一步一步写一个短消息收发协议栈（1）

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

V1.0

bpesun@163.com

1. 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。

2. 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。

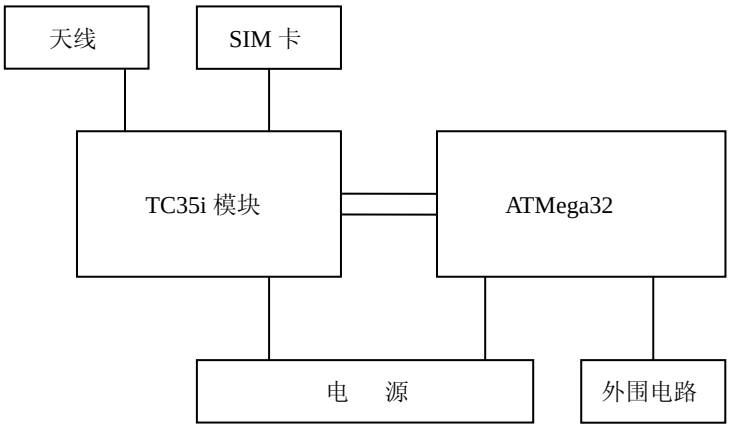
3. FreeSmsStack 协议栈对硬件的需求

下面列出的是本协议栈在所有功能使能的情况下对单片机的需求，可以看出普通的中档次的单片机都能满足要求。

- ✓ 具有一个串口，具备发送寄存器空中断和接收到中断
- ✓ 具有一个定时器
- ✓ RAM 最好有 1.5K 以上（实现全部功能）

3.1. 项目硬件

项目的硬件结构如下图所示。主要由 GSM 模块 TC35i、单片机 ATmega32、电源等模块组成。单片机和 TC35i 模块之间通过 TTL 串口进行通信。



下面简单介绍一下项目中所用到的硬件。

3.2. TC35i

短消息模块采用西门子的 TC35i。该模块的特性如下。

特 性	说 明
信息传送内容	语音和数据
电源	单电源 3.3V ～ 4.8V
频段	双频 GSM900MHz 和 DCS1800 MHz(Phase 2+)
发射功率	2W （GSM900MHz Class 4） 1W （DCS1800MHz Class 1）
SIM 卡连接方式	外接
天线	由天线连接器连接外部天线
温度范围	工作温度：-20°C to +55°C 储存温度：-30°C to +85°C
工作电流损耗	通话模式: 300mA (典型值.) 空闲模式: 3.0mA (最大值) 省电模式: 50μA (最大值)
短信息	MT, MO, CB 和 PDU 模式
外型尺寸	54.5 x 36 x 3.6mm
通讯接口	RS232（指令和数据的双向传送）
SIM 卡操作电压	3V/1.8V
电话薄功能	存储于 SIM 卡中
模块复位	采用 AT 指令或掉电复位
串口通讯波特率	300bps...115kbps
自动波特率范围	4.8kbps...115kbps

TC35i 模块有 40 个引脚，通过一个 ZIF(Zero Insertion Force，零阻力插座)连接器引出。这 40 个引脚可以划分为 5 类，即电源、数据输入/输出、SIM 卡、音频接口和控制。

TC35i 的第 1～5 引脚是正电源输入脚通常推荐值 4.2V，第 6～10 引脚是电源地。11、12 为充电引脚，可以外接锂电池，13 为对外输出电压(共外电路使用)，14 为 ACCU-TEMP 接负温度系数的热敏电阻，用于锂电池充电保护控制。

15 脚是启动脚 IGT，系统加电后为使 TC35i 进入工作状态,必须给 IGT 加一个大于 100ms 的低脉冲,电平下降持续时间不可超过 1ms。

16～23 为数据输入/输出，分别为 DSR0、RING0、RxD0、TxD0、CTS0、RTS0、DTR0 和 DCD0。tc35i 模块的数据输入/输出接口实际上是一个串行异步收发器，符合 ITU-T RS232 接口标准。它有固定的参数：8 位数据位和 1 位停止位，无校验位，波特率在 300bps~115kbps 之间可选，默认 9600。硬件握手信号用 RTS0/CTS0，软件流量控制用 XON/XOFF，CMOS 电平，支持标准的 AT 命令集。

其中 18 脚 RXD、19 脚 TXD 为 TTL 的串口通讯脚，需要和单片机或者 PC 通讯。

TC35i 使用外接式 SIM 卡, 24～29 为 SIM 卡引脚，SIM 卡同 TC35i 是这样连接的:SIM 上的 CCRST、CCIO、CCCL、CCVCC 和 CCGND 通过 SIM 卡阅读器与 TC35i 的同名端直接相连，ZIF 连接座的 CCIN 引脚用来检测 SIM 卡是否插好，如果连接正确，则 CCIN 引脚输出高电平，否则为低电平。

TC35i 的第 32 脚 SYNC 引脚有两种工作模式，一种是指示发射状态时的功率增长情况，另一种是指示 TC35i 的工作状态，可用 AT 命令 AT+SYNC 进行切换，本模块使用的是后一种。当 LED 熄灭时,表明 TC35i 处于关闭或睡眠状态；当 LED 为 600 ms 亮/600ms 熄时，表明 SIM 卡没有插入或 TC35i 正在进行网络登录；当 LED 为 75 ms 亮/3s 熄时，表明 TC35i 已登录进网络，处于待机状态。

30、31、32 脚为控制脚，其中 30 为 RTC backup，31 为 Power down，32 为 SYNC。

35～38 为语音接口，35、36 接扬声器放音。37、38 可以直接接驻极体话筒来采集声音（37 是话筒正端，39 是话筒负端）

单片机通过两根 I/O 口控制 TC35 的开关机、复位等，通过串口与 TC35 进行数据通信。本项目通信速率为 4800bps，采用 8 位异步通讯方式，1 位起始位，8 位数据位，1 位停止位。

TC35 模块输入输出的 TTL 正电平逻辑不是+5V，而是+2.9V，因此必要时加端口保护。

3.3. 电源

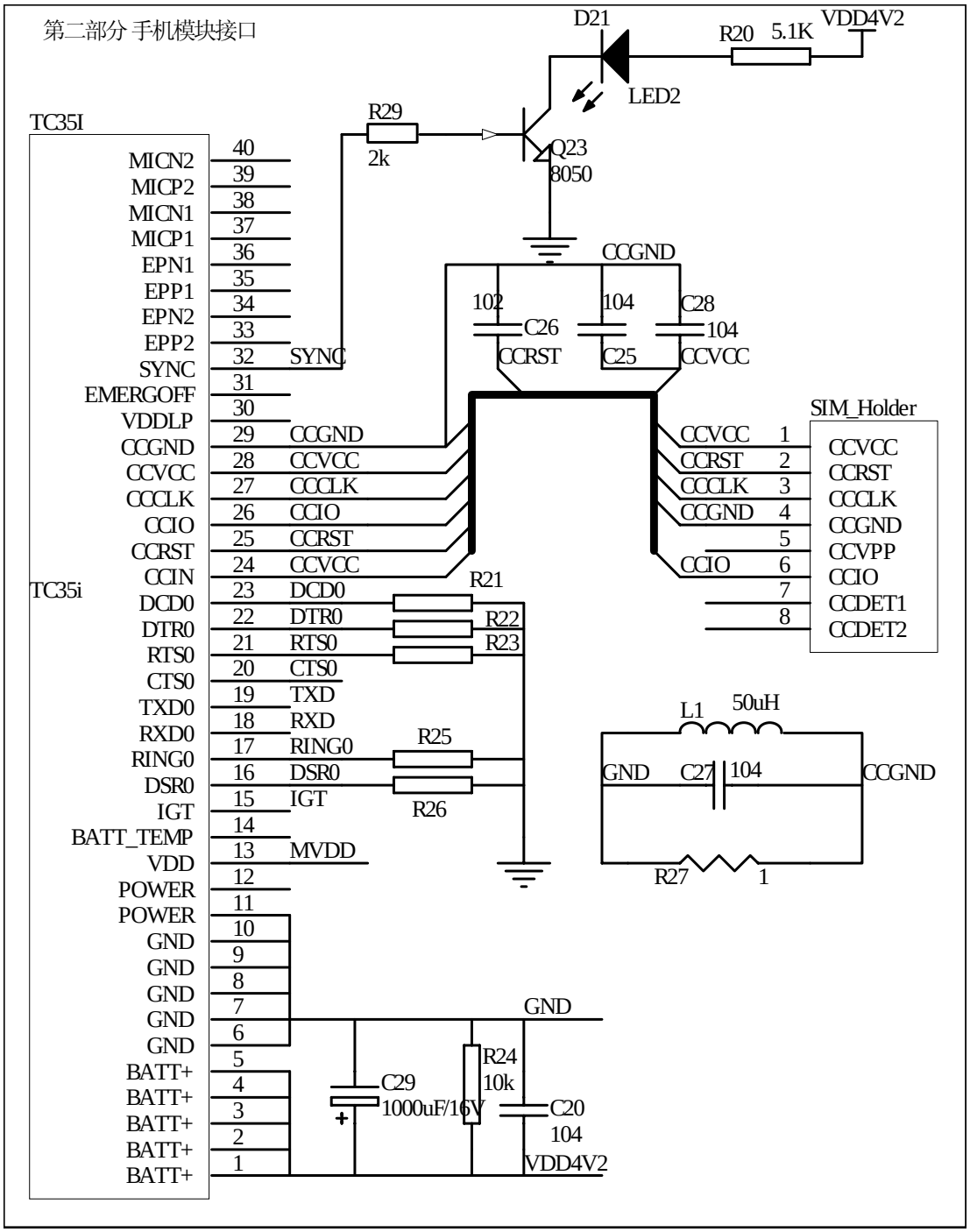
模块的供电电压如果低于 3.3V 会自动关机。同时模块在在发射时，电流峰值可高达 2A。同时在此电流峰值时，电源电压（送入模块的电压）下降值不能超过 0.4V。所以该模块对电源的要求较高，理想工作电压是 4.2V。

我们这里只列出几种经过验证的开关电源方案，具体的实施原理可以参考数据手册或者网上的资料。第一，使用可调 DC-DC 电源芯片 LM2941S。第二，使用 DC-DC 控制器，如 R1224N102E。

3.4. 具体连接示例

下图是我在项目中使用的 TC35i 连接图。主要的连接包括：电源连接、SIM 卡连接、SYNC 信号灯、串口连接（**TXD 和 RXD 分别接单片机的 TXD 和 RXD**），以及一些辅助的电阻连接。IGT 是点火控制信号，MVDD 可以用来监测模块是否启动。图中未注明的管脚

可以悬空。



3.5. 调试串口

为了方便调试，可以增加一个调试串口。如果选择的单片机是具有双串口的单片机，那么可以将第二个串口作为调试串口，通过 MAX3232 等电平转换芯片变成 RS232 电平，连接 PC 串口。如果你选择的单片机只有一个串口，也没有问题，只需要将串口复用（单片机的串口既连接 TC35i 又连接电平转换芯片），同样通过一个电平转换芯片进行转换并连接到 PC。编程的过程中，我们可以将一些调试信息通过这个串口输出到我们的串口助手上，方便我们调试。

串口调试软件网上有好多，但是我推荐使用 SSCOM32。大家可以在网上搜索并下载。

4. 熟悉AT 命令

4.1. 先看 AT 命令的语法

准备好硬件，就可以进行我们的协议栈的编写了。在编写之前，我们先了解一下微控制器跟 TC35i 进行通信所使用的 AT 命令。如果你以前没有接触过 AT 命令的化，这一部分内容需要仔细看一下，记得很久以前，那时候手机还不是很多，我们的一个项目想使用手机短信来控制家电，让我做项目预研。那个时候刚刚接触微控制器不久，并且还没有养成好的学习习惯，所以在网上匆匆查了些资料就开始“动手做东西”。那时候我的硬件是一个 NOKIA 的手机，要通过手机数据线连接 PC。按照资料，先做一个无源的 TTL 到 RS232 转换器，然后连接 PC。最后，通过串口助手测试 AT 命令。看资料上写着发送“AT”应该回复“OK”，于是敲“AT”进去，可怎么也不见“OK”回来。于是又怀疑是硬件问题，又去重新做硬件连线，然后再试验，仍然没有数据反馈回来。然后拿其他命令来试验，仍然不行。折腾几天，仍然没有结果，领导只好取消这个项目。当几年后再学习 AT 命令的时候，突然发现了我当时的问题所在：还没有弄清楚 AT 命令是怎么样的结构就去盲目地试验——我们发送 AT 命令的时候，需要用“\r”做为结束符号，而我当年发的所有的 AT 命令都没有加“\r”这个符号，又怎么可能有“OK”回来哪？惭愧啊，一个很好的项目也由于我的低级错误而被扼杀了。

AT 命令语法

还是先看几个例子吧。

AT\r

\r\nOK\r\n

我们发送 AT 握手信号，以“\r”为结束符号。

TC35i 模块反馈一个“OK”，“OK”的前后都是“\r\n”字符。

下面是在网上找的几个例子，省略了发送时候的“\r”，每个开始新行的地方起始都省略了“\r\n”。你可以看着官方的 AT 命令手册来看具体的含义，或者你暂时不去弄懂也可以，而是在编程的时候再去参考一下 AT 指令手册。

例如下面的命令：

AT+CNMI=?　(查看能支持的设置范围)

+CNMI: (0-2),(0-3),(0,2,3),(0,1),(0,1)\r\n

OK

将上面的命令补齐后，看起来应该如下所示：

AT+CNMI=?\r　(查看能支持的设置范围)

\r\n+CNMI: (0-2),(0-3),(0,2,3),(0,1),(0,1)\r\n

\r\nOK\r\n

AT+CNMI?　　(查看当前设置)

+CNMI: 0,0,0,0,0
OK

AT+CNMI=2,1 (设置为 mode=2, mt=1)
OK

AT+CNMI?　　(再查看当前设置)
+CNMI: 2,1,0,0,0
OK

(过了一段时间，有一条消息到达)
+CMTI “ME”,8　(通知：消息已经存储在 ME 内存中，序号为 8)

AT+CMGR=8　　(读第 8 条消息)
+CMGR: 8,27
0891683108200505F0240D91683158812764F800004020521103738007 41E19058341E01
OK

你如果有短消息模块的话，用户可以通过串口助手来试验这些 AT 命令。仔细分析命令和应答之间的特殊字符（\r\n）。如果没有的话，就记住下面的 AT 命令语法。

- ✓ 所有的 AT 命令都以“AT”开头
- ✓ 根据命令形式可以将 AT 命令分为：
  - ■ 测试命令，形式为“AT+C***=?”，执行该命令将返回该命令所支持的参数及参数范围；
  - ■ 读命令，形式为“AT+C***?”，执行该命令将返回该命令当前的参数值；
  - ■ 写命令，形式为“AT+C***=<….>”，执行该命令将设置该命令的参数值；
  - ■ 无参数执行命令，形式为“AT+C***”，这个是执行命令。
- ✓ 发送的命令以 AT 开头，以“\r”结尾。如握手信号“AT\r”，关闭回显命令“ATE0\r”
- ✓ TC35i 反馈的命令以“\r\n”开头，也以“\r\n”结尾，但是，反馈的命令串中间也可能夹杂着\r\n。

在本文中：
\r 代表 ASCII 字符中的回车字符，值为 0x0D；
\n 代表 ASCII 字符中的换行字符，值为 0x0A；
SP 代表 ASCII 字符中的空格字符，值为 0x20；
Ctrl+Z 代表 ASCII 字符中的文件结束字符，值为 0x1A；

下面我们再认真弄懂下面几个 AT 命令，这些命令是我们协议栈必须实现的功能函数。

4.2. 发送短信命令

下面给出一个采用 PDU 方式发送中文短信 B01234568。
第一步，首先需要向 TC35i 发送下面的 AT 命令：
AT+CMGS=33\r

第二步，如果 TC35i 运行正确，它会反馈如下字符：\r\n>SP

第三步，收到>后，需要向 TC35i 输入短信编码信息：

0011000D91683125312011F1000800120042003000310032003300340035003600 38Ctrl+Z

第四步，如果 TC35i 接收成功，会反馈如下字符：

\r\n+CMGS: 89\r\n

\r\nOK\r\n

总结一下

发送命令：AT+CMGS=<length><CR>

应答命令：接收到该命令后等待"\r\n>SP"提示符，收到该提示符后输入短信，以 Ctrl+Z 结束，Esc 放弃本次发送。

4.3. 接收短信命令

下面的内容摘自网上，我觉得我自己的表述肯定不如下面的这个描述清楚。具体的出处为：

通过串口实时接收短消息 选择自 bhw98 的 Blog

关键字 短消息,短信,串口,手机,AT

出处 http://dev.csdn.net/article/24/24861.shtm

本协议使用的接收是一种通过串口“实时”接收短消息的方法。当 ME 收到一条消息时，主动发出通知给 TE，或者直接将消息转发到 TE。与查询机制相比，它类似于中断机制。

先简要说明一下短消息类(class)的概念：根据指定储存的位置，短消息分为 class 0 – 3 四个类。也可以不指定类(no class)，由 ME 按默认设置进行处理，存储到内存或者 SIM 卡中。在 TPDU 的 TP-DCS 字节中，当 bit7-bit4 为 00x1，01x1，1111 时，bit1-bit0 指出消息所属类：

00 – class 0：只显示，不储存

01 – class 1：储存在 ME 内存中

02 – class 2：储存在 SIM 卡中

03 – class 3：直接传输到 TE

GSM Modem 一般都支持一条“AT+CNMI”指令，可用于设定当有某类短消息到达时，如何处置它：只储存在指定的内存(易失的/非易失的)中，先储存后通知 TE，还是直接转发到 TE，等等。

“AT+CNMI”指令语法为

AT+CNMI=[<mode>[,<mt>[,<bm>[,<ds>[,<bfr>]]]]]

mode - 通知方式：

0 – 不通知 TE。

1 – 只在数据线空闲的情况下，通知 TE；否则不通知 TE。

2 – 通知 TE。在数据线被占用的情况下，先缓冲起来，待数据线空闲，再行通知。

3 – 通知 TE。在数据线被占用的情况下，通知混合在数据中一起传输。

mt - 消息储存或直接转发到 TE：

0 – 储存到默认的内存位置(包括 class 3)

1 – 储存到默认的内存位置，并且向 TE 发出通知(包括 class 3)

2 – 对于 class 2，储存到 SIM 卡，并且向 TE 发出通知；对于其它 class，直接将消息转发到 TE

3 – 对于 class 3，直接将消息转发到 TE；对于其它 class，同 mt=1

bm, ds, bfr 的含义，请参考相关标准文档。一般不需要去关心它们。

在程序中具体实现时，使用 mode=2, mt=1，比较简单。对所有类型的短消息，只要在收到 ME 送来的“+CMTI”通知后，用“AT+CMGR”指令读取消息内容就行了。TE 与 ME 之间的通信过程，举例如下：

(初始化)

AT+CNMI=? (查看能支持的设置范围)

+CNMI: (0-2),(0-3),(0,2,3),(0,1),(0,1)

OK

AT+CNMI? (查看当前设置)

+CNMI: 0,0,0,0,0

OK

AT+CNMI=2,1 (设置为 mode=2, mt=1)

OK

AT+CNMI? (再查看当前设置)

+CNMI: 2,1,0,0,0

OK

(过了一段时间，有一条消息到达)

\r\n+CMTI “ME”,8\r\n (通知：消息已经存储在 ME 内存中，序号为 8)

我们接收到上面的提示信息后，进行解析，我们需要读第 8 条短消息，发出的具体的 AT 命令如下啊：

AT+CMGR=8\r (读第 8 条消息)

TC35i 将进行反馈，输出第 8 条短消息的具体内容如下：

\r\n+CMGR: 8,27\r\n

0891683108200505F0240D91683158812764F800004020521103738007 41E19058341E01\r\n

\r\nOK\r\n

4.4. 删除短信命令

以删除第 1 条短信为例子。

第一步，发送删除命令：AT+CMGD=1\r

第二步，如果删除成功，TC35i 反馈的指令为：\r\nOK\r\n

5. *FreeSmsStack 协议栈移植*

看到这儿好多人估计已经没有耐心了。那么，我们现在把源代码放出来，如果你的硬件已经准备好了，并且有一个可以使用的 SIM 卡，就可以先将 FreeSmsStack 移植到你的目标板上，看看短信的收发功能是否正常了。一旦成功了，成就感就能出来一点，让你更有兴趣来深入了解 FreeSmsStack 协议栈具体的实现过程。

5.1. 硬件控制管脚相关移植

如果你正好使用 ATmega32 微控制器，那么软件的移植就只是改变一条语句定义，即把点火信号的管脚重新定义一下，来适合你的硬件。

具体的修改在文件 SmsPortSerialMega32.c 文件中。里面有针对 IGT 管脚的定义。

5.2. 硬件相关的移植文件

如果你换了微控制器，除了完成上面的硬件控制管脚的移植外，你还要完成串口移植和时钟移植，这 2 个移植分别对应着 2 个不同的 c 语音源文件：SmsPortSerialMega32.c、SmsPortTimer.c。

5.2.1. 串口移植文件

串口部分的函数是最底层的串口硬件驱动层。实现的功能包括：串口初始化、串口的使能、发送一个字符、接收一个字符、发送中断函数和接收中断函数。发送和接收的过程都是通过中断方式进行的。

1）串口初始化函数 *xSmsPortSerialInit*，完成波涛率、数据格式的定义。数据帧层在初始化的时候会调用本函数，完成真正的串口的初始化。具体功能见下面的注释。

```
/*********************************************************************
* 名      称:xSmsPortSerialInit
* 功能描述:串口初始化
* 输入参量:UCHAR ucPORT                                    //串口号
           ULONG ulBaudRate                                //波特率
           UCHAR ucDataBits                                //数据位
           eSmsParity eParity                              //奇偶校验位
* 输出参量:无
* 返      回:是否初始化成功
* 调用子程:
* 使用方法: 数据帧层在初始化的时候会调用本函数实现硬件的初始化
---------------------------------------------------------------------*/
BOOL
 xSmsPortSerialInit( UCHAR ucPORT, ULONG ulBaudRate, UCHAR ucDataBits, eSmsParity
                     eParity )
{
    UCHAR ucUCSRC = 0;

    /* prevent compiler warning. */
    (void)ucPORT;

    UBRR = UART_BAUD_CALC( ulBaudRate, F_CPU );
```

```
    switch ( eParity )                                          //奇偶校验设置
    {
        case SMS_PAR_EVEN:
            ucUCSRC |= _BV( UPM1 );
            break;
        case SMS_PAR_ODD:
            ucUCSRC |= _BV( UPM1 ) | _BV( UPM0 );
            break;
        case SMS_PAR_NONE:
            break;
    }

    switch ( ucDataBits )                                       //数据位设置
    {
        case 8:
            ucUCSRC |= _BV( UCSZ0 ) | _BV( UCSZ1 );
            break;
        case 7:
            ucUCSRC |= _BV( UCSZ1 );
            break;
    }

#if defined (__AVR_ATmega168__)
    UCSRC |= ucUCSRC;
#elif defined (__AVR_ATmega169__)
    UCSRC |= ucUCSRC;
#elif defined (__AVR_ATmega8__)
    UCSRC = _BV( URSEL ) | ucUCSRC;
#elif defined (__AVR_ATmega16__)
    UCSRC = _BV( URSEL ) | ucUCSRC;
#elif defined (__AVR_ATmega32__)
    UCSRC = _BV( URSEL ) | ucUCSRC;
#elif defined (__AVR_ATmega128__)
    UCSRC |= ucUCSRC;
#endif

    vSmsPortSerialEnable( FALSE, FALSE );                       //禁止串口
    return TRUE;
}
```

2）接收一个字符函数 *xSmsPortSerialGetByte*，接收串口寄存器的一个字节数据。串口接收中断函数中会调用这个函数来接收一个字符。具体功能见下面的注释。

```
/*************************************************************
* 名      称:xSmsPortSerialGetByte
* 功能描述:接收一个字节数据
* 输入参量:无
* 输出参量:CHAR * pucByte  接收到的数据存放地址
* 返      回:TRUE
* 调用子程:无
* 使用方法:串口接收中断函数中会调用这个函数来接收一个字符
-------------------------------------------------------------*/
BOOL
xSmsPortSerialGetByte( CHAR * pucByte )
{
    *pucByte = UDR;
    return TRUE;
}
```

3）发送一个字符函数 *xSmsPortSerialPutByte*，发送一个字节数据到串口寄存器中。串口接收中断函数中会调用这个函数来发送一个字符。具体功能见下面的注释。

```
/*************************************************************
* 名      称:xSmsPortSerialPutByte
* 功能描述:发送一个字节的数据
* 输入参量:CHAR ucByte     需要发送的一个字节数据
* 输出参量:无
* 返      回:TRUE
* 调用子程:
* 使用方法:
-------------------------------------------------------------*/
BOOL
xSmsPortSerialPutByte( CHAR ucByte )
{
    UDR = ucByte;
    return TRUE;
}
```

4）发送寄存器空中断服务函数 *SIGNAL(SIG_USART_DATA)*，发送寄存器空中断发生后，系统会自动调用本中断服务函数。本服务函数会调用数据帧层的数据帧发送函数，完成一个数据帧的发送。具体功能见下面的注释。

```
/*************************************************************
* 名      称:SIGNAL( SIG_USART_DATA )
* 功能描述:发送中断服务函数，当数据帧处理函数判断接收完一个数据帧后，
            发送数据帧发送完成事件标志。
* 输入参量:无
```

```
* 输出参量:无
* 返　　回:无
* 调用子程:调用发送数据帧回调函数xSmsFrameTransmitFSM(　)，函数在SmsStack.h
            中声明，在SmsFrame.c 中实现定义
* 使用方法:
-----------------------------------------------------------------*/
SIGNAL( SIG_USART_DATA )
{
    xSmsFrameTransmitFSM(　);                                    //回调函数
}
```

5）串口接收缓冲寄存器接收到数据中断服务函数 *SIGNAL(SIG_USART_RECV)*，接收寄存器接收到字符中断发生后，系统会自动调用本中断服务函数。本服务函数会调用数据帧层的数据帧接收函数，完成一个数据帧的接收。具体功能见下面的注释。

```
/*****************************************************************
* 名　　称:SIGNAL( SIG_USART_RECV )
* 功能描述:接收中断服务函数
* 输入参量:无
* 输出参量:无
* 返　　回:无
* 调用子程:调用接收数据帧回调函数xSmsFrameReceiveFSM(　)，该函数在在SmsStack.h
            中声明，在SmsFrame.c 中实现定义
* 使用方法:
-----------------------------------------------------------------*/
SIGNAL( SIG_USART_RECV )
{
    xSmsFrameReceiveFSM(　);                                     //回调函数
}
```

5.2.2. 时钟移植文件

时钟部分的函数是最底层的时钟硬件驱动层。实现的功能包括：时钟初始化、时钟的使能、时钟的禁止和定时器中断处理。

1）时钟初始化函数 *xSmsPortTimersInit*，完成输出比较寄存器计算，配置时钟寄存器。数据帧层在初始化的时候会调用本函数，完成真正的时钟的初始化。具体功能见下面的注释。

```
/*****************************************************************
* 名　　称:xSmsPortTimersInit
* 功能描述:时钟初始化
* 输入参量:USHORT usTim1Timerout50us  多少个50us
* 输出参量:无
* 返　　回:TRUE
* 调用子程:无
```

```
* 使用方法:超时用，当接收数据帧的时候，在大于这个定时值的时候还没有接收到
                下一个字符，认为数据帧接收完成。
----------------------------------------------------------------------*/
BOOL
xSmsPortTimersInit( USHORT usTim1Timerout50us )
{
    //计算输出比较寄存器的数值
     /* Calculate overflow counter an OCR values for Timer1. */
     usTimerOCRADelta =
          ( Sms_TIMER_TICKS * usTim1Timerout50us ) / ( Sms_50US_TICKS );

     TCCR1A = 0x00;
     TCCR1B = 0x00;
     TCCR1C = 0x00;

     vSmsPortTimersDisable(   );

     return TRUE;
}
```

2）时钟使能函数 *xSmsPortTimersInit*，完成时钟使能。具体功能见下面的注释。

```
/*********************************************************************
* 名      称:vSmsPortTimersEnable
* 功能描述:时钟使能，使能中断，设置输出比较寄存器，并且开启时钟。
* 输入参量:无
* 输出参量:无
* 返      回:无
* 调用子程:无
* 使用方法:
----------------------------------------------------------------------*/
inline void
vSmsPortTimersEnable(   )
{
     TCNT1 = 0x0000;
     if( usTimerOCRADelta > 0 )
     {
          TIMSK1 |= _BV( OCIE1A );                    //使能中断
          OCR1A = usTimerOCRADelta;                    //设置输出比较寄存器
     }

     TCCR1B |= _BV( CS12 ) | _BV( CS10 );             //启动时钟
}
```

3）时钟禁止函数 *vSmsPortTimersDisable*，完成时钟使能。具体功能见下面的注释。

```
/*************************************************************
* 名      称:vSmsPortTimersDisable
* 功能描述:时钟禁止，禁止中断，停止时钟运行，清除中断标志
* 输入参量:
* 输出参量:无
* 返      回:无
* 调用子程:无
* 使用方法:
-------------------------------------------------------------*/
inline void
vSmsPortTimersDisable(   )
{
    /* Disable the timer. */
    TCCR1B &= ~( _BV( CS12 ) | _BV( CS10 ) );
    /* Disable the output compare interrupts for channel A/B. */
    TIMSK1 &= ~( _BV( OCIE1A ) );
    /* Clear output compare flags for channel A/B. */
    TIFR1 |= _BV( OCF1A ) ;
}
```

4）时钟中断处理函数 *SIGNAL(SIG_OUTPUT_COMPARE1A)*，时钟中断发生的时候，系统会自动调用本服务函数。调用的回调函数在数据帧层实现，发生中断后，根据数据帧层状态机所处的状态来判断是否接收到数据、是否是等待 AT 命令超时，然后系统会将这个时间标志发送给协议栈。具体功能见下面的注释。

```
/*************************************************************
* 名      称:SIGNAL( SIG_OUTPUT_COMPARE1A )
* 功能描述:时钟中断服务程序。发生中断后，根据数据帧层状态机所处的状态来判断是否
            接收到数据、是否是等待 AT 命令超时，然后系统会将这个时间标志发送给协议栈。
* 输入参量:无
* 输出参量:无
* 返      回:无
* 调用子程:调用数据帧层的回调函数 xSmsFrameTimerT1SExpired(   )
* 使用方法:定时器中断发生后，系统会自动调用本函数。
-------------------------------------------------------------*/
SIGNAL( SIG_OUTPUT_COMPARE1A )
{
    ( void )xSmsFrameTimerT1SExpired(   );                    //回调函数
}
```

5.3. 试验效果

如果移植完成，你再看看用户文件 DemoMega32.c。将发送短信的目标号码修改为你自己的号码，具体地，将下面语句中的第一个参数修改为你的手机号码：

eSmsSend("8613124773290", "Hello!", 6, GSM_7BIT);

编译后，你将目标代码烧写到微控制器后，你将看到如下的试验效果：

1）你会收到你的模块所发送的短信，内容为“Hello!”；

2）你发送短消息到你模块所使用的号码，你的模块可以通过串口将收到的短消息的号码、时间、编码格式以及解码后的数据打印出来。

3）你拨打你模块所使用的号码，模块会自动挂断，并且将给你的手机发送一个“Hello!”短信。

6. 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。**（我将在后续文档中完成这部分内容的编写，下面是这部分内容的提纲。）**

这个框架在逻辑上从底层到高层分为：

- ✓ 硬件层（微控制器串口发送和接收）
- ✓ 数值帧层（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

参考资料

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