# 嵌入式手扎九 USB+(SPI/UART)

说明：这次手札是和课程报告一起在写的，说来惭愧，直到课程设计的时候才接触到USB编程，双机通信的IIC也只单独使用了UART进行了测试；并没有和SPI一样用USB进行主机通信；但对USB编程也有了一定的学习认知；

## 实验目的：

1、通过USB+SPI组合进行板间通信和PC机之间信息传递

## 实验环境：

1）PC机一台，KL25开发板一片，导线若干

2) KDS开发环境，VS2017或更高平台

## 实验内容：

## 实验一： 1、通过USB+SPI组合进行板间通信和PC机之间信息传递

### 1）USB模块的通信原理

#### 一、USB通信 两个模块：USB主机/USB从机

USB主机：主要负责发送： 接收从机数据命令：*cmdINTESTDATA* 发送数据给从机命令：*cmdOUTTESTDATA* 给USB从机来控制主从机之间的数据传输；

USB从机：主要负责接收主机发送过来的命令，接收主机发送的数据或者向主机发送数据；

注：本实验统一让MCU当作USB从机，PC端当作USB主机，进行SPI+USB板间通信；

#### 二、USB通讯 发送数据/接收数据 的过程分析：

##### 1）USB发送数据



**@源码：**

1）USB发送数据到USB从机函数

///函数名:sendData

///功 能:通过USB发送一个字节数组

///参 数:(1)sendData:存放需要发送的字节数组

/// (2)count:需要发送的字节数量，从数组第0个元素算起

///返 回:布尔值,当发送成功后,返回True，否则返回False

/// ----------------------------------------------------------------

public bool sendData(byte[] sendData, int count)

{

ErrorCode ec = ErrorCode.None;

int bytesWritten;

try

{

ec = writer.Write(sendData, 0, count, 100, out bytesWritten);

if (ec != ErrorCode.None)

throw new Exception(ec.ToString());

return true;

}

catch (Exception ex)

{

Console.WriteLine("Error:" + ex.Message);

return false;

}

}

///函数名:sendData

///功 能:通过USB发送一个字节

///参 数:(1)sendData:存放需要发送的一个字节

///返 回:布尔值,当发送成功后,返回True，否则返回False

/// ----------------------------------------------------------------

public bool sendData(byte sendData)

{

ErrorCode ec = ErrorCode.None;

int bytesWritten;

try

{

ec = writer.Write(sendData, 100, out bytesWritten);

if (ec != ErrorCode.None)

throw new Exception(ec.ToString());

return true;

}

catch (Exception ex)

{

Console.WriteLine("Error:" + ex.Message);

return false;

}

}

##### 2）USB接收数据





**@源码**

1）USB主机接收USB从机的函数

/// ----------------------------------------------------------------

///函数名:recvString

///功 能:通过USB接收一个字符换

///参 数:无

///返 回:从USB接收到的字符串

/// ----------------------------------------------------------------

public string recvString()

{

byte[] readBuffer = new byte[64];

int bytesRead;

reader.Read(readBuffer, 100, out bytesRead);

return Encoding.Default.GetString(readBuffer, 0, bytesRead);

}

/// ----------------------------------------------------------------

///函数名:recvByte

///功 能:通过USB接收一个字节数组

///参 数:无

///返 回:从USB接收到的字节数组

/// ----------------------------------------------------------------

public byte[] recvByte()

{

byte[] readBuffer = new byte[64];

int bytesRead;

reader.Read(readBuffer, 100, out bytesRead);

if (bytesRead > 0)

{

byte[] recvDataBuffer = new byte[bytesRead];

Array.Copy(readBuffer, recvDataBuffer, bytesRead);

return recvDataBuffer;

}

else

{

return null;

}

}

### 2）SPI模块分析

#### 一、串行外设接口SPI的通用基础知识：

串行外设接口SPI是一种同步串行通信接口，用于微处理器和外围扩展芯片指尖的串行连接，发展成为一种工业标准。

（1）主机与从机概念：一个SPI系统，由一个主机和一个或多个从机构成，主机启动一个与从机的同步通信，从而完成数据交换。

（2）主出从入引脚MOSI与主入从出引脚MISP

主出从入引脚是主机输出，从机出入数据线。

主入从出引脚是主机输入，从机输出数据线。

（3）SPI串行时钟引脚：SPI串行时钟引脚用于控制主机与从机之间的数据传输。

（4）时钟极性与时钟相位：时钟极性表示时钟信号在空闲时是高电平还是低电平。

（5）从机选择引脚SS：一些芯片带有从机选择引脚SS，也称为片选引脚。

#### 二、传输原理：

从主机CPU发出启动传输信号开始，将要传送的数据装入8位移位寄存器，并同时产生8个时钟信号，以此从SCK引脚送出，在SCK信号的控制下，主机中8位移位寄存器中的数据依次从MOSI引脚送出，到从机的MOSI引脚后送入它的8位移位寄存器。

SPI的时序：

（1）空闲电平低电平，上升沿取数

（2）空闲电平低电平，下降沿取数

（3）空闲电平高电平，下降沿取数

（4）空闲电平高电平，上升沿取数

#### 三、双向通信的两种方式

##### 1）八线双向

###### @1：设计思路：

这种方式是最简单的，两个MCU同时充当主机和从机，发送数据只用到每个板子的主机模块；接收数据只用到每个板子的从机模块

###### @2：源码：

#1：main.c

**#include** "includes.h" //包含总头文件

//USB序列号

//注意：修改USB序列号时，要将设备名的第一个字节数据长度字段进行修改

uint\_8 Serial\_String[] = {

0x10, //设备名长度字段(包含该字段本身长度)

0x03, //设备名索引

//以下是用户自定义设备名

'H', 0x00, 'W', 0x00, 'W', 0x00, '\_', 0x00, 'U', 0x00, 'S', 0x00, 'B',

0x00, };

//==========================================================

//函数名称：Delay\_ms

//函数返回：无

//参数说明：无

//功能概要：延时 - 毫秒级

//==========================================================

**void** **Delay\_ms**(uint16\_t u16ms) {

uint32\_t u32ctr;

**for** (u32ctr = 0; u32ctr < ((48000 / 10) \* u16ms); u32ctr++) {

**asm** ("NOP");

}

}

**int** **main**(**void**) {

uint\_32 mRuncount; //主循环计数器

// uint\_8 i;

uint\_8 j;

//2. 关总中断

DISABLE\_INTERRUPTS;

//3. 初始化外设模块

light\_init(RUN\_LIGHT\_BLUE, LIGHT\_ON); //蓝灯初始化

uart\_init(UART\_1, 9600); //串口1初始化,波特率9600

//把SPI0初始化为主机,波特率6000，时钟极性0，时钟相位0

SPI\_init(SPI\_0, 1, 6000, 0, 0);

//把SPI1初始化为从机,波特率6000，时钟极性0，时钟相位0

SPI\_init(SPI\_1, 0, 6000, 0, 0);

//USB设备初始化

usb\_init(Serial\_String);

//4. 给有关变量赋初值

mRuncount = 0; //主循环计数器

g\_USBRecvLength = 0; //USB接收数据长度初始化

g\_USBSendLength = 0; //USB发送数据长度初始化

g\_USBSendFlag = 0; //USB执行发送标志初始化

//5. 使能模块中断

uart\_enable\_re\_int(UART\_1); //uart接收中断

SPI\_enable\_re\_int(SPI\_1); //从机SPI\_1的接收中断

//6. 开总中断

ENABLE\_INTERRUPTS;

//进入主循环

//主循环开始===========================================================

**for** (;;) {

//运行指示灯（RUN\_LIGHT）闪烁---------------------------------------------

mRuncount++; //主循环次数计数器+1

**if** (mRuncount >= RUN\_COUNTER\_MAX) //主循环次数计数器大于设定的宏常数

{

mRuncount = 0; //主循环次数计数器清零

}

//以下加入用户程序--------------------------------------------------------

//USB主机向USB设备发送要数据命令

**if** (g\_USBRecv[0] == *cmdINTESTDATA* && g\_USBSendLength != 0) {

g\_USBRecv[0] = *cmdNULL*;

g\_USBSend[0] = *cmdINTESTDATA*;

g\_USBSendFlag = 1;

}

//USB主机发送数据，写入到发送数组中，以便主机要数据时进行发送

**else** **if** (g\_USBRecv[0] == *cmdOUTTESTDATA* && g\_USBRecvLength != 0) {

//禁止USB中断

disable\_irq(USB\_INTERRUPT\_IRQ);

//将接收数据的内容传输到发送缓存数组中进行保存，待发送

**for** (j = 0; j < g\_USBRecvLength; j++) {

spi\_Send[j + 2] = g\_USBRecv[j]; //Recv数组存放到SPI缓存数组中

}

//组成帧的格式

spi\_Send[0] = 'P';

spi\_Send[1] = g\_USBRecvLength + '0';

spi\_Send[g\_USBRecvLength + 2] = 'C';

spi\_SendLength = g\_USBRecvLength + 3;

g\_USBRecv[0] = *cmdNULL*;

g\_USBRecvLength = 0;

//组帧成功，发送操作，再SPI从机的接收中断中执行

//注释掉SPI主机发送程序

**int** m = 0;

**for** (m = 0; m < spi\_SendLength; m++) {

SPI\_send1(SPI\_0, spi\_Send[m]);

uart\_send1(UART\_1, spi\_Send[m]);

}

//使能USB中断

enable\_irq(USB\_INTERRUPT\_IRQ);

}

//将spi从机接受的内容传递给usb\_send[]内，准备发送给USB主机

**if** (spi\_flag == 1) {

uint\_8 t = 0;

**for** (t = 0; t < spi\_RecvLength ; t++) {

g\_USBSend[t] = spi\_Recv[t + 2];

}

g\_USBSendLength = spi\_RecvLength;

spi\_flag = 0;

}

} //主循环end\_for

//主循环结束=====================================================

}

#2：isr.c

**void** **USB0\_IRQHandler**(**void**) {

uint\_8 isr\_type;

DISABLE\_INTERRUPTS; //关总中断

//1. 获取中断类型

isr\_type = usb\_get\_isr();

//2. 若不是令牌完成中断，调用相应处理程序

**if** (isr\_type != USB\_TOKDNE\_INT) {

usb\_isr\_handler(isr\_type); //调用非令牌完成中断的处理程序

**goto** USB0\_IRQ\_exit;

}

//3. 是令牌完成中断，执行块数据传输或设备枚举

//(3.1)清收发ODD区，并指定EVEN区

FLAG\_SET(USB\_CTL\_ODDRST\_SHIFT, USB0\_CTL);

//(3.2)若是设备枚举请求，进行设备枚举

**if** ((USB0\_STAT >> 4) == 0) {

usb\_enumerate();

**goto** USB0\_IRQ\_exit;

}

//(3.3)若不是设备枚举请求，进行数据收发操作

//从端点2发送数据至主机

**if** ((USB0\_STAT & 0xF8) == mEP2\_IN) {

//若执行发送标志被置1，则执行发送数据函数

**if** (g\_USBSendFlag == 1) {

//调用发送数据处理函数

usb\_send(g\_USBSend, &g\_USBSendLength);

**if** (g\_USBSendLength == 0) //剩余未发送的数据长度为0

{

g\_USBSendFlag = 0; //发送完成，执行发送标志清0

}

**goto** USB0\_IRQ\_exit;

}

//若无数据需要发送，则向USB主机发送一个确认包

BDTtable[*bEP2IN\_ODD*].Cnt = 0;

vEP2State ^= 0x40;

BDTtable[*bEP2IN\_ODD*].Stat.\_byte = vEP2State;

//从端点3接收主机发来的数据

} **else** **if** ((USB0\_STAT & 0xF8) == mEP3\_OUT) {

usb\_recv(g\_USBRecv, &g\_USBRecvLength);

send\_flag = 1;

}

USB0\_IRQ\_exit:

//4. 清除中断标志，结束中断处理

FLAG\_SET(USB\_ISTAT\_TOKDNE\_SHIFT, USB0\_ISTAT);

ENABLE\_INTERRUPTS; //开总中断

**return**;

}

//=====================================================================

//函数名: SPI1\_IRQHandler

//功 能: SPI1接收中断服务程序

//参 数: 无

//返 回: 无

//说 明: 需要启动中断并注册才可使用

//=====================================================================

**void** **SPI1\_IRQHandler**(**void**) {

uint\_8 redata;

DISABLE\_INTERRUPTS;

redata = SPI\_receive1(SPI\_1); //接收主机发送过来的一个字节数据。

**if** (CreateFrame(redata, spi\_Recv) != 0) {

//组帧成功

spi\_RecvLength = spi\_Recv[1];

spi\_flag = 1;

}

ENABLE\_INTERRUPTS;

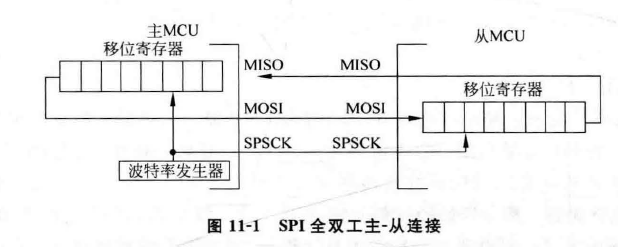
}

##### 2）四线双向

###### @1：设计思路：

这种思路是有一定难度的，但是最有价值的，两个MCU设定唯一的主机和从机；

通过利用SPI双机通信的特点来进行从机向主机的信息传递；



主机发送从机过程跟上面的八线双向相同，但是从机发往主机则利用主机发送的时序信号‘0’来将从机移位寄存器中的待发送数据按字节逐次发送给主机；

###### @2：问题解决

#1：（主机/从机--接收/发送） 数据的信号控制

通过定义全局变量uint\_8 rec\_flag(主机); uint\_8 spi\_flag(从机)来控制；每次接收到USB发送数据后通过USB中断设置为1；

1）主机rec\_flag信号设置为1代表：

1、不提供从机发主机的时序信号‘0’

2、主机接收中断开始判断从机移位寄存器发送过来的内容

2）从机spi\_flag信号设置为1代表：

1、从机开始根据主机发送的时序信号开始将数据内容逐字节发送到主机

3）源码

#1 USB主机

**void** **SPI0\_IRQHandler**(**void**) {

//仅在从机发送数据时 rec\_flag == 0 调用

uint\_8 redata;

DISABLE\_INTERRUPTS;

redata = SPI\_read1(SPI\_0); //接收主机发送过来的一个字节数据。

**if** (rec\_flag == 0) {//SPI主机不处于发送数据时间

**if** (redata == '0') { //若是时序信号则不进行操作

} **else** **if** (CreateFrame(redata, spi\_Recv) != 0) {

//组帧成功

spi\_RecvLength = spi\_Recv[1];

spi\_flag = 1;

}

}

ENABLE\_INTERRUPTS;

}

#2 USB从机

**void** **SPI1\_IRQHandler**(**void**) {

uint\_8 redata;

DISABLE\_INTERRUPTS;

redata = SPI\_receive1(SPI\_1); //接收主机发送过来的一个字节数据。

//SPI从机发送模块

**if** (send\_flag == 1) {

**if** (redata == '0') {

**int** m = 0;

**if** (m < spi\_SendLength) {

//收到一个时序信号，后将待发送内容转入移位寄存器

//待一个时序信号发送过来后，循环发送至SPI主机的移位寄存器中

SPI1\_D = spi\_Send[m];

m++;

}

**if** (m == spi\_SendLength) {

send\_flag = 0;//发送完毕后，SPI从机发送标志重新清零

}

}

}

//SPI从机接收模块

**if** (CreateFrame(redata, spi\_Recv) != 0) {

//组帧成功

spi\_RecvLength = spi\_Recv[1];

spi\_flag = 1;

}

ENABLE\_INTERRUPTS;

}

#2：主机发往从机的 时序信号控制

1）通过主机main.c中对的rec\_flag标志的判断：

为1则不发送时序标志；为0则发送时序标志

2）源码

**for** (;;) {

//运行指示灯（RUN\_LIGHT）闪烁---------------------------------------------

mRuncount++; //主循环次数计数器+1

**if** (mRuncount >= RUN\_COUNTER\_MAX) //主循环次数计数器大于设定的宏常数

{

mRuncount = 0; //主循环次数计数器清零

}

//以下加入用户程序--------------------------------------------------------

//SPI主机只要不发送数据，就一直为SPI从机提供时序信号‘0’，根据rec\_flag标志判断

**if** (rec\_flag == 0) {

SPI\_send1(SPI\_0, '0');

}

//USB主机向USB设备发送要数据命令

**if** (g\_USBRecv[0] == *cmdINTESTDATA* && g\_USBSendLength != 0) {

g\_USBRecv[0] = *cmdNULL*;

g\_USBSend[0] = *cmdINTESTDATA*;

g\_USBSendFlag = 1;

}

//USB主机发送数据，写入到发送数组中，以便主机要数据时进行发送

**else** **if** (g\_USBRecv[0] == *cmdOUTTESTDATA* && g\_USBRecvLength != 0) {

//禁止USB中断

disable\_irq(USB\_INTERRUPT\_IRQ);

//将接收数据的内容传输到发送缓存数组中进行保存，待发送

**for** (j = 0; j < g\_USBRecvLength; j++) {

spi\_Send[j + 2] = g\_USBRecv[j]; //Recv数组存放到SPI缓存数组中

}

//组成帧的格式

spi\_Send[0] = 'P';

spi\_Send[1] = g\_USBRecvLength + '0';

spi\_Send[g\_USBRecvLength + 2] = 'C';

spi\_SendLength = g\_USBRecvLength + 3;

g\_USBRecv[0] = *cmdNULL*;

g\_USBRecvLength = 0;

**int** m = 0;

**for** (m = 0; m < spi\_SendLength; m++) {

SPI\_send1(SPI\_0, spi\_Send[m]);

}

//发送一个时序信号，清空SPI从机移位寄存器的原有的内容

//保证下一次SPI主机中断接收的非‘0’ 内容一定是SPI从机发送的有效数据

SPI\_send1(SPI\_0, '0');

//SPI主机发送数据完毕，重新设置SPI主机接收数据信号标志为0；

rec\_flag = 0;

//使能USB中断

enable\_irq(USB\_INTERRUPT\_IRQ);

}

//将spi主机接受的内容传递给usb\_send[]内，准备发送给USB主机

**if** (spi\_flag == 1) {

uint\_8 t = 0;

**for** (t = 0; t < spi\_RecvLength; t++) {

g\_USBSend[t] = spi\_Recv[t + 2];

}

g\_USBSendLength = spi\_RecvLength;

spi\_flag = 0;

}

} //主循环end\_for

#3: 从机对发送数据的组帧处理

1）方法：

跟SPI主机发送到SPI从机对发送数据的main.c里的组帧方法相同，只不过将发送到主机的SPI的操作放在了SPI从机的接收中断函数里；

## 实验总结

通过这次试验深入理解了USB通信和SPI主从双向通信的原理；一开始学习的时候确实花了相当大的时间。

不同于以往，这次实验综合了两个模块，需要对两个模块的整体结构有一个很好的了解才能着手设计整体的通讯。

特别是在主从双向通讯的时候，一开始只完成了八线双向，往后深入做四线双向通信的时候遇到相当大的困难，我通过向身边优秀的同学请教，一点一点的尝试，才学会了。

这算是学期最后一次的实验了，这学期的嵌入式课程带给我的成长是最大的，给了我一个机会去了解硬件的相关知识，同时培养了自己编程能力和学习习惯。

通过嵌入式课程让我学会了：在遇到不懂的问题时懂得不放弃，肯花时间钻研。以往遇到的问题往往是种类比较单一的，但是练习嵌入式的时候，出的问题五花八门，解决它更需要相当大的精力。

甚至到现在，我还有很多很多的知识没学到。但是我相信：有了这种经验，以后尽管遇到困难，我也不会退缩了，问题没能解决之前都很困难，最好的办法就是解决它，自然问题就不困难了。