第一百三十三节：常用的三种串口发送函数。

【133.1　发送单字节的底层驱动函数。】

单片机内置的“独立硬件串口模块”能直接实现“发送一个字节数据”的基础功能，因此，发送单字节的函数是应用层与硬件层的最小单位的接口函数，也称为底层驱动函数。应用层再复杂的发送函数都基于此最小单位的接口函数来实现。单片机应用层与“独立硬件串口模块”之间的接口通信是靠寄存器 SBUF 作为中间载体的，要实现发送单字节的最小接口函数，有如下三个关键点。

第一个，单片机应用层如何知道“硬件模块”已经发送完了一个字节，靠什么来识别？答：在初始化函数里，可以把“硬件模块”配置成，每发送完一个字节后都产生一次发送中断，在发送中断函数里让一个全局变量从 0 变成 1，依此全局变量作为识别是否已经发送完一个字节的标志。

第二个，发送一个字节数据的时候，如果“硬件模块”通讯异常，没有按预期产生发送中断，单片机就会一直处于死循环等待“完成标志”的状态，怎么办？答：在等待“完成标志”的时候，加入超时处理的机制。

第三个，在连续发送一堆数据时，如果接收方（或者上位机）发现有丢失数据的时候，如何调节此发送函数？答：可以根据实际调试的结果，如果接收方发现丢失数据，可以尝试在每发送一个字节之后插入一个 Delay 延时，延时的时间长度根据实际调试为准。我个人的经验中，感觉 stm32 这类 M3 核或者 M4 核的单片机在发送一个字节的时候只需判断是否发送完成的标志位即可，不需要插入 Delay 延时。但是在其它某些个别厂家单片机的串口发送数据中，是需要插入 Delay 延时作为调节，否则在连续发送一堆数据时会丢失数据，这个，应该以实际调试项目为准。

片段的讲解代码如下：

```
unsigned char Gu8ReceData;
unsigned char Gu8SendByteFinish=0; //发送一个字节完成的标志
void usart(void) interrupt 4     //串口的中断函数
{
    if(1==RI)
    {
        RI = 0;
        Gu8ReceData=SBUF;
    }
    else  //发送数据引起的中断
    {
       TI = 0;  //及时清除发送中断的标志，避免一直无缘无故的进入中断。
       Gu8SendByteFinish=1; //从 0 变成 1 通知主函数已经发送完一个字节的数据了。
    }
}

void UsartSendByteData(unsigned char u8SendData) //发送一个字节的底层驱动函数
{
    static unsigned int Su16TimeOutDelay;  //超时处理的延时计时器

    Gu8SendByteFinish=0;  //在发送一个字节之前，必须先把此全局变量的标志清零。
    SBUF =u8SendData; //依靠寄存器 SBUF 作为载体发送一个字节的数据
```

```
    Su16TimeOutDelay=0xffff;  //超时处理的延时计时器装载一个相对合理的计时初始值
    while(Su16TimeOutDelay>0)  //超时处理
    {
        if(1==Gu8SendByteFinish)
        {
            break;  //如果 Gu8SendByteFinish 为 1，则发送一个字节完成，退出当前循环等待。
        }
        Su16TimeOutDelay--;  //超时计时器不断递减
    }

    //Delay();//在实际应用中，当连续发送一堆数据时如果发现丢失数据，可以尝试在此增加延时
}
```

【133.2　发送任意起始位置任意长度的函数。】

要连续发送一堆数据，必须先把这堆数据封装成一个数组，然后编写一个发送数组的函数。该函数内部是基于“发送单字节的最小接口函数”来实现的。该函数对外通常需要两个接口，一个是数组的任意起始位置，一个发送的数据长度。数组的任意起始位置只需靠指针即可实现。片段的讲解代码如下：

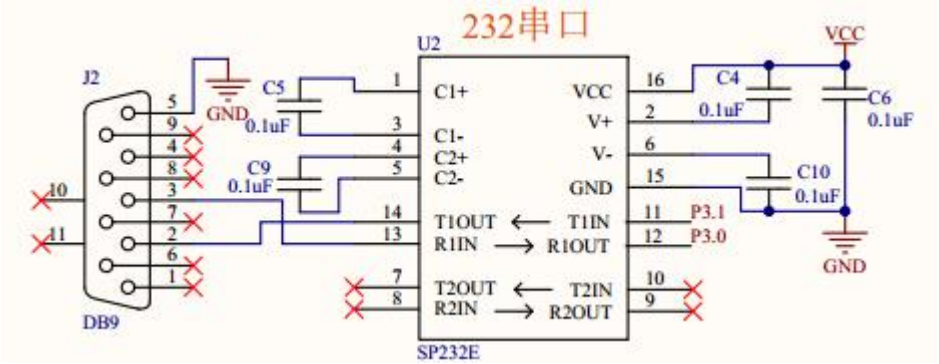
```
//任意数组
unsigned char Gu8SendBuffer[11]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A};

//发送任意起始位置任意长度的函数
void UsartSendBuffer(const unsigned char *pCu8SendBuffer,unsigned long u32SendSize)
{
    static unsigned long i;
    for(i=0;i<u32SendSize;i++) //u32SendSize 为发送的数据长度
    {
       UsartSendByteData(pCu8SendBuffer[i]);  //基于“发送单字节的最小接口函数”来实现的
    }
}

void main()
{
    UsartSendBuffer((const unsigned char *)&Gu8SendBuffer[0],5);//从第 0 位置发送 5 个数据
    UsartSendBuffer((const unsigned char *)&Gu8SendBuffer[6],5);//从第 6 位置发送 5 个数据
    while(1)
    {

    }
}
```

【133.3　发送带协议的函数。】

前面章节中，我们讲过接收“带固定协议”的程序框架，这类“带固定协议”的数据串里本身就自带了“数据的长度”，因此，要编程一个发送带协议的函数，关键在于，在函数内部根据协议先提取整串数据的有效长度。该函数对外通常也需要两个接口，一个是数组的起始位置，一个发送数据的最大限制长度。最大限制长度的作用是用来防止数组越界，增强程序的安全性。片段的讲解代码如下：

```
//“固定协议”十六进制的数据格式：EB 01 00 00 00 0B 03 E8 00 01 0B 。其中：
// EB 是数据头。
// 01 是代表数据类型。
// 00 00 00 0B 代表数据长度是 11 个（十进制）。
// 03 E8 00 01 0B 代表其它数据

//“带固定协议”的数组
unsigned char Gu8SendMessage[11]={0xEB,0x01,0x00,0x00,0x00,0x0B,0x03,0xE8,0x00,0x01,0x0B};

//发送带协议的函数
void UsartSendMessage(const unsigned char *pCu8SendMessage,unsigned long u32SendMaxSize)
{
    static unsigned long i;
    static unsigned long *pSu32;
    static unsigned long u32SendSize;

    pSu32=(const unsigned long *)&pCu8SendMessage[2];
    u32SendSize=*pSu32;  //从带协议的数组中提取整包数组的有效发送长度

    if(u32SendSize>u32SendMaxSize) //如果“有效发送长度”大于“最大限制的长度”,数据异常
    {
       return;  //数据异常，直接退出当前函数，预防数组越界
    }

    for(i=0;i<u32SendSize;i++) //u32SendSize 为发送的数据长度
    {
       UsartSendByteData(pCu8SendMessage[i]); //基于“发送单字节的最小接口函数”来实现的
    }
}

void main()
{
    UsartSendMessage((const unsigned char *)&Gu8SendMessage[0],100); //必须从第 0 位置发送
    while(1)
    {

    }
}
```

【133.4　程序例程。】



上图 133.4.1　　232 串口电路

程序功能如下：

单片机上电瞬间，直接发送三串数据。

第一串是十六进制的任意数据：00 01 02 03 04

第二串是十六进制的任意数据：06 07 08 09 0A

第三串是十六进制的“带协议”数据：EB 01 00 00 00 0B 03 E8 00 01 0B

波特率 9600，校验位 NONE（无），数据位 8，停止位 1。在电脑的串口助手软件里，设置接收显示的为“十六进制”（HEX 模式），即可观察到发送的三串数据。

代码如下：

```
#include "REG52.H"

void UsartSendByteData(unsigned char u8SendData); //发送一个字节的底层驱动函数

//发送任意起始位置任意长度的函数
void UsartSendBuffer(const unsigned char *pCu8SendBuffer,unsigned long u32SendSize);

//发送带协议的函数
void UsartSendMessage(const unsigned char *pCu8SendMessage,unsigned long u32SendMaxSize);

void usart(void);  //串口接收的中断函数
void SystemInitial(void);
void Delay(unsigned long u32DelayTime);
void PeripheralInitial(void);

unsigned char Gu8ReceData;
unsigned char Gu8SendByteFinish=0; //发送一个字节完成的标志
```

```
//任意数组
unsigned char Gu8SendBuffer[11]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A};

//“固定协议”十六进制的数据格式：EB 01 00 00 00 0B 03 E8 00 01 0B 。其中：
// EB 是数据头。
// 01 是代表数据类型。
// 00 00 00 0B 代表数据长度是 11 个（十进制）。
// 03 E8 00 01 0B 代表其它数据
//“带固定协议”的数组
unsigned char Gu8SendMessage[11]={0xEB,0x01,0x00,0x00,0x00,0x0B,0x03,0xE8,0x00,0x01,0x0B};

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();    //在此函数内部调用了发送的三串数据
    while(1)
    {
    }
}

void usart(void) interrupt 4     //串口的中断函数
{
        if(1==RI)
        {
            RI = 0;
            Gu8ReceData=SBUF;
        }
        else  //发送数据引起的中断
        {
           TI = 0;  //及时清除发送中断的标志，避免一直无缘无故的进入中断。
           Gu8SendByteFinish=1; //从 0 变成 1 通知主函数已经发送完一个字节的数据了。
        }
}

void UsartSendByteData(unsigned char u8SendData) //发送一个字节的底层驱动函数
{
        static unsigned int Su16TimeOutDelay;  //超时处理的延时计时器

        Gu8SendByteFinish=0;  //在发送以字节之前，必须先把此全局变量的标志清零。
        SBUF =u8SendData; //依靠寄存器 SBUF 作为载体发送一个字节的数据
        Su16TimeOutDelay=0xffff;  //超时处理的延时计时器装载一个相对合理的计时初始值
        while(Su16TimeOutDelay>0)  //超时处理
        {
```

```
            if(1==Gu8SendByteFinish)
            {
                break;  //如果 Gu8SendByteFinish 为 1，则发送一个字节完成，退出当前循环等待。
            }
            Su16TimeOutDelay--;  //超时计时器不断递减
        }

        //Delay();//在实际应用中，当连续发送一堆数据时如果发现丢失数据，可以尝试在此增加延时
}

//发送任意起始位置任意长度的函数
void UsartSendBuffer(const unsigned char *pCu8SendBuffer,unsigned long u32SendSize)
{
        static unsigned long i;
        for(i=0;i<u32SendSize;i++) //u32SendSize 为发送的数据长度
        {
           UsartSendByteData(pCu8SendBuffer[i]);  //基于“发送单字节的最小接口函数”来实现的
        }
}

//发送带协议的函数
void UsartSendMessage(const unsigned char *pCu8SendMessage,unsigned long u32SendMaxSize)
{
        static unsigned long i;
        static unsigned long *pSu32;
        static unsigned long u32SendSize;

        pSu32=(const unsigned long *)&pCu8SendMessage[2];
        u32SendSize=*pSu32;  //从带协议的数组中提取整包数组的有效发送长度

        if(u32SendSize>u32SendMaxSize) //如果“有效发送长度”大于“最大限制的长度”,数据异常
        {
           return;  //数据异常，直接退出当前函数，预防数组越界
        }

        for(i=0;i<u32SendSize;i++) //u32SendSize 为发送的数据长度
        {
           UsartSendByteData(pCu8SendMessage[i]); //基于“发送单字节的最小接口函数”来实现的
        }
}

void SystemInitial(void)
{
    unsigned char u8_TMOD_Temp=0;
```

```
    //以下是定时器 0 的中断的配置
    TMOD=0x01;
    TH0=0xfc;
    TL0=0x66;
    EA=1;
    ET0=1;
    TR0=1;

    //以下是串口接收中断的配置
    //串口的波特率与内置的定时器 1 直接相关，因此配置此定时器 1 就等效于配置波特率。
    u8_TMOD_Temp=0x20; //即将把定时器 1 设置为：工作方式 2，初值自动重装的 8 位定时器。
    TMOD=TMOD&0x0f; //此寄存器低 4 位是跟定时器 0 相关，高 4 位是跟定时器 1 相关。先清零定时器 1。
    TMOD=TMOD|u8_TMOD_Temp;  //把高 4 位的定时器 1 填入 0x2，低 4 位的定时器 0 保持不变。
    TH1=256-(11059200L/12/32/9600);  //波特率为 9600。11059200 代表晶振 11.0592MHz，
    TL1=256-(11059200L/12/32/9600);  //L 代表 long 的长类型数据。根据芯片手册提供的计算公式。
    TR1=1;  //开启定时器 1

    SM0=0;
    SM1=1;  //SM0 与 SM1 的设置：选择 10 位异步通信，波特率根据定时器 1 可变
    REN=1;  //允许串口接收数据

    //为了保证串口中断接收的数据不丢失，必须设置 IP = 0x10，相当于把串口中断设置为最高优先级，
    //这个时候，串口中断可以打断任何其他的中断服务函数实现嵌套，
    IP =0x10;  //把串口中断设置为最高优先级，必须的。

    ES=1;          //允许串口中断
    EA=1;          //允许总中断
}

void Delay(unsigned long u32DelayTime)
{
    for(;u32DelayTime>0;u32DelayTime--);
}

void PeripheralInitial(void)
{
    //发送任意数组
    UsartSendBuffer((const unsigned char *)&Gu8SendBuffer[0],5);//从第 0 位置发送 5 个数据
    UsartSendBuffer((const unsigned char *)&Gu8SendBuffer[6],5);//从第 6 位置发送 5 个数据

    //发送带协议的数组
    UsartSendMessage((const unsigned char *)&Gu8SendMessage[0],100); //必须从第 0 位置发送
}
```