第一百二十七节： 单片机串口接收数据的机制。

【127.1　单片机串口接收数据的底层时序。】

上一节“单线的肢体接触通信”其实是为本节打基础的，通信线只用了一根“数据”线，没有用到“时钟”线，属于异步通信方式，还分析时序中的“1 个开始位，8 个数据位，1 个停止位”等细节内容，这些时序其实就是本节单片机串口通信的底层时序，一模一样。继续上一节的内容（很有必要重新温习一次上一节的异步通信原理），继续沿用甲乙双方靠各自“心跳”的节拍来异步通信的例子，本节单片机串口接收数据是代表乙方，我把乙方串口接收数据的过程翻译成 C 语言，代码如下：

```
sbit USART_RX=P3^0;  //用来接收串口数据的数据线
unsigned char Gu8ReceiveData=0;  //串口接收到的 8 位数据
unsigned char i; //连续接收 8 位数据的循环变量
void main()
{
    Gu8ReceiveData=0;
    while(1)
    {
        USART_RX=1;   //51 单片机的规则，每次读取数据前都执行一条“置 1”指令
        Delay();      //乙的心跳间隔时间，待机时，每一个节拍监控一次数据线的状态
        if(0==USART_RX) //如果监控到甲发送的“开始位 0”，从下一个节拍开始连续接收 8 位数据
        {
              for(i=0;i<8;i++) //连续循环接收 8 个“数据位”
              {
                  USART_RX=1;   //51 单片机的规则，每次读取数据前都执行一条“置 1”指令
                  Delay();      //乙的心跳间隔时间，每个节拍判断读取一位数据
                  if(1==USART_RX)  //判断读取数据线上的状态
                  {
                     Gu8ReceiveData=Gu8ReceiveData | 0x80;
                  }
                  else
                  {
                     Gu8ReceiveData=Gu8ReceiveData & 0x7F;
                  }
                  Gu8ReceiveData=Gu8ReceiveData>>1; //右移一位，为即将接收下一位做准备
              }
              Delay();    //乙的心跳间隔时间，这里额外增加一个节拍，作为“停止位”的开销。
        }
    }
}
```

【127.2　单片机内置的“硬件串口模块”。】

很显然，上面【127.1】分享的时序代码会占用单片机大量的时间，单片机每接收一个字节的数据都会被束缚一次手脚，耽误了其它大事，怎么办？为了把单片机从底层繁琐的时序中解放出来，单片机内置了很多“硬货”，俗称“硬件资源”，“硬件串口模块”便是其中之一。何谓“硬件”，单片机内置的“硬件”可以看作是另外一个独立运行的“核”，这个“核”可以看作是另外一个 CPU，可以独立工作，相当于单片机主人在某个领域的一个专用助手。单片机只需要跟这个“核”通信发指令就可以，具体的执行过程由这个“核”独立去完成，这个“核”完成工作之后再把处理结果反馈给单片机。那么，单片机是如何跟这些内置“硬件资源”通信呢?其实它们的通信接口是“寄存器”，不管是单片机给“硬件资源”发送指令，还是单片机从“硬件资源”里读取所需要的结果数据，都是通过“寄存器”来完成。

【127.3　单片机与硬件串口通信的接口“寄存器”。】

硬件串口的寄存器主要涉及：串口的方式选择，波特率，允许串口接收数据，中断的优先级，中断的允许，等等。比如，51 单片机的串口是兼容很多种方式的，可以同步通信，也可以异步通信，异步通信还可以兼容 10 位（1 开始位、8 数据位、1 停止），11 位（1 开始位、8 数据位、1 校验位、1 停止），等等，这些就是多选题,我们要在某个特定的寄存器里面做出选择。波特率,是用来衡量通信的速度,比如波特率是 9600，就意味着 1 秒钟能收发 9600 个二进制的位数据，也就是 1 秒钟能产生 9600 个时钟节拍，波特率越高通信的速度越快，这些也需要我们往相关的寄存器填入相应的数据，来告知“硬件串口”以哪种波特率进行通信。

那么，对于初学者，寄存器如何配置呢？主要有这些思路：查看芯片手册（datasheet），产看 C 编译器的手册，查看芯片相关的 C 语言的头文件(比如 51 单片机的 REG.H)，在网上参考别人已经配置好的代码，或者购买相关芯片的学习板时所配套的程序例程。

本节用到的串口，是 10 位数据长度的异步通信，波特率 9600，相关配置的代码如下：

```
unsigned char u8_TMOD_Temp=0;

//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;  //往高 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，实现嵌套的功能，
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
```

【127.4　硬件串口的中断函数。】

硬件串口接收完一个字节的数据之后，会及时产生一个串口中断去通知单片机接收数据。单片机在串口中断函数里直接读取“串口专用缓存寄存器”SBUF 的数据，就可以直接获得一个完整的 8 位宽度的数据，省去了繁琐的驱动时序底层。

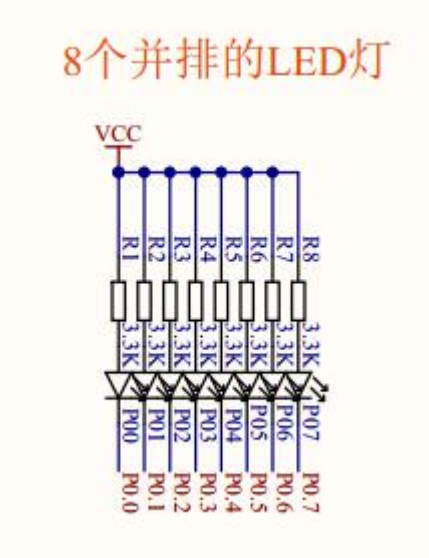
串口的中断函数跟定时器的中断函数很类似，只不过中断号不一样而已，比如我们前面章节用的定时器 0 的中断号是“1”，而本节串口的中断号是“4”。这些其实是 C 编译器定的游戏规则，我们只要根据它提供的数据手册遵守它的游戏规则就好了。串口中断函数里还有一个地方要注意，硬件串口“接收完一个字节”的数据后产生一次中断，而硬件串口“发送完一个字节”的数据后也产生一次中断，这两个一“收”一“发”的中断都是共用中断号为“4”的中断函数，因此，我们必须在中断函数里通过判断寄存器的 RI 和 TI 的标志位来判断到底是“收”的中断，还是“发”的中断，并且软件上要及时把 RI 或者 TI 及时清零，避免不断进入中断的情况。参考代码如下：

```
unsigned char Gu8ReceiveData=0;  //接收到一个字节的数据

void usart(void) interrupt 4   //串口接发的中断，中断号为 4
{
   if(1==RI)  //接收完一个字节后引起的中断
   {
        RI = 0; //及时清零，避免一直无缘无故的进入中断。
        Gu8ReceiveData=SBUF;  //直接读取“串口专用缓存寄存器”SBUF 的 8 位数据。
   }
   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。
        //以下可以添加一个全局变量的标志位的相关代码，通知主函数已经发送完一个字节的数据了。

   }
}
```

【127.5　上位机与单片机的串口通信例程。】



上图 127.5.1　　灌入式驱动 8 个 LED

程序功能如下：

波特率 9600，校验位 NONE（无），数据位 8，停止位 1。在上位机的串口助手里，发送一个十六进制（HEX 模式）的 01，让单片机的一颗 LED“亮”。发送一个十六进制（HEX 模式）的 00，让单片机的一颗 LED“灭”。上位机的串口助手的使用，请参考前面第 11 节的相关内容，或者自己在网上查找串口助手软件的使用方法，串口助手软件网上很多，我们的使用要求不高，随便选用一家都可以。代码如下：

```
#include "REG52.H"

void usart(void);

sbit P0_0=P0^0;  //一颗 LED 灯

unsigned char Gu8ReceiveData=0;  //接收到一个字节的数据

void main()
{
    unsigned char u8_TMOD_Temp=0;

    P0_0=1;  //LED 灭。1 代表 LED 灭， 0 代表亮

    //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
```

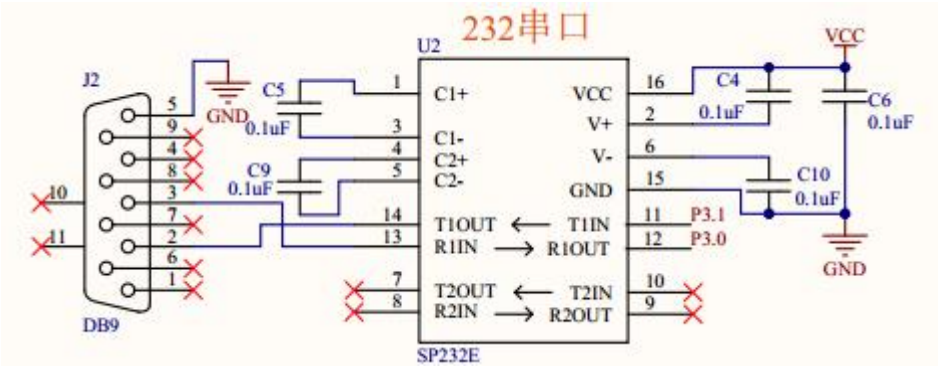
```
    SM0=0;
    SM1=1;  //SM0 与 SM1 的设置：选择 10 位异步通信，波特率根据定时器 1 可变
    REN=1;  //允许串口接收数据

    //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
    while(1)  //主循环
    {
    }
}

void usart(void) interrupt 4   //串口接发的中断函数，中断号为 4
{
   if(1==RI)  //接收完一个字节后引起的中断
   {
        RI = 0; //及时清零，避免一直无缘无故的进入中断。
        Gu8ReceiveData=SBUF;  //直接读取“串口专用缓存寄存器”SBUF 的 8 位数据。
        if(0x01==Gu8ReceiveData)
        {
            P0_0=0;  //LED 亮。1 代表 LED 灭， 0 代表亮
        }
        else
        {
            P0_0=1;  //LED 灭。1 代表 LED 灭， 0 代表亮
        }
   }
   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。
        //以下可以添加一个全局变量的标志位的相关代码，通知主函数已经发送完一个字节的数据了。
   }
}
```

【127.6　单片机串口电路的简易分析。】



上图 127.6.1　　232 串口电路

单片机串口对外的引脚是与 IO 口的“P3.1、P3.0”共用的。P3.1 是串口的 TX 引脚，即对外发送数据的引脚。P3.0 是串口的 RX 引脚，即接收外部数据的引脚。一旦项目中用了串口，那么这两个引脚就必须“专脚专用”，只给串口单独使用，不再做 IO 口。平时通信的时候，跟其它单片机或者系统进行串口通信，除了接 TX 和 RX 这两根数据线之外，必须一定把双方的负极 GND 也“共地”接上，否则双方建立不了同样的电压参考点，通信毕然失败。因此，串口通信最低标配是 3 根线：RX,TX,GND。

如果两个甲乙单片机都布在一块板子上，距离不超过半米，他们两个要进行串口通信，怎么接线？把他们的 GND 连起来，然后 RX 与 TX“交叉”对接，甲的 RX 接到乙的 TX，甲的 TX 接到乙的 RX。这种在短距离通信的时候，不用增加任何外部辅助压差信号放大芯片，这种方式叫做“串口的 TTL”接线方式。

如果两个系统串口通信的距离比较远，比如在不同的板子上，1 米以上 10 米以下的距离，这时就不能采用原始的“串口的 TTL”接线方式，因为线缆越长电阻越大，本身就要消耗一些压降，而 3.3V 的压降很容易就会被消耗完，通信的可靠度和抗扰能力就会降低。为了解决这个问题，可以引用 232 标准的接线方式，外部需接一个压差放大的芯片，把从原来 3.3V 的压差放大到一两倍左右，通信的距离就大大提高。具体 232 的细节，大家可以网上搜搜“RS232”。注意，采用 232 协议通信，也要注意“共地”和数据线“交叉”的两个问题，232 通信的最低标配也是 3 根线：R,T,GND。上图 SP232E 就是一个压差信号放大的通信专用芯片。