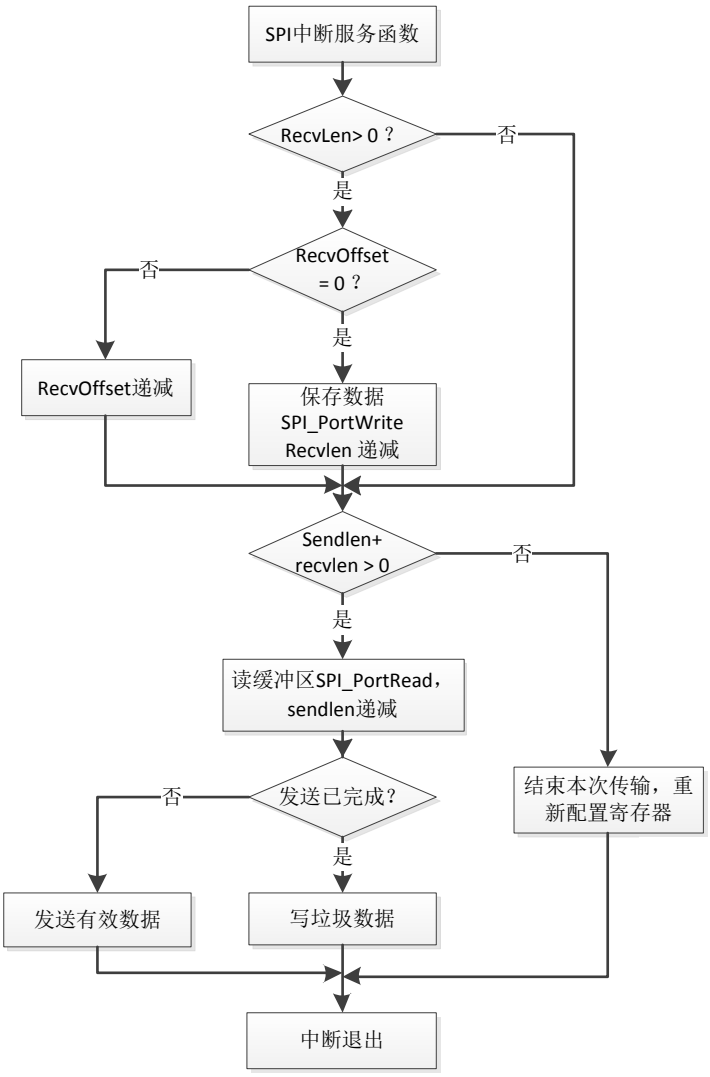

图 4-4 中断服务函数流程图

5.5. 移植建议

为了简化编程，提高工作效率， BSP 程序人员可采取下面的步骤快速的完成 DJYOS 驱动架构下 SPI 底层驱动的开发。

1、 拷贝其他工程已测试通过的 SPI 驱动文件 cpu_peri_spi.c/cpu_peri_spi.h；

2、 添加 SPI 的中断号到 critical.c 文件下面 tg_IntUsed 数组；

3、 修改 `cpu_peri_spi.c/cpu_peri_spi.h` 中与具体 `SPI` 寄存器相关的部分；

4、 回调函数的具体实现和中断收发数据。

调用器件驱动程序前，确保已经调用 `ModuleInstall_DjyBus` 和 `ModuleInstall_SPIBus` 安装 `DjyBus` 和 `SPIBus` 模块。

6. SPI 器件驱动接口

建议将器件驱动的存放目录为 djysrc\bsp\chip\xxx，其中，xxx 是具体芯片的文件夹名称。

SPI 总线初始化完成后，添加一个器件到总线上的过程，非常简单，就是初始化一下该器件的寻址特性参数，然后调用 SPI_DevAdd_s 或 SPI_DevAdd 函数把器件添加到总线上即可。需配置的参数，都在 spibus.h 文件中定义的 struct tagSPI_Device 中描述。struct tagSPI_Device 结构定义如下：

代码 5-1 SPI 器件结构体

```
//SPI总线器件结构体
struct tagSPI_Device
{
    struct tagRscNode DevNode;
    u8 Cs;                                  //片选信号
    bool_t AutoCs;                          //自动片选
    u8 CharLen;                             //数据长度
    u8 Mode;                                //模式选择
    u8 ShiftDir;                            //MSB or LSB
    u32 Freq;                               //速度,Hz
};
```

6.1. 初始化过程

添加器件到总线的过程就是将器件节点挂到相应的“SPIn”总线节点的过程，同时，配置好相应的总线通信参数。现对添加 SPI 器件要点作如下说明：

1、 若使用 SPI_DevAdd_s 挂载器件，定义 static struct tagSPI_Device 类型的静态变量；

2、 若使用 SPI_DevAdd_s 挂载器件，初始化数据 struct tagSPI_Param 的各成员；

3、 调用 SPI_DevAdd_s 或 SPI_DevAdd 添加设备到总线节点。

4、 调用 SPI_BusCtrl 设置总线参数

SPI_DevAdd_s 或 SPI_DevAdd 都可以把器件添加到总线上，但两者是有区别的：

1、 使用 SPI_DevAdd_s 的话，你需要自己准备 struct tagSPI_Device 结构，并且自行初始化，特别是，当操作系统的 spibus 模块被修改导致该结构的定义发生变化时，器件驱动程序也需要修改。

2、 使用 SPI_DevAdd_s 的好处是，该结构无须动态分配，符合像 OSEK 之类的严谨规范。

3、 使用 SPI_DevAdd 的好处是，驱动程序非常简单。

下面用 ATMEL 公司的 AT45 的 EEPROM 芯片为例说明添加设备过程。如代码 5-2所示，将 AT45 芯片添加到总线“SPI”，并命名为“SPI_Dev_AT45”。

代码 5-2 添加 SPI 设备实例

```
bool_t AT45_HardInit(void)
{
    bool_t result = false;
    if(s_AT45_InitFlag == true)
        return true;
    static struct tagSPI_Device s_AT45_Dev;

    s_AT45_Dev.AutoCs    = false;
    s_AT45_Dev.CharLen   = 8;
    s_AT45_Dev.Cs        = CN_AT45_SPI_CS;
    s_AT45_Dev.Freq      = CN_AT45_SPI_FRE;
    s_AT45_Dev.Mode      = SPI_MODE_1;
    s_AT45_Dev.ShiftDir = SPI_SHIFT_MSB;

    if(NULL != SPI_DevAdd_s("SPI","SPI_Dev_AT45",&s_AT45_Dev))
    {
        ps_AT45_Dev = &s_AT45_Dev;

        if(true == _at45db321_Check_ID())   //校验芯片ID
        {
            _at45db321_Binary_Page_Size_512();
```

```
            s_AT45_InitFlag = true;
            result = true;
        }
    }
    return result;
}
```

7. 访问器件

器件装载到装载总线之后，可以通过访问 SPI 总线实现访问器件，具体就是调用 spibus.h 提供的 API 函数 SPI_Transfer（）。