第一百三十节： 接收带“动态密匙”与“异或”校验数据的串口程序框架。

【130.1 “异或”的校验。】

通信的校验常用有两种，一种是“累加和”，另一种是“异或”。“异或”算法的详细介绍请看前面章节的第 32 节。

上一节讲的“累加和”，放在数据串的最后一个字节，是前面所有字节的累加之和（不包括自己本身的字节），累加的结果高于一个字节的那部分自动溢出丢掉，只保留低 8 位的一个字节的数据。本节讲的“异或”，也是放在数据串的最后一个字节，是前面所有字节的异或结果（不包括自己本身的字节）。本节在上一节的基础上，只更改以下这段校验算法的代码即可。

上一节的“累加和”算法如下：

```
Gu8ReceZZ=Gu8ReceBuffer[Gu32ReceDataLength-1];  //提取“累加和”

Su8RecZZ=0;
for(i=0;i<(Gu32ReceDataLength-1);i++)
{
    Su8RecZZ=Su8RecZZ+Gu8ReceBuffer[i];   //计算“累加和”
}

if(Su8RecZZ==Gu8ReceZZ) //验证“累加和”，“计算的”与“接收的”是否一致
{
    //此处省去若干代码
}
```

本节的“异或”算法如下：

```
Gu8ReceZZ=Gu8ReceBuffer[Gu32ReceDataLength-1];  //提取接收到的“异或”

Su8RecZZ=Gu8ReceBuffer[0]; //提取数据串第“i=0”个数据作为异或的原始数据
for(i=1;i<(Gu32ReceDataLength-1);i++) //注意，这里是从第“i=1”个数据开始
{
    Su8RecZZ=Su8RecZZ^Gu8ReceBuffer[i];   //计算“异或”
}

if(Su8RecZZ==Gu8ReceZZ) //验证“异或”，“计算的”与“接收的”是否一致
{
    //此处省去若干代码
}
```

【130.2 通信协议。】

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。

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。

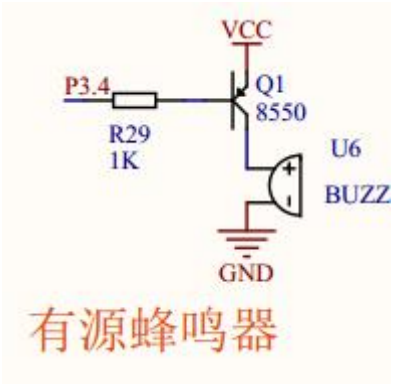
数据长度（00 00 00 0B）：占 4 个字节。用来告诉通信的对方，这串数据一共有多少个字节。

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。

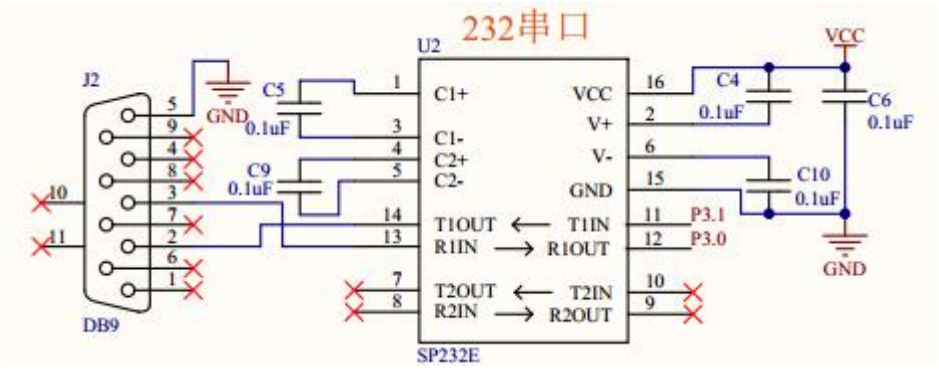
动态密匙（00 01）：这两个字节代表一个 unsigned int 类型的数据，数据范围是从 0 到 65535，但是考虑到数据更加安全可靠，一般丢弃了首尾的 0（十六进制的 00 00）与 65535（十六进制的 FF FF），只保留从 1 到 65534 的变化。大部分的通信模型都是主机对从机的“一问一应答”模式，也就是，主机每发送一条指令给从机，从机才返回一条消息作为应答。如果主机发送了信息后，在规定的时间内，没有收到从机的应答指令，主机就继续发送信息给从机，但是此时，从机本来应该应答主机当前指令的，可能因为某种情况导致反馈的信息发生了延时，导致此时应答的数据是主机的上一条指令，从而造成“一问一应答”的数据帧发送了错位，这种情况加上“动态密匙”就能使问题得到有效的解决。主机每发送一条信息，信息里都携带了 2 个字节的“动态密匙”，从机每收到主机的一条信息，在应答此信息时都把收到的“动态密匙”原封不动的反馈给主机，主机再查看发送的“动态密匙”与接收到的“动态密匙”是否一致，以此来判断应答数据是否有效。“动态密匙”像流水号一样，每发送一次指令后都累加 1，不断发生变化，从 1 到 65534，依次循环。这是数据校验的一种方式。

异或（0B）。“异或”放在数据串的最后一个字节，是前面所有字节的异或结果（不包括自己本身的字节）。比如：本例子中，数据串是：EB 01 00 00 00 0B 03 E8 00 01 0B。其中最后一个字节 0B 就是“异或”字节，前面所有字节相“异或”等于十六进制的 0B。验证“异或”的方法，可以借用电脑“附件”自带的“计算器”软件来实现，打开“计算器”软件后，在“查看”的下拉菜单里，选择“程序员”，然后选择“十六进制”，该计算器软件的异或运算按键是“Xor”。不管是主机还是从机，每接收到一串数据后，都要自己计算一次“异或”，把自己计算得到的“异或”与接收到的最后一个字节的“异或”进行对比，来判断接收到的数据是否发生了丢失或者错误。

【130.3　程序例程。】



上图 130.3.1　　有源蜂鸣器电路



上图 130.3.2　　232 串口电路

程序功能如下：

（1）单片机模拟从机，上位机的串口助手模拟主机。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）本节因为还没有讲到数据发送的内容，因此应答“动态密匙”那部分的代码暂时不写，只写验证“异或”那部分的代码。

（3）波特率 9600，校验位 NONE（无），数据位 8，停止位 1。

（4）十六进制的数据格式：EB 01 00 00 00 0B XX XX YY YY ZZ 。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）。

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。

YY YY 代表一个 unsigned int 的动态密匙，每收发一条指令，此数据累加一次 1，范围从 1 到 65534。

ZZ 代表前面所有字节的异或结果。

比如：

让蜂鸣器鸣叫 1000 毫秒，密匙为 00 01，发送十六进制的：EB 01 00 00 00 0B 03 E8 00 01 0B

让蜂鸣器鸣叫 100 毫秒， 密匙为 00 02，发送十六进制的：EB 01 00 00 00 0B 00 64 00 02 87

```
#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 int Gu16ReceYY=0; //接收的动态密匙
unsigned char Gu8ReceZZ=0; //接收的异或
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
    static unsigned int i;
    static unsigned char Su8RecZZ=0;  //计算的“异或”



    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[Gu32ReceDataLength-3]; //数据转换
                  Gu16ReceYY=*pSu16Data; //提取“动态密匙”。本例子中暂时不做返回应答的处理

                  Gu8ReceZZ=Gu8ReceBuffer[Gu32ReceDataLength-1];  //提取接收到的“异或”

                  Su8RecZZ=Gu8ReceBuffer[0]; //提取数据串第“i=0”个数据作为异或的原始数据
                  for(i=1;i<(Gu32ReceDataLength-1);i++) //注意，这里是从第“i=1”个数据开始
                  {
                      Su8RecZZ=Su8RecZZ^Gu8ReceBuffer[i];   //计算“异或”
                  }

                  if(Su8RecZZ==Gu8ReceZZ) //验证“异或”，“计算的”与“接收的”是否一致
                  {
                      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();
               }

          }
      }
}
```