第一百三十二节：“转发、透传、多种协议并存”的双缓存串口程序框架。

【132.1　字节间隔时间、双缓存切换、指针切换关联。】

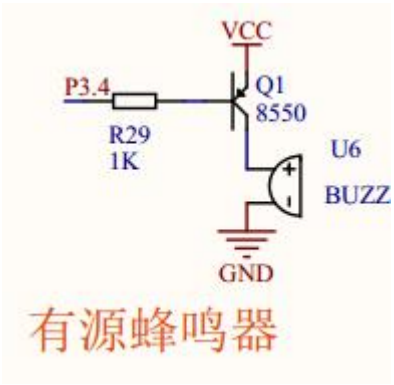
在一些通讯模块的项目中，常常涉及数据的转发，透传，提取关键字的处理，单片机接收到的数据不许随意丢失，必须全部暂存，然后提取关键字，再把整包数据原封不动地转发或者透传给“下家”。这类项目的特点是，通讯协议不是固定唯一的，因此，前面章节那种接头暗号（数据头）的程序框架不适合这里，本节跟大家分享另外一种程序框架。

第一个要突破的技术难点是，既然通讯协议不是固定唯一的，那么，如何识别一串数据已经接收完毕？答案是靠接收每个字节之间的间隔时间来识别。当一串数据正在接收时，每个字节之间的间隔时间是“短暂的相对均匀的”。当一串数据已经接收完毕时，每个字节之间的间隔时间是“突然变长的”。代码的具体实现，是靠一个软件定时器，模拟单片机“看门狗”的“喂狗”原理。

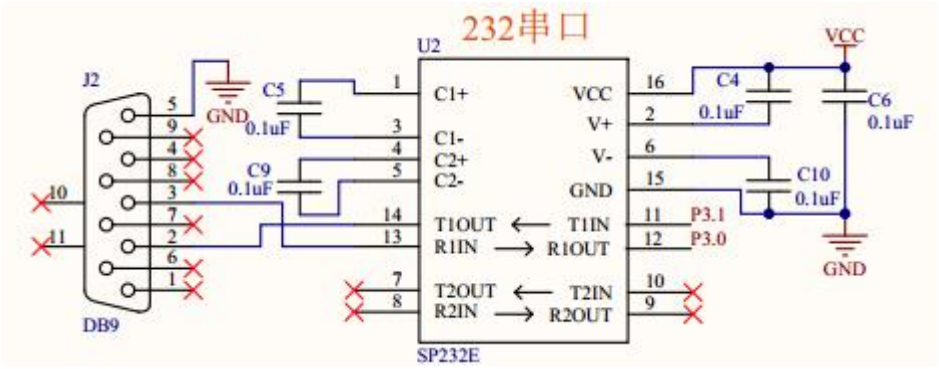
第二个要突破的技术难点是，既然通讯协议不是固定唯一的，数据内容带有随机性，甚至字节之间的间隔时间的长短也带有随机性和不确定性，那么，如何预防正在处理数据时突然“接收中断”又接收到的新数据覆盖了尚未来得及处理的旧数据，或者，如何预防正在处理旧数据时丢失了突然又新过来的本应该接收的新数据？答案是用双缓存轮流切换的机制。双缓存，一个用在处理刚刚接收到的旧数据，另一个用在时刻准备着接收新数据，轮流切换，两不误。

第三个要突破的技术难点是，既然是用双缓存轮流切换的机制，那么，在主程序里如何统一便捷地处理两个缓存的数组？这里的“统一”是关键，要把两个数组“统一”成（看成是）一个数组，方法是，只需用“指针切换关联”的技术就可以了。

【132.2　程序例程。】



上图 132.2.1　　有源蜂鸣器电路



上图 132.2.2　　232 串口电路

程序功能如下：单片机接收任意长度（最大一次不超过 30 字节）的一串数据。如果发现连续有三个字节是 0x02 0x03 0x04，蜂鸣器则“短叫”100ms 提示；如果发现连续有四个字节是 0x06 0x07 0x08 0x09，蜂鸣器则“长叫”2000ms 提示。

比如测试“短叫”100ms，发送十六进制的数据串：05 02 00 00 02 03 04 09

比如测试“长叫”2000ms，发送十六进制的数据串：02 02 06 07 08 09 01 08 03 00 05

代码如下：

```
#include "REG52.H"

#define DOG_TIME_OUT  20  //理论上，9600 波特率的字节间隔时间大概 0.8ms 左右，因此取 20ms 足够
#define RECE_BUFFER_SIZE  30   //接收缓存的数组大小

void usart(void);  //串口接收的中断函数
void T0_time();    //定时器的中断函数

void UsartTask(void);    //串口接收的任务函数，放在主函数内

void SystemInitial(void) ;
void Delay(unsigned long u32DelayTime) ;
void PeripheralInitial(void) ;

void BeepOpen(void);
void BeepClose(void);
void VoiceScan(void);

sbit P3_4=P3^4;

unsigned char Gu8CurrentReceBuffer_Sec=0; //当前接收缓存的选择标志。0 代表缓存 A，1 代表缓存 B

unsigned char Gu8ReceBuffer_A[RECE_BUFFER_SIZE]; //双缓存其中之一的缓存 A
unsigned long Gu32ReceCnt_A=0;    //缓存 A 的数组下标与计数器，必须初始化为 0，做好接收准备
```

```
unsigned char Gu8ReceBuffer_B[RECE_BUFFER_SIZE]; //双缓存其中之一的缓存 B
unsigned long Gu32ReceCnt_B=0;    //缓存 B 的数组下标与计数器，必须初始化为 0，做好接收准备

unsigned char Gu8ReceFeedDog=1; //“喂狗”的操作变量。
unsigned char Gu8FinishFlag=0; //接收完成标志。0 代表还没有完成，1 代表已经完成了一次接收

volatile unsigned char vGu8ReceTimeOutFlag=0;//通信过程中字节之间的超时定时器的开关
volatile unsigned int vGu16ReceTimeOutCnt=0; //通信过程中字节之间的超时定时器，“喂狗”的对象

volatile unsigned char vGu8BeepTimerFlag=0;
volatile unsigned int vGu16BeepTimerCnt=0;

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
    {
       UsartTask();   //串口接收的任务函数
    }
}

void usart(void) interrupt 4
{
   if(1==RI)
   {
        RI = 0;

        Gu8FinishFlag=0; //此处也清零，意味深长，当主函数正在处理数据时，可以兼容多次接收完成
        Gu8ReceFeedDog=1; //看门狗的“喂狗”操作，给软件定时器继续“输血”
        if(0==Gu8CurrentReceBuffer_Sec)   //0 代表选择缓存 A
        {
              if(Gu32ReceCnt_A<RECE_BUFFER_SIZE)
              {
                  Gu8ReceBuffer_A[Gu32ReceCnt_A]=SBUF;
                  Gu32ReceCnt_A++; //记录当前缓存 A 的接收字节数
              }
        }
        else     //1 代表选择缓存 B
        {
              if(Gu32ReceCnt_B<RECE_BUFFER_SIZE)
              {
```

```
                Gu8ReceBuffer_B[Gu32ReceCnt_B]=SBUF;
                Gu32ReceCnt_B++;  //记录当前缓存 B 的接收字节数
            }

        }
    }
    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。
    }
}

void UsartTask(void)    //串口接收的任务函数，放在主函数内
{
    static unsigned char *pSu8ReceBuffer;  //“指针切换关联”中的指针，切换内存
    static unsigned char Su8Lock=0;  //用来避免一直更新的临时变量
    static unsigned long i;  //用在数据处理中的循环变量
    static unsigned long Su32ReceSize=0; //接收到的数据大小的临时变量


    if(1==Gu8ReceFeedDog) //每被“喂一次狗”，就及时更新一次“超时检测的定时器”的初值
    {
        Gu8ReceFeedDog=0;

        Su8Lock=0; //解锁。用来避免一直更新的临时变量

        //以下三行代码是看门狗中的“喂狗”操作。继续给软件定时器“输血”
        vGu8ReceTimeOutFlag=0;
        vGu16ReceTimeOutCnt=DOG_TIME_OUT;//正在通信时，两个字节间隔的最大时间，本节选用 20ms
        vGu8ReceTimeOutFlag=1;
    }
    else if(0==Su8Lock&&0==vGu16ReceTimeOutCnt) //超时，代表一串数据已经接收完成
    {
        Su8Lock=1;  //避免一直进来更新
        Gu8FinishFlag=1; //两个字节之间的时间超时，因此代表了一串数据已经接收完成
    }


    if(1==Gu8FinishFlag)  //1 代表已经接收完毕一串新的数据，需要马上去处理
    {
        if(0==Gu8CurrentReceBuffer_Sec)
        {
            Gu8CurrentReceBuffer_Sec=1; //以最快的速度先切换接收内存，避免丢失新发过来的数据
```

```
        //Gu32ReceCnt_B=0;//这里不能清零缓存 B 的计数器，意味深长，避免此处临界点发生中断
        Gu8FinishFlag=0;  //尽可能以最快的速度清零本次完成的标志，为下一次新数据做准备
        pSu8ReceBuffer=(unsigned char *)&Gu8ReceBuffer_A[0]; //关联刚刚接收的数据缓存
        Su32ReceSize=Gu32ReceCnt_A; //记录当前缓存的有效字节数
        Gu32ReceCnt_A=0; //及时把当前缓存计数清零，为一次切换接收缓存做准备。意味深长。
    }
    else
    {
        Gu8CurrentReceBuffer_Sec=0; //以最快的速度先切换接收内存，避免丢失新发过来的数据
        //Gu32ReceCnt_A=0;//这里不能清零缓存 A 的计数器，意味深长，避免此处临界点发生中断
        Gu8FinishFlag=0;  //尽可能以最快的速度清零本次完成的标志，为下一次新数据做准备
        pSu8ReceBuffer=(unsigned char *)&Gu8ReceBuffer_B[0]; //关联刚刚接收的数据缓存
        Su32ReceSize=Gu32ReceCnt_B; //记录当前缓存的有效字节数
        Gu32ReceCnt_B=0; //及时把当前缓存计数清零，为一次切换接收缓存做准备。意味深长。
    }

    //Gu8FinishFlag=0; //之所以不选择在这里清零，是因为在上面清零更及时快速。意味深长。

    //开始处理刚刚接收到的一串新数据，直接“统一”处理 pSu8ReceBuffer 指针为代表的数据即可
    for(i=0;i<Su32ReceSize;i++)
    {
         if(0x02==pSu8ReceBuffer[i]&&
            0x03==pSu8ReceBuffer[i+1]&&
            0x04==pSu8ReceBuffer[i+2]) //连续三个数是 0x02 0x03 0x04
          {
              vGu8BeepTimerFlag=0;
              vGu16BeepTimerCnt=100;   //让蜂鸣器“短叫”100ms
              vGu8BeepTimerFlag=1;
              return; //直接退出当前函数
          }

         if(0x06==pSu8ReceBuffer[i]&&
            0x07==pSu8ReceBuffer[i+1]&&
            0x08==pSu8ReceBuffer[i+2]&&
            0x09==pSu8ReceBuffer[i+3]) //连续四个数是 0x06 0x07 0x08 0x09
          {
              vGu8BeepTimerFlag=0;
              vGu16BeepTimerCnt=2000;   //让蜂鸣器“长叫”2000ms
              vGu8BeepTimerFlag=1;
              return; //直接退出当前函数
          }

    }
```

```
    }
}

void T0_time() interrupt 1
{
    VoiceScan();

    if(1==vGu8ReceTimeOutFlag&&vGu16ReceTimeOutCnt>0) //通信过程中字节之间的超时定时器
    {
         vGu16ReceTimeOutCnt--;
    }

    TH0=0xfc;
    TL0=0x66;
}


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)
{

}

void BeepOpen(void)
{
    P3_4=0;
}

void BeepClose(void)
{
    P3_4=1;
}

void VoiceScan(void)
{

      static unsigned char Su8Lock=0;

      if(1==vGu8BeepTimerFlag&&vGu16BeepTimerCnt>0)
      {
          if(0==Su8Lock)
          {
               Su8Lock=1;
               BeepOpen();
          }
          else
          {

               vGu16BeepTimerCnt--;

               if(0==vGu16BeepTimerCnt)
```

```
                {
                    Su8Lock=0;
                    BeepClose();
                }

            }
        }
}
```