第一百二十八节： 接收“固定协议”的串口程序框架。

【128.1　固定协议。】

实际项目中，串口一个回合的收发数据量远远不止 1 个字节，而是往往携带了某种“固定协议”的一串数据（专业术语称“一帧数据”）。一串数据的“固定协议”因为起到类似“校验”和“密码确认”的功能，因此在安全可靠性方面大大增强。但是上一节也提到，单片机利用最底层硬件的串口接口，一次收发的最小单位是“1 个字节”，那么，怎么样在此基础上搭建一个能快速收发并且能快速解析数据的程序框架就显得尤为重要。本节我跟大家分享我常用的串口程序框架，此框架主要包含“数据头，数据类型，数据长度，其它数据”这四部分。比如，为了通过串口去控制单片机的蜂鸣器发出不同长度的声音，我专门制定了一串十六进制的数据：EB 01 00 00 00 08 03 E8 ，下面以此串数据来跟大家详细分析。

数据头（EB）：占 1 个字节，作为“起始字节”，起到“接头暗号”的作用，平时用来过滤无关的数据。只有“接头暗号”吻合，单片机才会进入到接收其它有效数据的步骤，否则一直被“数据头”挡在门外视为无效数据。注意，数据头不能用十六进制的 00 或者 FF，因为 00 和 FF 的密码等级太弱，很多单片机一上电的瞬间因为硬件的某种不确定的原因，会直接误发送 00 或者 FF 这类干扰数据。

数据类型（01）：占用 1 个字节。数据类型是用来定义这串数据的用途，比如，01 代表用来控制蜂鸣器的，02 代表控制 LED 的，03 代表机器启动，等等功能，都可以用这个字节的数据进行分类定义。本例子用 01 代表控制蜂鸣器发出不同时间长度的声音。

数据长度（00 00 00 08）：占 4 个字节。用来告诉通信的对方，这串数据一共有多少个字节。本例子中，数据长度占用了 4 个字节，就意味着最大数据长度是一个 unsigned long 类型的数据范围，从 0 到 4294967295。比如，本例子中一串数据的长度是 8 个字节（EB 01 00 00 00 08 03 E8 ），因此这“数据长度”四个字节分别是 00 00 00 08，十六进制的 08 代表十进制的 8 字节。注意，51 单片机的内存是属于大端模式，因此十进制的 8 在四字节 unsigned long 的内存排列顺序是 00 00 00 08，也就是低位放在数组的高下标。如果是 stm32 的单片机，stm32 单片机的内存是属于小端模式，十进制的 8 在四字节 unsigned long 的内存排列顺序是 08 00 00 00，低位放在数组的低下标。为什么强调这个？因为主要方便我们用指针的方法实现数据的拆分和整合，这个知识点的内容我在前面第 62 节详细讲解过。

其它数据（03 E8）：此数据根据不同的“数据类型”可以用来做不同的用途，根据具体的项目而定。本例子十六进制的 03 E8，代表一个 unsigned int 的十进制数据 1000。此数据的大小用来控制蜂鸣器发声的长度，1000 代表长叫 1000ms。如果想让蜂鸣器短叫 100ms，只需把这两个字节改为：00 64。

【128.2　程序框架的四个要点分析。】

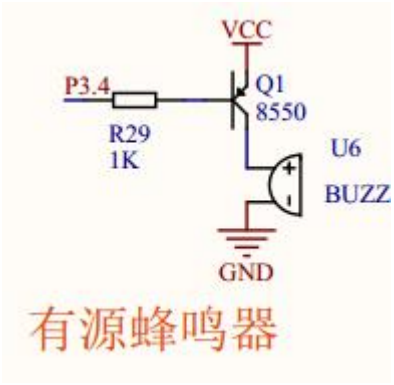
第一点：先接收后处理，开辟一块专用的内存数组。要处理一串数据，必须先征用一块内存数组专门用来缓存接收到的数据，等接收完此串数据再处理。

第二点：接头暗号。本节例子的数据头 EB 就是接头暗号。一旦接头暗号吻合，才会进入到下一步接收其它有效数据的步骤上。

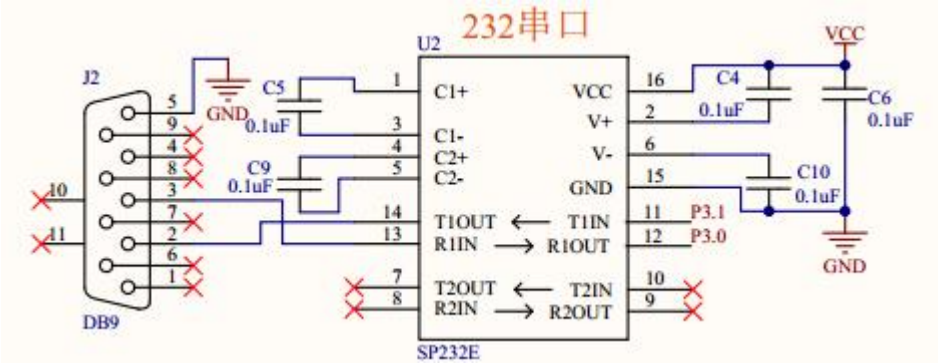
第三点：如何识别接收一串数据的完毕。本节例子中，是靠“固定协议”提供的“数据长度”来判别是否已经接收完一串数据。中断函数接收完一串数据后，应该用一个全局变量来给外部 main 函数一个通知，让 main 函数里面的相关函数来处理此串数据。

第四点：接收数据中相邻字节之间通信超时的异常处理。如果接头暗号吻合之后，马上切换到“接受其它有效数据”的步骤，但是，如果在此步骤的通信过程中一旦发现通信不连贯，就应该及时退出当下“接受其它有效数据”的步骤，继续返回到刚开始的“接头暗号”的步骤，为下一次接收新的一串数据做准备。那么，如何识别通信不连贯？靠判断接收数据中相邻字节之间的时间是否超时来决定，详细内容请看下面的程序例程。

【128.3　程序例程。】



上图 128.3.1　有源蜂鸣器电路



上图 128.3.2　232 串口电路

程序功能如下：

（1）在上位机的串口助手里，发送一串数据，控制蜂鸣器发出不同长度的声音。

（2）波特率 9600，校验位 NONE（无），数据位 8，停止位 1。

（3）十六进制的数据格式如下：

EB 01 00 00 00 08 XX XX

其中 EB 是数据头，01 是代表数据类型，00 00 00 08 代表数据长度是 8 个（十进制）。XX XX 代表一个 unsigned int 的数据，此数据的大小决定了蜂鸣器发出声音的长度。比如：

让蜂鸣器鸣叫 1000ms 的时间，发送十六进制的： EB 01 00 00 00 08 03 E8

让蜂鸣器鸣叫 100ms 的时间，发送十六进制的： EB 01 00 00 00 08 00 64

```
#include "REG52.H"

#define RECE_TIME_OUT    2000  //通信过程中字节之间的超时时间 2000ms
#define REC_BUFFER_SIZE  20    //接收数据的缓存数组的长度
```

```
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;

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

unsigned char Gu8ReceBuffer[REC_BUFFER_SIZE]; //开辟一片接收数据的缓存
unsigned long Gu32ReceCnt=0;  //接收缓存数组的下标
unsigned char Gu8ReceStep=0;  //接收中断函数里的步骤变量
unsigned char Gu8ReceFeedDog=1; //“喂狗”的操作变量。
unsigned char Gu8ReceType=0; //接收的数据类型
unsigned long Gu32ReceDataLength=0;  //接收的数据长度
unsigned char Gu8FinishFlag=0;  //是否已接收完成一串数据的标志
unsigned long *pu32Data; //用于数据转换的指针
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

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

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。

/* 注释一：
* 以下 Gu8FinishFlag 变量的用途。
* 此变量一箭双雕，0 代表正处于接收数据的状态，1 代表已经接收完毕并且及时通知主函数中的处理函数
* UsartTask()去处理新接收到的一串数据。除此之外，还起到一种“自锁自保护”的功能，在新数据还
* 没有被主函数处理完毕的时候，禁止接收其它新的数据，避免新数据覆盖了尚未处理的数据。
*/
       if(0==Gu8FinishFlag)  //1 代表已经完成接收了一串新数据，并且禁止接收其它新的数据
       {

/* 注释二：
* 以下 Gu8ReceFeedDog 变量的用途。
* 此变量是用来检测并且识别通信过程中相邻的字节之间是否存在超时的情况。
* 如果大家听说过单片机中的“看门狗”这个概念，那么每接收到一个数据此变量就“置 1”一次，它的
* 作用就是起到及时“喂狗”的作用。每接收到一个数据此变量就“置 1”一次，在主函数里，相关
* 的定时器就会被重新赋值，只要这个定时器能不断及时的被补充新的“能量”新的值，那么这个定时器
* 就永远不会变成 0，只要不变成 0 就不会超时。如果两个字节之间通信时间超过了固定的长度，就意味
* 着此定时器变成了 0，这时就需要把中断函数里的接收步骤 Gu8Step 及时切换到“接头暗号”的步骤。
*/
          Gu8ReceFeedDog=1; //每接收到一个字节的数据，此标志就置 1 及时更新定时器的值。
          switch(Gu8ReceStep)
          {
              case 0:     //接头暗号的步骤。判断数据头的步骤。
                   Gu8ReceBuffer[0]=SBUF; //直接读取刚接收完的一个字节的数据。
                   if(0xeb==Gu8ReceBuffer[0])  //等于数据头 0xeb，接头暗号吻合。
                   {
                      Gu32ReceCnt=1; //接收缓存的下标
                      Gu8ReceStep=1;  //切换到下一个步骤，接收其它有效的数据
                   }
                   break;

              case 1:     //数据类型和长度
                   Gu8ReceBuffer[Gu32ReceCnt]=SBUF; //直接读取刚接收完的一个字节的数据。
                   Gu32ReceCnt++; //每接收一个字节，数组下标都自加 1，为接收下一个数据做准备
                   if(Gu32ReceCnt>=6)  //前 6 个数据。接收完了“数据类型”和“数据长度”。
                   {
                        Gu8ReceType=Gu8ReceBuffer[1];  //提取“数据类型”
                         //以下的数据转换，在第 62 节讲解过的指针法
                        pu32Data=(unsigned long *)&Gu8ReceBuffer[2]; //数据转换
                        Gu32ReceDataLength=*pu32Data; //提取“数据长度”
                        if(Gu32ReceCnt>=Gu32ReceDataLength) //靠“数据长度”来判断是否完成
```

```
                        {
                            Gu8FinishFlag=1; //接收完成标志“置 1”，通知主函数处理。
                            Gu8ReceStep=0;   //及时切换回接头暗号的步骤
                        }
                        else   //如果还没结束，继续切换到下一个步骤，接收“其它数据”
                        {
                            Gu8ReceStep=2;   //切换到下一个步骤
                        }
                    }
                    break;
                case 2:     //其它数据
                    Gu8ReceBuffer[Gu32ReceCnt]=SBUF; //直接读取刚接收完的一个字节的数据。
                    Gu32ReceCnt++; //每接收一个字节，数组下标都自加 1，为接收下一个数据做准备

                    //靠“数据长度”来判断是否完成。也不允许超过数组的最大缓存的长度
                    if(Gu32ReceCnt>=Gu32ReceDataLength||Gu32ReceCnt>=REC_BUFFER_SIZE)
                    {
                       Gu8FinishFlag=1; //接收完成标志“置 1”，通知主函数处理。
                       Gu8ReceStep=0;   //及时切换回接头暗号的步骤
                    }
                    break;
            }
        }
    }
    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 int *pSu16Data; //数据转换的指针
    static unsigned int Su16Data;  //转换后的数据

    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;

        vGu8ReceTimeOutFlag=0;
       vGu16ReceTimeOutCnt=RECE_TIME_OUT;//更新一次“超时检测的定时器”的初值
        vGu8ReceTimeOutFlag=1;
```

```
    }
    else if(Gu8ReceStep>0&&0==vGu16ReceTimeOutCnt) //超时，并且步骤不在接头暗号的步骤
    {
        Gu8ReceStep=0; //串口接收数据的中断函数及时切换回接头暗号的步骤
    }


    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
    {
        switch(Gu8ReceType)  //接收到的数据类型
        {
            case 0x01:   //驱动蜂鸣器
                  //以下的数据转换，在第 62 节讲解过的指针法
                  pSu16Data=(unsigned int *)&Gu8ReceBuffer[6]; //数据转换。
                  Su16Data=*pSu16Data; //提取“蜂鸣器声音的长度”

                  vGu8BeepTimerFlag=0;
                  vGu16BeepTimerCnt=Su16Data;   //让蜂鸣器鸣叫
                  vGu8BeepTimerFlag=1;
                  break;
        }

        Gu8FinishFlag=0;  //上面处理完数据再清零标志，为下一次接收新的数据做准备
    }
}



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();
                }

          }
      }
}
```