一步一步写一个短消息收发协议栈（2）

——基于 TC35i 和 ATMega32 的短消息协议栈 FreeSmsStack

V1.1

bpesun@163.com

1. 说明

第 1 个文档的目的是让大家尽快完成 FreeSmsStack 的移植，并且在自己的项目中使用它。本文档的目的是详细描述协议栈的实现。我尽量把编写这个协议栈的思路和步骤来一步一步描述清楚。

2. 目的

本项目的目的是完成一个建立在 TC35i 模块上的短消息协议栈。我给这个协议栈起的名字是 FreeSmsStack。从名字上可以看出，这个协议栈是一个免费的开源协议栈。

短消息业务（SMS）作为 GSM 的一种增值服务，随着 GSM 网络覆盖范围的不断扩大，得到了迅速发展，它具有传输速度快，费用低，不占用语音通信通道等优点，因而在远程智能控制系统中得到了广泛的应用，如：基于 GSM 和 GPS 的车辆跟踪监视系统，基于 GSM 的远程 LED 信息发布系统等。

3. FreeSmsStack 协议栈的功能

目前，这个协议栈能完成如下的功能：

- ✓ 中、英文短信发送
- ✓ 中、英文短信接收
- ✓ 短信删除
- ✓ 振铃后挂断来电并且反馈短信到来电号码
- ✓ 普通 AT 命令发送

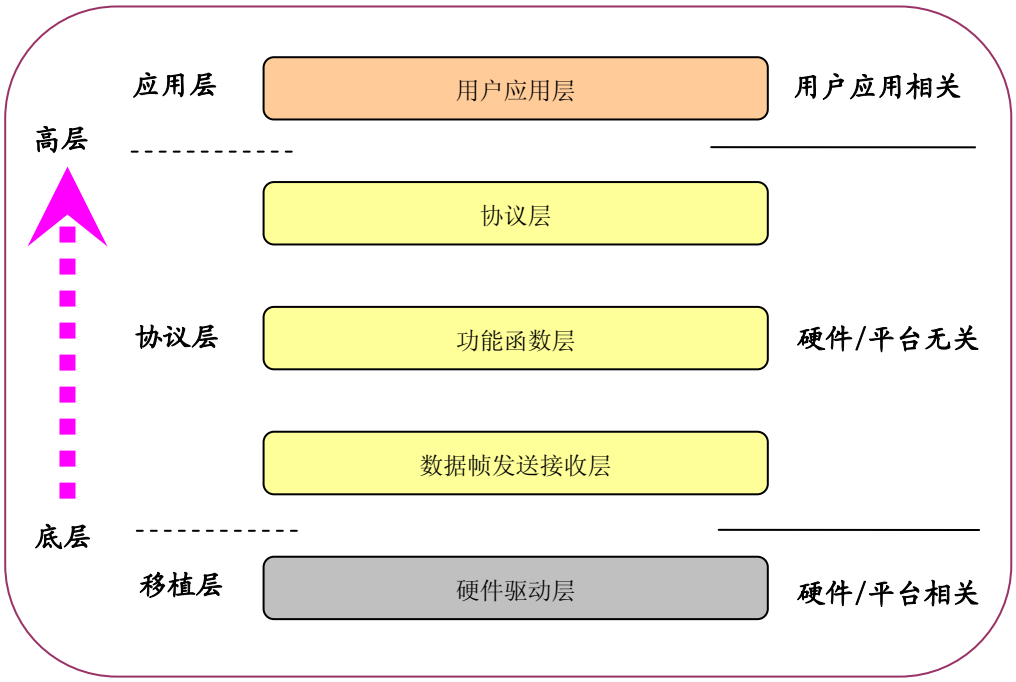
注意：目前，中文短信编码不能通过单片机实现，只能通过查表的方式将某些短信编码存储在单片机中。

4. 协议栈实现过程

如果你有兴趣，可以看看这个 FreeSmsStack 是怎么一步一步编写出来的。这个协议栈其实是参考了多个人的劳动成果，并不是从底层一点一点敲出来的，而是一开始就有一个整体的框架了。

这个框架在逻辑上从底层到高层分为：

- ✓ 硬件驱动层（微控制器串口发送和接收）
- ✓ 数值帧发送接收层（符合 AT 命令的数据帧的发送和接收）
- ✓ 功能函数层（完成短消息的读、写和删除）
- ✓ 协议层（短消息协议栈的开始、停止、删除等）
- ✓ 用户应用层（用户根据项目的实际功能要求，增加自己的应用或处理函数）



但是，为了好理解，我们按照下面的顺序来将框架分解，来慢慢解释。

- ➢ 采用中断方式的串口数据发送和接收
- ➢ 发送 AT 命令数据帧
- ➢ 接收 AT 命令数据帧
- ➢ 发送短消息功能函数
- ➢ 接收短消息功能函数
- ➢ 删除短消息功能函数
- ➢ 实时短消息接收的实现
- ➢ 协议栈完善
- ➢ 用户功能函数添加

4.1. 采用中断方式的串口数据发送和接收

这儿需要实现的是硬件驱动层的函数。由于该层与所采用的平台和硬件相关，因此这部分内容也是属于移植层。

需要实现的功能包括：串口的初始化、串口发送和接收的使能和禁止、串口发送中断服务、串口接收中断服务。

4.1.1. 底层硬件相关的移植函数

vSmsPortSerialEnable 函数说明如表所示。

函数	void vSmsPortSerialEnable(BOOL xRxEnable, BOOL xTxEnable)
功能描述	串口发送和接收的使能和禁止
参数	BOOL xRxEnable 接收使能，TURE 允许，FALSE 禁止 BOOL xTxEnable 发送使能，TURE 允许，FALSE 禁止
返回值	无
特殊说明和注意点	半双工工作

xSmsPortSerialInit 函数说明如表所示。

函数	BOOL xSmsPortSerialInit(UCHAR ucPORT, ULONG ulBaudRate, UCHAR ucDataBits, eSmsParity eParity)
功能描述	串口初始化函数
参数	UCHAR ucPORT //串口号 ULONG ulBaudRate //波特率 UCHAR ucDataBits //数据位 eSmsParity eParity //奇偶校验位
返回值	TURE
特殊说明和注意点	串口初始化完成后，串口的发送和接收是禁止的，需要调用 vSmsPortSerialEnable 来打开或者关闭发送和接收。

xSmsPortSerialInit 函数说明如表所示。

函数	BOOL xSmsPortSerialPutByte(CHAR ucByte)
功能描述	串口发送一个字节的数据
参数	CHAR ucByte 需要发送的一个字节数据
返回值	TURE
特殊说明和注意点	只是给发送寄存器赋值，不去等待是否发送完成。在发送中断函数中，如果发送完成一个字符后，就调用这个函数向发送寄存器赋一个新的需要发送的数据。

xSmsPortSerialGetByte 函数说明如表所示。

函数	BOOL xSmsPortSerialGetByte(CHAR * pucByte)
功能描述	串口接收一个字节的数据
参数	CHAR * pucByte 返回参数，接收到的数据存放地址
返回值	TURE
特殊说明和注意点	接收中断发生后，先调用这个函数获取接收寄存器的数据

4.1.2. 串口发送中断函数

数据帧的发送是采用中断方式的，因此需要在发送中断函数中完成数据的发送。为了方便移植，发送中断服务函数写成了一个通用的发送函数。在移植层我们只需要调用这个通用发送函数即可。具体的，针对 AVR 微控制器的实现代码如下所示。

```
SIGNAL( SIG_USART_DATA )
{
    xSmsFrameTransmitCallBack(    );                    //数据帧发送回调函数
}
```

其中，函数 xSmsFrameTransmitCallBack(　);是一个通用的发送函数，叫做数据帧发送回调函数。该函数在 SmsFrame.c 文件中实现，具体如下所示：

```
BOOL
xSmsFrameTransmitCallBack( void )
{
    BOOL    xNeedPoll = FALSE;

    assert( eRcvState == STATE_RX_IDLE );

    switch ( eSndState )
    {
        /* We should not get a transmitter event if the transmitter is in idle state.    */
    case STATE_TX_IDLE:
        /* enable receiver/disable transmitter. */
        vSmsPortSerialEnable( TRUE, FALSE );                    //可以接收的状态
        break;

    case STATE_TX_XMIT:
        /* check if we are finished. */
        if( usSndBufferCount != 0 )
        {
            xSmsPortSerialPutByte( ( CHAR )*pucSndBufferCur );
            pucSndBufferCur++;
            usSndBufferCount--;
        }
        else
        {
            /* Disable transmitter. This prevents another transmit buffer empty interrupt. */
            vSmsPortSerialEnable( TRUE, FALSE );    //发送完成，设置为接收状态
            eSndState = STATE_TX_IDLE;
        }
        break;
    }
    return xNeedPoll;
}
```

该函数根据发送数据帧的当前状态进行处理，如果当前处于发送空闲状态，就调用xSmsPortSerialPutByte((CHAR)*pucSndBufferCur);将当前数据发送出去。如果所有数据都发送完成了，就禁止发送，使能接收，并且转入发送空闲状态。

其中的 xSmsPortSerialPutByte（）函数是发送一个数据函数，跟硬件相关，需要针对不同的平台进行相应移植。

4.1.3. 串口接收中断函数

数据帧的接收也是采用中断方式，即在接收中断函数中实现数据帧的接收。为了方便移植，接收中断服务函数写成了一个通用的发送函数。在移植层我们只需要调用这个通用接收函数即可。具体的，针对 AVR 微控制器的实现代码如下所示。

```
SIGNAL( SIG_USART_RECV )
{
    xSmsFrameReceiveCallBack(    );
}
```

其中，函数 xSmsFrameReceiveCallBack(   );是一个通用的接收函数，叫做数据帧接收回调函数。该函数在 SmsFrame.c 文件中实现，具体如下所示：

```
BOOL
xSmsFrameReceiveCallBack( void )
{
    UCHAR               ucByte;

    assert( eSndState == STATE_TX_IDLE );

    /* Always read the character. */
    ( void )xSmsPortSerialGetByte( ( CHAR * ) & ucByte );

    switch ( eRcvState )
    {
        /* In the error state we wait until all characters in the
          * damaged frame are transmitted.
          */
    case STATE_RX_ERROR:
        usRcvBufferPos = 0;
        break;

          /* We are currently receiving a frame. Reset the timer after
          * every character received. If more than the maximum possible
          * number of bytes in a modbus frame is received the frame is
          * ignored.
          */
    case STATE_RX_RCV:
        if( usRcvBufferPos < SMS_SER_FRAME_SIZE_MAX )
        {
```

```
            ucSmsFrameBuf[usRcvBufferPos++] = ucByte;
        }
        else
        {
            eRcvState = STATE_RX_ERROR;
        }
        break;

    case STATE_RX_HEADER:
        if( SMS_FRAME_DEFAULT_LF == ucByte )    //接收到换行
        {

            /* Reset the input buffers to store the frame. */
            usRcvBufferPos = 0;
            eRcvState = STATE_RX_RCV;
        }
        break;

    case STATE_RX_IDLE:
        if( SMS_FRAME_DEFAULT_CR == ucByte )    //接收到回车
        {
            eRcvState = STATE_RX_HEADER;
        }
        break;

    default: break;
    }
    vSmsPortTimersEnable(    );
    return 1;
}
```

函数中首先调用(void)xSmsPortSerialGetByte((CHAR *) & ucByte);获取接收到的数据。然后根据接收状态进行处理，当收到回车和换行符号后进入接收中状态STATE_RX_RCV，在此状态下将数据接收到接收缓冲区中。如果接收数据长度过长的话，会进入错误状态。

接收数据帧还需要定时中断的配合才能完成数据帧的接收。每接收到一个字符发生接收中断后，都会使能定时器。当定时器发生中断后，会根据当前所处的接收状态判断是否收到了有效数据帧。该定时中断如下所示。

```
BOOL
xSmsFrameTimerT1SExpiredCallBack( void )
{
    BOOL                xNeedPoll = FALSE;
    switch ( eRcvState )
    {
```

```
            /* A frame was received and t35 expired. Notify the listener that
              * a new frame was received. */
        case STATE_RX_RCV:
            ucSmsFrameBuf[usRcvBufferPos] = '\0';   //增加一个字符结束标志
            bReceivedData = TRUE;       //一段时间后收到了字符，认为接收字符结束
            break;

            /* An error occured while receiving the frame. */
        case STATE_RX_ERROR:
            break;

            /* Function called in an illegal state. */
        default:
            assert( ( eRcvState == STATE_RX_RCV ) || ( eRcvState == STATE_RX_ERROR ) );
        }

        vSmsPortTimersDisable(  );
        eRcvState = STATE_RX_IDLE;

        return xNeedPoll;
    }
```

定时中断发生后，如果接收状态处于 STATE_RX_RCV，则认为收到了一个有效的数据帧，并且转入接收空闲状态。如果接收状态处于其他状态，则停止定时器并转入接收空闲状态。

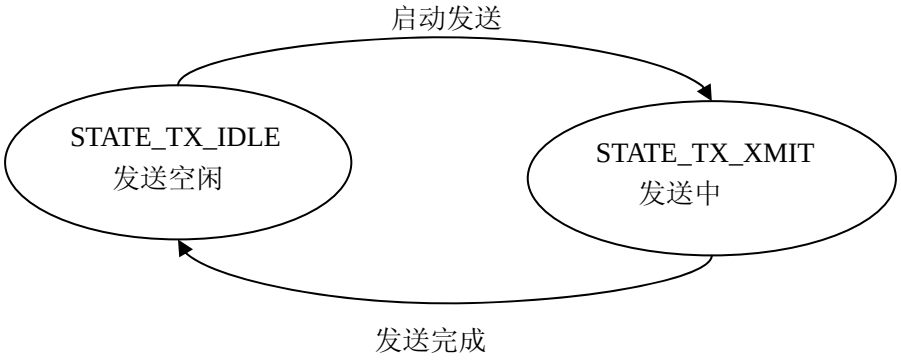
4.2. 发送 AT 命令数据帧

采用中断方式，发送数据帧。

发送和接收数据分别使用了 2 个小的状态机。需要注意的是发送和接收不能同时运行。

发送状态机用来发送一条 AT 指令。

由于发送过程中是禁止掉接收的，并且发送采用的是中断方式，所以只有空闲和发送中两个状态。默认状态是发送空闲状态，当启动发送后，转入发送中状态。发送完成后，转入发送空闲状态。



具体的发送过程可以参考上一节中关于发送中断函数的描述。

通过分析发送的 AT 指令，我们可以提出一种替代方案。先看看发送的 AT 命令有什么特点：

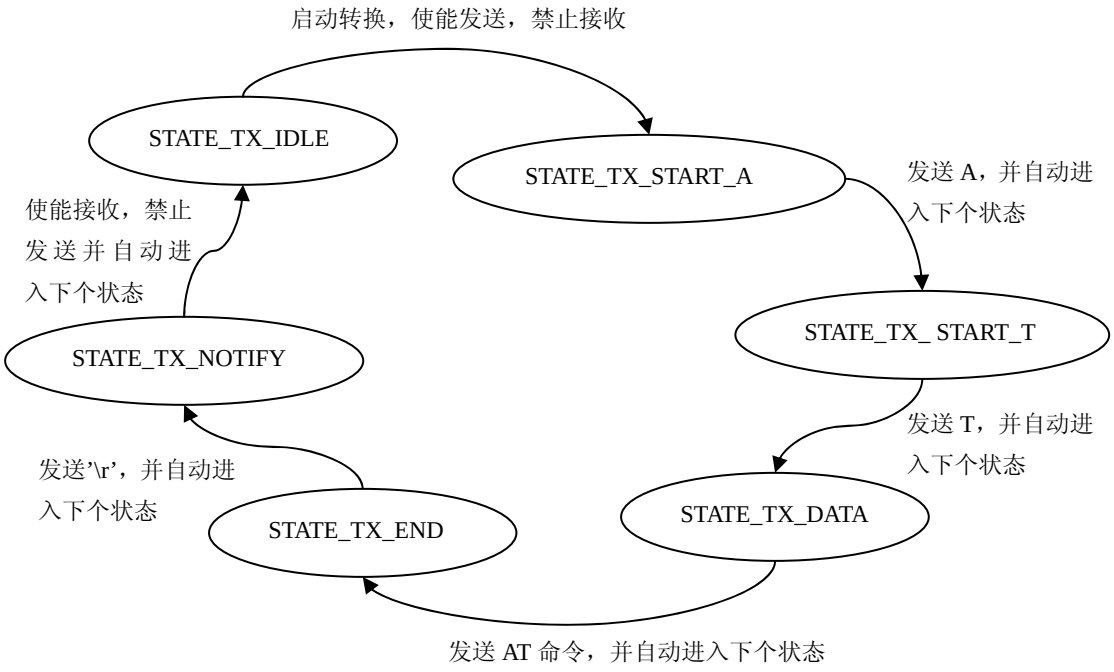
1，以 AT 作为数据帧开头

2，以'\r'作为数据帧结束标志

根据上面的特点，可以重新设计一个状态机，用枚举类型表示如下：

```
typedef enum
{
    STATE_TX_IDLE,              //空闲状态
    STATE_TX_START_A,           //发送 A
    STATE_TX_START_T,           //发送 T
    STATE_TX_DATA,              //发送具体的命令
    STATE_TX_END,               //发送回车
    STATE_TX_ NOTIFY            //发送完成
} eMBSndState;
```

用图表示如下：



特例

发送短消息时

4.3. 接收 AT 命令数据帧

微控制器跟 TC35i 模块采用 AT 命令进行通信，所以最小的有效数据就是一条从 TC35i 模块反馈的有效的 AT 反馈指令。接收数据帧的过程就是接收一条 AT 反馈指令的过程。

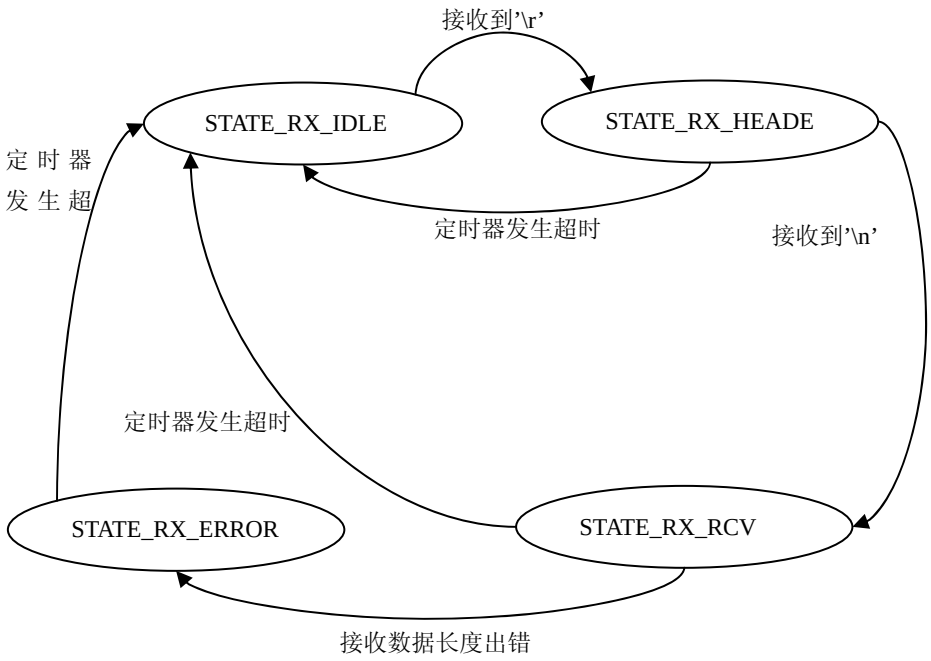
通过分析 AT 指令的反馈格式，我们发现 AT 反馈指令有如下的特点：

1，以’\r\n’为数据帧（AT 反馈指令）的开头，并且以其作为数据帧（AT 反馈指令）结尾

2，返回的数据中间可能夹杂着’\r\n’

因此，我们可以设置接收状态如下枚举类型所示：

```
typedef enum
{
    STATE_RX_IDLE,          //空闲状态
    STATE_RX_HEADER,        //接收到头
    STATE_RX_RCV,           //接收有效数据
    STATE_RX_ERROR          //接收出错
} eSmsRcvState;
```



具体的发送过程可以参考上一节中关于发送中断函数的描述。

上面描述了如何发送和接收 AT 指令。利用这些函数，我们就可以发送我们需要的 AT 指令，用这些 AT 指令完成用户需要的操作，如读写短信、拨打电话功能。

4.4. 发送短消息功能函数

发送短信的过程如下：

首先，发送 AT 命令：AT+CMGS=<length><CR>

接着，等待"\r\n>"提示符

最后，收到该提示符后输入编码后的短信内容，并且以 Ctrl+Z 结束，Esc 放弃本次发送。

该函数的具体实现如下所示：

```
eSmsErrorCode
eSmsSend( char *pObjNo,  CHAR *pContent, USHORT usLen, eSmsEncodeType eCodemode )
{
    CHAR sms_buffer[13];
    CHAR temp[5];
   USHORT usCodeLen;
   USHORT i;
    CHAR *pout;


   pout = (char *)ucSmsFrameBuf;  //还是利用数据帧缓冲区来做发送的缓冲区
   // 求发送的 PDU 数据包的长度
   // 需要编码
   usCodeLen=usSmsCreatePduPackage( pObjNo, pContent, pout, usLen, eCodemode);

   pout[usCodeLen] = 26;        // CTRL+Z 表示短消息结束
   pout[usCodeLen+1] = '\0';    //或者=0 结束字符串

   usCodeLen = usCodeLen/2;
   usCodeLen = usCodeLen - 1;  // 中心号码长度现在只使用短信中心长度字节(00)

   i = ucByte2String10( usCodeLen, temp);                              // 信息长度

   temp[i]='\r';
   temp[i+1]='\0';                    // 注意加字符串结束符

   i=0;
   strcpy(sms_buffer,"AT+CMGS=");
   strcat(sms_buffer,temp);

   while(1)
   {
      // 等待接收”>"符号
      if( SMS_RETURN_OK != eSmsSendATCommand(sms_buffer, ">", "No > receive!", 0, 4))  // 无法发送,延时后重新发送
      {
```

```
            if(++i>3) return 0;                                    // 3 次都无法发送，放弃
        }
        else
        {
            break;
        }
        UCHAR t;
        for( t = 0; t < 50; t++ )
            _delay_ms(10);              // 0.5s 后重新发送
        wdt_reset();
    }

    //由于接收会用到缓冲区，所有，还需要重新进行一次编码
    pout = (char *)ucSmsFrameBuf; //还是利用数据帧缓冲区来做发送的缓冲区
    //  求发送的 PDU 数据包的长度
    //  需要编码
    usCodeLen=usSmsCreatePduPackage( pObjNo, pContent, pout, usLen, eCodemode);

    pout[usCodeLen] = 26;               // CTRL+Z 表示短消息结束
    pout[usCodeLen+1] = '\0';

    return eSmsSendATCommand(pout,"+CMGS:","Send Message Error!",0,30);
}
```

4.5. 接收短消息功能函数

接收短消息的过程如下：

发送 AT+CMGS=<Position><CR>

等待回应短信相关信息和具体的短信内容，如：+CMGR: 0,,26\r\n0891.....

上面回应中，\r\n 后面的数据是有效的短信内容，我们需要将其取出并进行解码，得到具体的短信内容。

该函数具体的实现如下所示：

```
eSmsErrorCode
xSmsRead( SMSSTRUCT *pSMSPayload, char *pSave, UCHAR nPos, UCHAR NoCode)
{
     CHAR *paddr;
    UCHAR i;
    //SMSSTRUCT    *pStruct=NULL;
    eSmsErrorCode       eStatus = SMS_ENOERR;
    if( pSave != NULL)
    {//  设置存储空间
        strcpy( (char *)ucSmsFrameBuf, "AT+CPMS=" );
        strcat( (char *)ucSmsFrameBuf, pSave );
        strcat( (char *)ucSmsFrameBuf, "\r" );
```

```
        i = eSmsSendATCommand( (char *)ucSmsFrameBuf, "+CPMS: ", "CPMS Error\r\n", 0, 2 );

        if( SMS_RETURN_OK != i )
        {
            eStatus = SMS_UNKNOWNERR;
            return eStatus;
        }
    }

    // 读取短信
    strcpy( (char *)ucSmsFrameBuf, "AT+CMGR=" );
    ucByte2String10( nPos, (char *)&ucSmsFrameBuf[strlen( (char *)ucSmsFrameBuf )] );
    strcat( (char *)ucSmsFrameBuf, "\r" );

    i = eSmsSendATCommand( (char *)ucSmsFrameBuf, "+CMGR: ", "Read SMS Error\r\n", &paddr, 2 );

    if( SMS_RETURN_OK == i )
    {// 成功，应答的格式为 +CMGR: 0,,26\r\n0891..... 或者是 +CMGR: 0,,26\r\n\r\nOK\r\n
        for(i=0;i<8;i++)
        {
            if( ( *paddr ) == '\n')
            {
                paddr++;
                break;
            }
            else paddr++;
        }
        if( *( paddr ) == '\r' )
        {
            eStatus = SMS_UNKNOWNERR;
            Prints("<No message!>");
            return eStatus;
        }
        if(i==8)
        {
            eStatus = SMS_UNKNOWNERR;
            Prints( "<Read Err>" );
            return eStatus;
        }
    }
```

```
    else
    {// 读短信时发生错误
        eStatus = SMS_UNKNOWNERR;
        Prints( "Read Err" );
        return eStatus;
    }
    // 接收并解码短信
    xSmsReceive( pSMSPayload, paddr, NoCode );

    if(pSMSPayload == NULL)
    {
        eStatus = SMS_UNKNOWNERR;
        Prints( "SMS Struct Error!\r\n" );
    }
    return eStatus;
}
```

函数会将短信读出，并且将解码后的内容放入结构体 SMSSTRUCT *pSMSPayload 中。

4.6. 删除短消息功能函数

删除短信的实现如下所示。

```
eSmsErrorCode
eSmsDelete(    CHAR *pSave, UCHAR nPos )
{
    UCHAR i;
     CHAR nBuff[5];

    if(pSave!=NULL)
    {// 设置存储空间
        strcpy( (char *)ucSmsFrameBuf, "AT+CPMS=" );
        strcat( (char *)ucSmsFrameBuf, pSave );
        strcat( (char *)ucSmsFrameBuf, "\r" );

        i = eSmsSendATCommand( (char *)ucSmsFrameBuf, "+CPMS: ", "CPMS
Error\r\n", 0, 4 );

        if(i != SMS_RETURN_OK)
        {
            return i;
        }
    }

    ucByte2String10( nPos, nBuff );
```

```
    strcpy( (char *)ucSmsFrameBuf, "AT+CMGD=" );
    strcat( (char *)ucSmsFrameBuf, nBuff );
    strcat( (char *)ucSmsFrameBuf, "\r" );

   return eSmsSendATCommand( (char *)ucSmsFrameBuf, "OK", "Delete SMS Error\r\n",
0, 6);
}
```

4.7. 实时短消息接收的实现

短消息的接收有 2 种方式。其一是采用查询的方式，不断发送 AT+CMGL=1 指令来查询是否收到了短消息，如果收到了，就能将短信的内容依次列出。

另外一种方式类似实时接收，将模块设置为短信自动通知模式，当短信到达的时候，模块自动发出短信到达指示。协议栈不断查询是否有短信指示，如果有，就分析出短信存储位置，并且调用读短信函数来读取并分析该短信。

具体地，协议栈轮询函数中，不断查询是否收到新数据。如果收到了，就调用 vSmsRecvURCProcess（）函数来分析具体的数据内容。这些数据内容指示了短信模块的状态，可能的内容包括：短信到达指示、电压指示、呼叫指示、短信满指示等。如果是短信指示，则调用读短信函数来实现短信的读取和解码。

协议栈轮询函数如下：

```
eSmsErrorCode
eSmsPoll( void )
{
    static CHAR     *ucSmsFrame;
    static USHORT     usLength;

    eSmsErrorCode       eStatus = SMS_ENOERR;

    /* Check if the protocol stack is ready. */
    if( eSmsState != STATE_ENABLED )
    {
        return SMS_EILLSTATE;
    }

    if( bSerialGetString( &ucSmsFrame, &usLength ) )
    {
      ucSmsFrame[usLength] = 0;
      vSmsRecvURCProcess(    ucSmsFrame );
    }

    return eStatus;
}
```

接收到数据后的处理函数如下：

```
void vSmsRecvURCProcess( CHAR *pBuffer)
```

```
{
    CHAR *pRecv;
    UCHAR i,t,u,nIndex;

    SMSSTRUCT xUserSmsStruct;                    //需要为用户数据分别一个存储空间？
    SMSSTRUCT *pStruct;
    pStruct = &xUserSmsStruct;                   //定义的指针指向该结构体首地址

    USHORT len = strlen(pBuffer);

    pRecv = pBuffer;
    /*(过了一段时间，有一条消息到达)
    +CMTI: "ME",3\r\n(通知：消息已经存储在 ME 内存中，序号为 8)*/

    if(StrInclude(pRecv,"+CMTI:"))
    {
        //eSmsFrameSend( "Rcv SMS!", 8 );
        pRecv += 12;

        for(i=0;i<len;i++)
        {
            if( pRecv[i] == '\r' )
            {
                pRecv[i] = '\0';
                break;
            }
        }

        nIndex = ucString2Byte( pRecv );        //转换为 10 进制的数据

        if( ( SMS_UNKNOWNERR == xSmsRead( pStruct, NULL, nIndex, 0 ) )||( pStruct
== NULL ) )
        {
            Prints("<Recv SMS Err>");
            return;
        }

        // 执行命令
        eSmsDataProcess(pStruct);

        // 删除短信
        for( u = 1; u <= nIndex; u++ )
        {
```

```
                for( i = 1; i < 5; i++    )
                {
                    if( SMS_RETURN_OK == eSmsDelete( NULL, u ) )
                    {
                        //Prints("Send_SMS_OK");
                        break;
                    }
                    Prints("DEL_SMS_err");
                    for( t = 0 ; t < 10; t++ )
                    {
                        _delay_ms(50);
                        wdt_reset();
                    }
                }
            }

        }
        else if(StrInclude(pRecv,"^SBC: Undervoltage"))
        {//  电压低
            eSmsFrameSend( "V Low!", 6 );
        }
        else if(StrInclude(pRecv,"^SBC:"))
        {//  电压高
            eSmsFrameSend( "V High!", 7 );
        }
        else if(StrInclude(pRecv,"^SMGO:"))
        {//  短信存储空间满
            eSmsFrameSend( "Storage Full!", 13 );
            for(i = 0; i < 30; i++)
                eSmsDelete( NULL, i );
        }
        else if(StrInclude(pRecv,"RING"))
        {//  呼叫
            //eSmsFrameSend( "Ring..", 6 );
            vSmsCallIn();
        }
        else
        {
            return;
        }
}
```

4.8. 用户应用函数添加

用户应用程序可能包括的功能如下：
1，发送短信，用户根据设备状态发送一条短信。
2，接收短信处理，收到短信后，进行分析处理
3，接到电话处理，收到来电后，进行分析处理。
我们按上面的三种功能，依次说明如何添加用户应用程序。

4.8.1. 发送短信

发送短信最简单，程序先根据设备状态组织一条短信内容，然后调用 SmsSend 函数将短信内容发送到目标号码即可。

4.8.2. 接收短信处理

当协议栈检测到有短信到达后，协议栈会调用首先调用 xSmsRead ()来读取并解码短消息，解码后短消息放入短消息结构体 SMSSTRUCT *pStruct 种，然后调用 eSmsDataProcess(pStruct);来处理该短消息。用户需要将自己的应用程序在 eSmsDataProcess() 函数中实现。

4.8.3. 接到电话处理

当协议栈检测到有电话时，协议栈会调用 vSmsCallIn()函数来处理呼叫。该函数会首先获取来电号码，然后调用 UserCallInCallBack（）函数。用户应用程序需要在该函数内实现。

参考资料
http://tech.ddvip.com/program/vc/network/index_3.html