PS2 无线遥控手柄说明书

湖南智宇科技设备有限公司
2017 年 7 月

版权声明

目录

版权声明 2
1、PS2 手柄介绍 4
2、手柄的使用、连接配对说明 7
3、硬件连接方式 7
4、程序设计 7
5、下载与测试 17

ps2 手柄兼容索尼的 PlayStation2 游戏机的遥控手柄。索尼的 psx 系列游戏主机在全球很是畅销。不知什么时候便有人打起 ps2 手柄的主意，破解了通讯协议，使得手柄可以接在其他器件上遥控使用，比如遥控我们熟悉的机器人。突出的特点是这款手柄性价比极高，按键丰富，方便扩展到其它应用中。

1、PS2 手柄介绍

PS2 手柄由手柄与接收器两部分组成，手柄主要负责发送按键信息；接收器与单片机（也可叫作主机，可直接用在 PS2 游戏机上）相连，用于接收手柄发来的信息，并传递给单片机，单片机也可通过接收器，向手柄发送命令，配置手柄的发送模式。

特别声明：因批次不同或代工厂家不同，手柄和接收器的外观会有所区别（接收器上都有指示灯，但一种接收器上有电源灯，另一种没有电源灯），但是接收器的引脚定义是一样的，解码方式是一样的，使用相同。

接收器引脚输出：

1	2	3	4	5	6	7	8	9
DI/DAT	DO/CMD	NC	GND	VDD	CS/SEL	CLK	NC	ACK

接收器图片（一定要注意端口顺序）

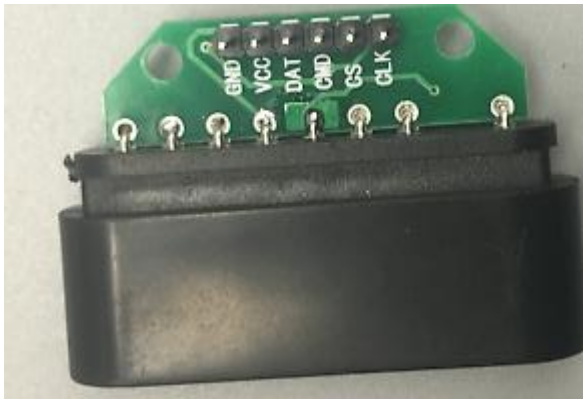


图 1-1：接收器引脚序号

DI/DAT:信号流向，从手柄到主机，此信号是一个 8bit 的串行数据，同步传送于时钟的下降沿。信号的读取在时钟由高到低的变化过程中完成。下降沿读取 CPOL=1,则CPOH=0，即硬件SPI模式2

MISO

DO/CMD:信号流向，从主机到手柄，此信号和 DI 相对，信号是一个 8bit 的串行数据， 同步传送于时钟的下降沿。

MOSI

NC:空端口；

GND:电源地；

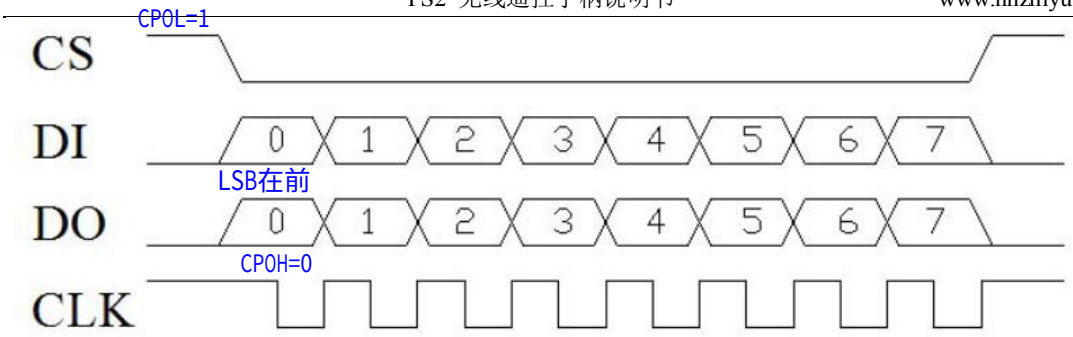
VDD:接收器工作电源，电源范围 3~5V;

CS/SEL:用于提供手柄触发信号。在通讯期间，处于低电平；

CLK:时钟信号，由主机发出，用于保持数据同步；

NC:空端口；

ACK:从手柄到主机的应答信号。此信号在每个 8bits 数据发送的最后一个周期变低并且 CS 一直保持低电平，如果 CS 信号不变低，约 60 微秒 PS 主机会试另一个外设。在编程时未使用 ACK 端口。



限制硬件SPI频率？

图 1- 2：通讯时序 显然，是硬件SPI的模式2

时钟频率 250KHz(4us)，如果接收数据不稳定，可以适当的增加频率。在通讯过中，一串数据通讯完成后 CS 才会由低转高，不是 1 个字节通讯完成后就由低转高，在通讯期间，一直处于低电平。准确来说，是下降沿或低电平 采样读取，上升沿或高电平 变化写入

在时钟下降沿时，完成数据（1bit)的发送与接收，发送和接收是同时完成的。当单片机想读手柄数据或向手柄发送命令时，将会拉低 CS 线电平，并发出一个命令“0x01”；手柄会回复它的 ID “0x41=绿灯模式，0x73=红灯模式”；在手柄发送 ID 的同时，单片机将传 送 0x42,请求数据；随后手柄发送出 0x5A，告诉单片机“数据来了”。

STM32和手柄 双方同时采样读取

idle:数据线空闲，该数据线无数据传送。

一个通讯周期有 9 个字节（8 位），这些数据是依次按位传送。

表 1- 1：数据意义对照表

顺序	DO	DI	Bit0、Bit1、Bit2、Bit3、Bit4、Bit5、Bit6、Bit7、
0	0X01	idle	
1	0x42	ID	
2	idle	0x5A	
3	WW	data	SELECT、L3、R3、START、UP、RIGHT、DOWN、LEFT
4	YY	data	L2、R2、L1、R1、△、○、╳、□
5	idle	data	PSS_RX（0x00=left、0xFF=right）
6	idle	data	PSS_RY（0x00=up、0xFF=down）
7	idle	data	PSS_LX（0x00=left、0xFF=right）
8	idle	data	PSS_LY（0x00=up、0xFF=down）

当有按键按下，对应位为“0”，其他位为“1”，例如当键“SELECT”被按下时，Data[3]=11111110B。

在设置了震动模式后，我们就能发送 WW、YY 来控制震动电机。WW，用来控制右侧的小震动电机，0x00 关，其他值为开；YY 用来控制左侧的大震动电机，0x40~0xFF 电机开，值越大，电机转动越快，震动越明显。具体的设置请看下面的程序部分。

红灯模式时：左右摇杆发送模拟值，0x00～0xFF 之间，且摇杆按下的键值 L3、R3 有效；

绿灯模式时：左右摇杆模拟值为无效，推到极限时，对应发送 UP、RIGHT、DOWN、

LEFT、△、○、X、□，按键 L3、R3 无效。

2、手柄的使用、连接配对说明

手柄需要两节 7 号 1.5V 的电池供电，接收器和单片机共用一个电源，电源范围为 3~5V，不能接反，不能超压，过压和反接，都会使接收器烧坏。

手柄上有个电源开关，ON 开/OFF 关，将手柄开关打到 ON 上，在未搜索到接收器的状况下，手柄的灯会不停的闪，在一定时间内，还未搜索到接收器，手柄将进入待机模式，手柄的灯将灭掉，这时，只有通过“START”键，唤醒手柄。

接收器供电，在未配对的情况下，绿灯闪。

手柄打开，接收器供电，手柄和接收器会自动配对，这时灯常亮，手柄配对成功。按键 “MODE”（手柄批次不同，上面的标识有可能是“ANALOG”，但使用方法一样），可以选择“红灯模式”，“绿灯模式”。

有些用户反映，手柄和接收器不能正常配对！多数问题是，接收器的接线不正确，或程序有问题。

解决方法：接收器只接电源（电源线一定要连接正确），不接任何数据线和时钟线，一般情况下手柄是能够配对成功。配对成功后灯常亮，说明手柄是好的，这时再检查接线是否正确，程序移植是否有问题。

3、硬件连接方式

接收器与 stm32 连接方式

DI->PB12；

DO->PB13；

CS->PB14；

CLK->PB15。

4、程序设计

完整程序详见工程文件。

这里主要介绍 pstwo.c 文件中的函数。

```
void PS2_Init(void)
{
    //输入  DI->PB12
    RCC->APB2ENR|=1<<3;        //使能 PORTB 时钟
    GPIOB->CRH&=0XFFF0FFFF;//PB12 设置成输入  默认下拉
    GPIOB->CRH|=0X00080000;
    //  DO->PB13      CS->PB14   CLK->PB15
    RCC->APB2ENR|=1<<3;      //使能 PORTB 时钟
    GPIOB->CRH&=0X000FFFFF;
    GPIOB->CRH|=0X33300000;    //PB13、PB14、PB15 推挽输出
}
```

端口初始化，PB12 为输入，PB13、PB14、PB15 为输出。

```
//向手柄发送命令
void PS2_Cmd(u8 CMD)
{
    volatile u16 ref=0x01;
    Data[1] = 0;
    for(ref=0x01;ref<0x0100;ref<<=1)
    {
        if(ref&CMD)
        {
            DO_H;                          //输出一位控制位
        }
        else DO_L;
        CLK_H;                             //时钟拉高
        delay_us(10);
        CLK_L;
        delay_us(10);
        CLK_H;
        if(DI)
            Data[1] = ref|Data[1];
    }
    delay_us(16);
}
```

```
//判断是否为红灯模式，0x41=模拟绿灯，0x73=模拟红灯
//返回值；0，红灯模式
//            其他，其他模式
u8 PS2_RedLight(void)
{
    CS_L;
    PS2_Cmd(Comd[0]);  //开始命令
    PS2_Cmd(Comd[1]);  //请求数据
    CS_H;
    if( Data[1] == 0X73)    return 0;
    else return 1;
}
//读取手柄数据
void PS2_ReadData(void)
{
    volatile u8 byte=0;
    volatile u16 ref=0x01;
    CS_L;
    PS2_Cmd(Comd[0]);  //开始命令
    PS2_Cmd(Comd[1]);  //请求数据
    for(byte=2;byte<9;byte++)          //开始接受数据
    {
        for(ref=0x01;ref<0x100;ref<<=1)
        {
            CLK_H;
            delay_us(10);
            CLK_L;
            delay_us(10);
            CLK_H;
              if(DI)
              Data[byte] = ref|Data[byte];
        }
        delay_us(16);
    }
    CS_H;
}
```

上面两个函数分别为主机向手柄发送数据、手柄向主机发送数据。手柄向主机发送的

数 据缓存在数组 Data[]中，数组中共有 9 个元素，每个元素的意义请见表 1-1。还有一个函数是 用来判断手柄的发送模式，也就是判断 ID 即 Data[1]的值。

```
//对读出来的 PS2 的数据进行处理,只处理按键部分
//按下为 0， 未按下为 1
u8 PS2_DataKey()
{
    u8 index;
    PS2_ClearData();
    PS2_ReadData();
    Handkey=(Data[4]<<8)|Data[3];      //这是 16 个按键  按下为 0， 未按下为 1
    for(index=0;index<16;index++)
    {
        if((Handkey&(1<<(MASK[index]-1)))==0)
        return index+1;
    }
    return 0;                //没有任何按键按下
}
//得到一个摇杆的模拟量      范围 0~256
u8 PS2_AnologData(u8 button)
{
    return Data[button];
}
//清除数据缓冲区
void PS2_ClearData()
{
    u8 a;
    for(a=0;a<9;a++)
        Data[a]=0x00;
}
```

8 位数 Data[3]与 Data[4],分别对应着 16 个按键的状态，按下为 0,未按下为 1。通过对这两个数的处理，得到按键状态并返回键值。

另一个函数的功能就是返回模拟值，只有在“红灯模式”下值才是有效的，拨动摇杆， 值才会变化，这些值分别存储在 Data[5]、Data[6]、Data[7]、Data[8]。

手柄配置初始化：

```
//short poll
void PS2_ShortPoll(void)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x42);
    PS2_Cmd(0X00);
    PS2_Cmd(0x00);
    PS2_Cmd(0x00);
    CS_H;
    delay_us(16);
}
//进入配置
void PS2_EnterConfing(void)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x43);
    PS2_Cmd(0X00);
    PS2_Cmd(0x01);
    PS2_Cmd(0x00);
    PS2_Cmd(0X00);
```

```
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    CS_H;
    delay_us(16);
}
//发送模式设置
void PS2_TurnOnAnalogMode(void)
{
    CS_L;
    PS2_Cmd(0x01);
    PS2_Cmd(0x44);
    PS2_Cmd(0X00);
    PS2_Cmd(0x01); //analog=0x01;digital=0x00  软件设置发送模式
    PS2_Cmd(0xEE); //Ox03 锁存设置，即不可通过按键“MODE”设置模式。
                   //0xEE 不锁存软件设置，可通过按键“MODE”设置模式。
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    CS_H;
    delay_us(16);
}
```

```
//振动设置
void PS2_VibrationMode(void)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x4D);
    PS2_Cmd(0X00);
    PS2_Cmd(0x00);
    PS2_Cmd(0X01);
    CS_H;
    delay_us(16);
}
//完成并保存配置
void PS2_ExitConfing(void)
```

```
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x43);
    PS2_Cmd(0X00);
    PS2_Cmd(0x00);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    CS_H;
    delay_us(16);
}
//手柄配置初始化
void PS2_SetInit(void)
{
    PS2_ShortPoll();
    PS2_ShortPoll();
    PS2_ShortPoll();
    PS2_EnterConfing();          //进入配置模式
    PS2_TurnOnAnalogMode(); //“红绿灯”配置模式，并选择是否保存
    PS2_VibrationMode();         //开启震动模式
    PS2_ExitConfing();           //完成并保存配置
}
```

可以看出配置函数就是发送命令，发送这些命令后，手柄就会明白自己要做什么了，发 送命令时，不需要考虑手柄发来的信息。

手柄配置初始化，PS2_ShortPoll()被执行了 3 次，主要是为了建立和恢复连接。具体的 配置方式请看注释。

```
/**************************************************************
Function:       void PS2_Vibration(u8 motor1, u8 motor2)
Description:  手柄震动函数，
Calls:              void PS2_Cmd(u8 CMD);
Input: motor1:右侧小震动电机  0x00 关，其他开
          motor1:左侧大震动电机  0x40~0xFF  电机开，值越大  震动越大
**************************************************************/
void PS2_Vibration(u8 motor1, u8 motor2)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);   //开始命令
    PS2_Cmd(0x42);   //请求数据
    PS2_Cmd(0X00);
    PS2_Cmd(motor1);
    PS2_Cmd(motor2);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    CS_H;
    delay_us(16);
}
```

只有在初始化函数 void PS2_SetInit(void)中，对震动电机进行了初始化(PS2_VibrationMode();//开启震动模式），这个函数命令才会被执行。

编写主函数：

```
int main(void)
{
    u8 key;
    Stm32_Clock_Init(9); //系统时钟设置
    delay_init(72);      //延时初始化
    uart_init(72,9600);  //串口 1 初始化
    LED_Init();
    PS2_Init();              //驱动端口初始化
    PS2_SetInit();       //配配置初始化,配置“红绿灯模式”，并选择是否可以修改
                         //开启震动模式
    while(1)
    {
        LED =! LED;
        key=PS2_DataKey();
        if(key!=0)                    //有按键按下
        {
            printf("  \r\n   %d  is  pressed  \r\n",key);
            if(key == 11)
            {
                PS2_Vibration(0xFF,0x00);     // 发 出 震 动 后 必 须 有 延 时
delay_ms(1000);
                delay_ms(500);
            }
            else if(key == 12)
            {
                PS2_Vibration(0x00,0x41);     // 发 出 震 动 后 必 须 有 延 时
delay_ms(1000);
                delay_ms(500);
            }
            else
                PS2_Vibration(0x00,0x00);
        }
      printf(" %5d %5d %5d %5d\r\n",PS2_AnologData(PSS_LX),PS2_AnologData(PSS_LY),
                                    PS2_AnologData(PSS_RX),PS2_AnologData(PSS_RY) );
        delay_ms(50);
    }
}
```

当有按键按下时，输出按键值。

5、下载与测试

编译程序并下载。遥控器上配配置为红灯模式，指示灯为红色，串口输出的模拟值为127 或 128，当晃动摇杆时，相应的模拟值就会改变，这时摇杆按键可以按下，可以输出键值，见图 5.2。

图 5.1


SSCOM 3.3
2 is pressed
128 89 128 127
2 is pressed
128 92 128 127
2 is pressed
128 92 128 127
2 is pressed
128 91 128 127
2 is pressed
128 89 128 127
2 is pressed
128 89 128 127
2 is pressed
128 89 128 127
打开文件
文件名
发送文件
停止发送
扩 展
RTS
串口号
COM4
打开串口
帮助
保存窗口
清除窗口
HEX显示
DTR
波特率
9600
定时发送
500
ms/次
发送新行
数据位
8
字符串输入框:
发送
HEX发送
大虾电子网
停止位
1
AT+NAMEYFRobot
www.daxia.cor
S:0
R:10716
COM4 已关闭 9600bps 8
CTS=0 DSR=0 RL

图 5.2

按下“△”，输出对应的键值“13”。

图 5.3

分别按下“L1”、“R1”，串口输出键值，并伴随手柄的震动。

图 5.4

按下“MODE”，改为绿灯模式，手柄上指示灯变为“绿色”，串口输出的模拟值为“255”，轻轻晃动摇杆，模拟值不变。

图 5.5

我们将右摇杆向上推到极限，这时串口输出“Mispressed”，键值对应“△”，但模拟的值不改变。

图 5.6

“红灯模式”和“绿灯模式”的主要区别就在与摇杆模拟值的输出。

尝试改变初始化中的设置，如发送模式的选择、是否锁存？看手柄有何变化。设置PS2_Vibration(0x00,0x41)中的值，感受手柄不同频率的震感。