

AN1409A ATK-NEO-6M GPS 模块使用

本应用文档（AN1409A，对应**探索者 STM32F407 开发板扩展实验 2**）将教大家如何在 ALIENTEK 探索者 STM32F407 开发板上使用 ATK-NEO-6M GPS 模块，并实现 GPS 定位。

本文档分为如下几部分：

- 1, ATK-NEO-6M GPS 模块简介
- 2, 硬件连接
- 3, 软件实现
- 4, 验证

1、ATK-NEO-6M GPS 模块简介

ATK-NEO-6M-V23 模块，是 ALIENTEK 生产的一款高性能 GPS 模块，模块核心采用 UBLOX 公司的 NEO-6M 模组，具有 50 个通道，追踪灵敏度高达-161dBm，测量输出频率最高可达 5Hz。ATK-NEO-6M-V23 模块具有以下特点：

- 1, 模块采用 U-BLOX NEO-6M 模组，体积小巧，性能优异。
- 2, 模块自带陶瓷天线及 MAXIM 公司 20.5dB 高增益 LNA 芯片，搜星能力强。
- 3, 模块可通过串口进行各种参数设置，并可保存在 EEPROM，使用方便。
- 4, 模块自带 IPX 接口，可以连接各种有源天线，适应能力强。
- 5, 模块兼容 3.3V/5V 电平，方便连接各种单片机系统。
- 6, 模块自带可充电后备电池，可以掉电保持星历数据¹。

注 1：在主电源断开后，后备电池可以维持半小时左右的 GPS 星历数据的保存，以支持温启动或热启动，从而实现快速定位。

ATK-NEO-6M 模块非常小巧（25.5mm*31mm），模块通过 5 个 2.54mm 间距的排针与外部连接，模块外观如图 1.1 所示：



图 1.1 ATK-NEO-6M 模块外观图

图 1.1 中，从右到左，依次为模块引出的 PIN1~PIN5 脚，各引脚的详细描述如表 1.1 所

示:

| 序号 | 名称 | 说明 |
|----|-----|--|
| 1 | PPS | 时钟脉冲输出脚 |
| 2 | RXD | 模块串口接收脚（TTL 电平，不能直接接 RS232 电平!），可接单片机的 TXD |
| 3 | TXD | 模块串口发送脚（TTL 电平，不能直接接 RS232 电平!），可接单片机的 RXD |
| 4 | GND | 地 |
| 5 | VCC | 电源（3.3V~5.0V） |

表 1.1 ATK-NEO-6M 模块各引脚功能描述

其中，PPS 引脚同时连接到了模块自带的状态指示灯：PPS，该引脚连接在 UBLOX NEO-6M 模组的 TIMEPULSE 端口，该端口的输出特性可以通过程序设置。PPS 指示灯（即 PPS 引脚），在默认条件下（没经过程序设置），有 2 个状态：

- 1，常亮，表示模块已开始工作，但还未实现定位。
- 2，闪烁（100ms 灭，900ms 亮），表示模块已经定位成功。

这样，通过 PPS 指示灯，我们就可以很方便的判断模块的当前状态，方便大家使用。

另外，图 1.1 中，左上角的 IPX 接口，可以用来外接一个有源天线，从而进一步提高模块的接收能力，通过外接有源天线，我们就可以把模块放到室内，天线放到室外，实现室内定位。

ATK-NEO-6M 模块默认采用 NMEA-0183 协议输出 GPS 定位数据，并可以通过 UBX 协议对模块进行配置，NMEA-0183 协议详细介绍请参考《ATK-NEO-6M 用户手册.pdf》，UBX 配置协议，请参考《u-blox6_ReceiverDescriptionProtocolSpec_GPS.G6-SW-10018-C.pdf》。

通过 ATK-NEO-6M GPS 模块，任何单片机（3.3V/5V 电源）都可以很方便的实现 GPS 定位，当然他也可以连接电脑，利用电脑软件实现定位。ATK-NEO-6M-V23 GPS 模块的原理图如图 1.2 所示：

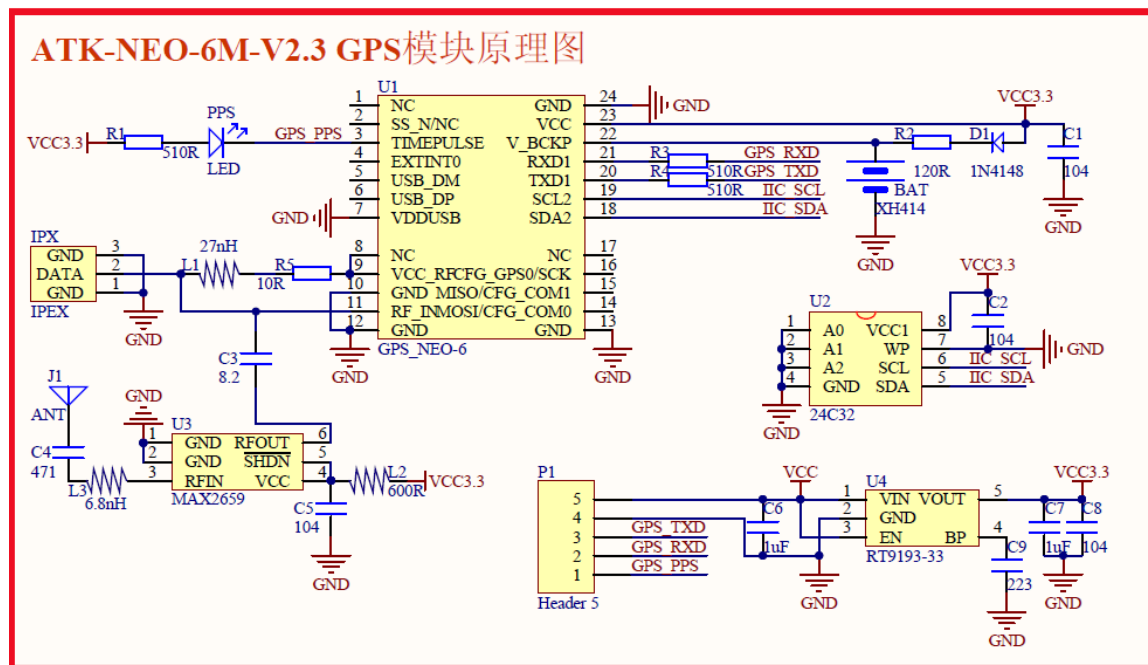


图 1.2 ATK-NEO-6M GPS 模块原理图

2、硬件连接

本实验功能简介：通过串口 3 连接 ATK-NEO-6M GPS 模块，然后通过液晶显示 GPS 信息，包括精度、纬度、高度、速度、用于定位的卫星数、可见卫星数、UTC 时间等信息。同时，

可以通过 USART 工具，设置 GPS 模块的刷新速率（最大支持 5Hz 刷新）和时钟脉冲的配置。另外，通过 KEY0 按键，可以开启或关闭 NMEA 数据的上传（即输出到串口 1，方便开发调试）。

所要用到的硬件资源如下：

- 1，指示灯 DS0
- 2，KEY0 按键
- 3，串口 1、串口 3
- 4，TFTLCD 模块
- 5，ATK-NEO-6M GPS 模块

接下来，我们看看 ATK-NEO-6M GPS 模块同 ALIENTEK 探索者 STM32F407 开发板的连接，前面我们介绍了 ATK-NEO-6M 模块的接口，探索者 STM32F407 开发板有 ATK MODULE 接口，ATK-NEO-6M GPS 模块可以直接插到该接口实现与开发板的连接。

ATK MODULE 接口同开发板主芯片的连接原理图如图 2.1 所示：

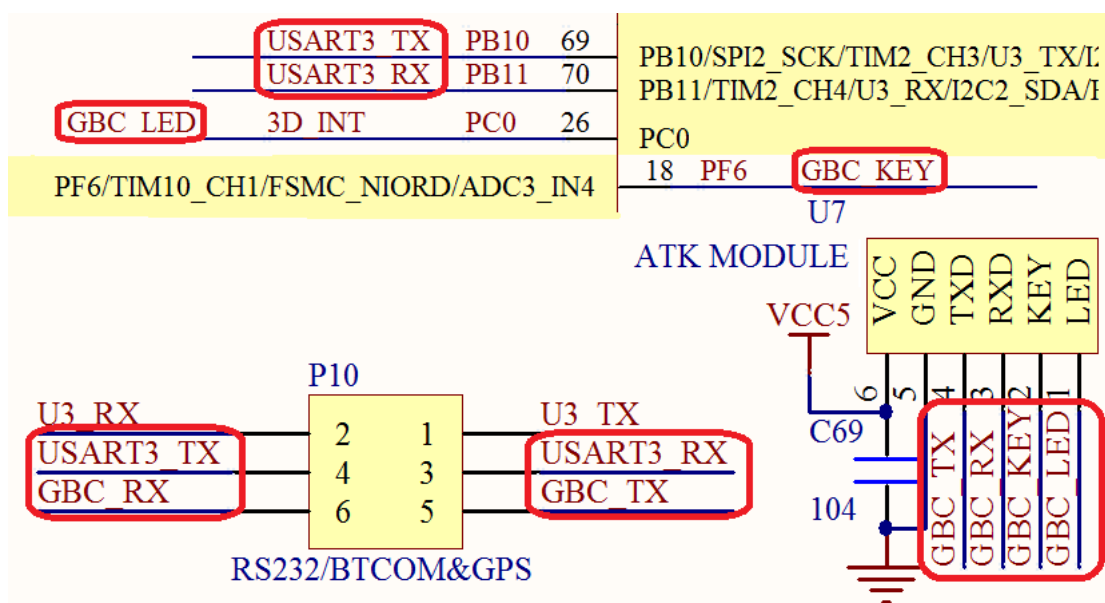


图 2.1 ATK-MODULE 接口与 MCU 连接关系

GPS 的串口将连接在探索者 STM32F407 开发板的串口 3 上面（需要用跳线帽短接 P10 的 USART3_RX 和 GBC_TX 以及 USART3_TX 和 GBC_RX），且 GPS 的 PPS 信号接在 GBC_KEY 信号上面。探索者 STM32F407 开发板与 ATK-NEO-6M GPS 模块的连接关系如表 2.1 所示：

| ATK-NEO-6M GPS 模块与开发板连接关系 | | | | | |
|---------------------------|-----|-----|------|------|-----|
| ATK-NEO-6M GPS 模块 | VCC | GND | TXD | RXD | PPS |
| 探索者 STM32F407 开发板 | 5V | GND | PB11 | PB10 | PF6 |

表 2.1 ATK-NEO-6M 模块同探索者 STM32F407 开发板连接关系表

使用时，我们只需要将 ATK-NEO-6M GPS 模块插入到开发板的 ATK-MODULE 接口（靠下插）即可，如图 2.2 所示：

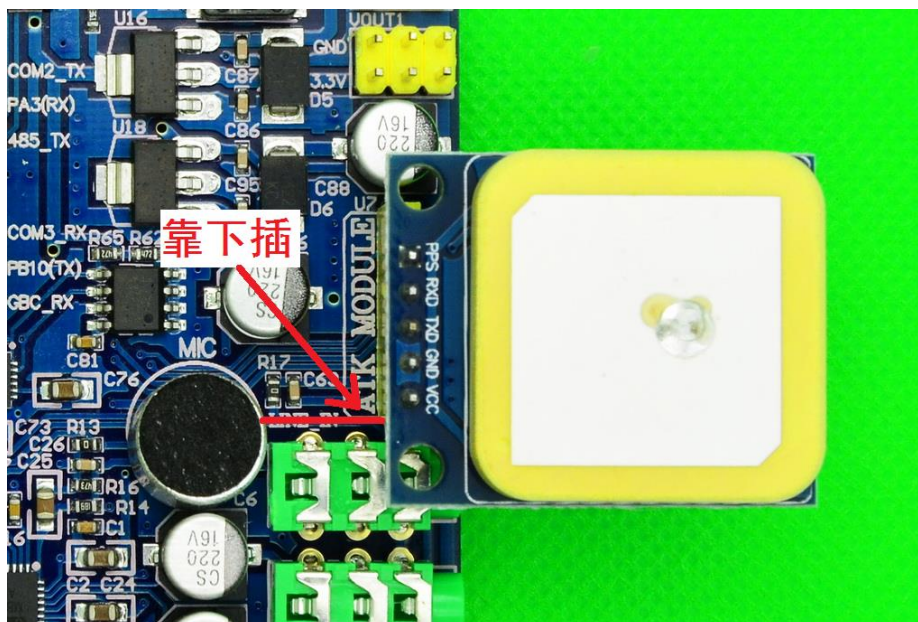


图 2.2 ATK-NEO-6M GPS 模块与探索者开发板对接实物图

注意，我们虽然将 PPS 连接到了开发板的 PF6，但是本例程并没有用到 PPS 信号，所以没做处理。注意，连接好之后，记得检查开发板 P10 的跳线帽哦!! **必须短接：USART3_RX 和 GBC_TX 以及 USART3_TX 和 GBC_RX。**

可以看出，模块与开发板的连接是很简单，不过这里特别提醒大家：

- 1, **请把 GPS 模块放到窗户边/阳台，否则可能收不到 GPS 信号。**
- 2, **如果想在室内开发，可以自备有源天线，将天线放外面，模块放室内，亦可实现定位。也可以考虑使用 ALIENTEK 提供的蓝牙串口模块（ATK-HC05）一对，这样，我们可以将 GPS 放到户外/窗口，而仍然在室内进行程序的调试开发。**

3、软件实现

本实验，我们在探索者 STM32F407 开发板扩展实验 1: **ATK-HC05 蓝牙串口模块实验** 的基础上修改，本例程用不到蓝牙模块，所以先删掉 hc05.c。

然后，在 HARDWARE 文件夹里面新建一个 GPS 文件夹，并新建 gps.c, gps.h 两个文件。然后在工程 HARDWARE 组里面添加 gps.c，并在工程添加 gps.h 的头文件包含路径。

在 gps.c 里面，我们输入如下代码：

```
#include "gps.h"
#include "led.h"
#include "delay.h"
#include "usart3.h"
#include "stdio.h"
#include "stdarg.h"
#include "string.h"
#include "math.h"
//从 buf 里面得到第 cx 个逗号所在的位置
//返回值:0~0XFE,代表逗号所在位置的偏移.
//      0XFF,代表不存在第 cx 个逗号
u8 NMEA_Comma_Pos(u8 *buf,u8 cx)
```

```
{
    u8 *p=buf;
    while(cx)
    {
        if(*buf=='*' || *buf<' ' || *buf>'z')return 0XFF;
        //遇到'*'或者非法字符,则不存在第 cx 个逗号
        if(*buf==',')cx--;
        buf++;
    }
    return buf-p;
}
```

//m^n 函数

//返回值:m^n 次方.

u32 NMEA_Pow(u8 m,u8 n)

```
{
    u32 result=1;
    while(n-->0)result*=m;
    return result;
}
```

//str 转换为数字,以','或者'*'结束

//buf:数字存储区

//dx:小数点位数,返回给调用函数

//返回值:转换后的数值

int NMEA_Str2num(u8 *buf,u8*dx)

```
{
    u8 *p=buf;
    u32 ires=0,fres=0;
    u8 ilen=0,flen=0,i;
    u8 mask=0;
    int res;
    while(1) //得到整数和小数的长度
    {
        if(*p=='-'){mask|=0X02;p++;} //是负数
        if(*p==' ' || (*p=='*'))break; //遇到结束了
        if(*p=='.'){mask|=0X01;p++;} //遇到小数点了
        else if(*p>'9' || (*p<'0')) //有非法字符
        {
            ilen=0;
            flen=0;
            break;
        }
        if(mask&0X01)flen++;
        else ilen++;
        p++;
    }
```



```
    }
    if(mask&0X02)buf++; //去掉负号
    for(i=0;i<ilen;i++) //得到整数部分数据
    {
        ires+=NMEA_Pow(10,ilen-1-i)*(buf[i]-'0');
    }
    if(flen>5)flen=5; //最多取 5 位小数
    *dx=flen; //小数点位数
    for(i=0;i<flen;i++) //得到小数部分数据
    {
        fres+=NMEA_Pow(10,flen-1-i)*(buf[ilen+1+i]-'0');
    }
    res=ires*NMEA_Pow(10,flen)+fres;
    if(mask&0X02)res=-res;
    return res;
}

//分析 GPGSV 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS 数据缓冲区首地址
void NMEA_GPGSV_Analysis(nmea_msg *gpsx,u8 *buf)
{
    u8 *p,*p1,dx;
    u8 len,i,j,slx=0;
    u8 posx;
    p=buf;
    p1=(u8*)strstr((const char *)p,"$GPGSV");
    len=p1[7]-'0'; //得到 GPGSV 的条数
    posx=NMEA_Comma_Pos(p1,3); //得到可见卫星总数
    if(posx!=0XFF)gpsx->svnum=NMEA_Str2num(p1+posx,&dx);
    for(i=0;i<len;i++)
    {
        p1=(u8*)strstr((const char *)p,"$GPGSV");
        for(j=0;j<4;j++)
        {
            posx=NMEA_Comma_Pos(p1,4+j*4);
            if(posx!=0XFF)gpsx->slmsg[slx].num=NMEA_Str2num(p1+posx,&dx);
            //得到卫星编号
            else break;
            posx=NMEA_Comma_Pos(p1,5+j*4);
            if(posx!=0XFF)gpsx->slmsg[slx].eledeg=NMEA_Str2num(p1+posx,&dx);
            //得到卫星仰角
            else break;
            posx=NMEA_Comma_Pos(p1,6+j*4);
            if(posx!=0XFF)gpsx->slmsg[slx].azideg=NMEA_Str2num(p1+posx,&dx);
```

```
        //得到卫星方位角
        else break;
        posx=NMEA_Comma_Pos(p1,7+j*4);
        if(posx!=0XFF)gpsx->slmsg[slx].sn=NMEA_Str2num(p1+posx,&dx);
        //得到卫星信噪比
        else break;
        slx++;
    }
    p=p1+1;//切换到下一个 GPGSV 信息
}
}
//分析 GPGGA 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS 数据缓冲区首地址
void NMEA_GPGGA_Analysis(nmea_msg *gpsx,u8 *buf)
{
    u8 *p1,dx;
    u8 posx;
    p1=(u8*)strstr((const char *)buf,"$GPGGA");
    posx=NMEA_Comma_Pos(p1,6);           //得到 GPS 状态
    if(posx!=0XFF)gpsx->gpssta=NMEA_Str2num(p1+posx,&dx);
    posx=NMEA_Comma_Pos(p1,7);           //得到用于定位的卫星数
    if(posx!=0XFF)gpsx->posslnum=NMEA_Str2num(p1+posx,&dx);
    posx=NMEA_Comma_Pos(p1,9);           //得到海拔高度
    if(posx!=0XFF)gpsx->altitude=NMEA_Str2num(p1+posx,&dx);
}
//分析 GPGSA 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS 数据缓冲区首地址
void NMEA_GPGSA_Analysis(nmea_msg *gpsx,u8 *buf)
{
    u8 *p1,dx;
    u8 posx;
    u8 i;
    p1=(u8*)strstr((const char *)buf,"$GPGSA");
    posx=NMEA_Comma_Pos(p1,2);           //得到定位类型
    if(posx!=0XFF)gpsx->fixmode=NMEA_Str2num(p1+posx,&dx);
    for(i=0;i<12;i++)                   //得到定位卫星编号
    {
        posx=NMEA_Comma_Pos(p1,3+i);
        if(posx!=0XFF)gpsx->possl[i]=NMEA_Str2num(p1+posx,&dx);
        else break;
    }
    posx=NMEA_Comma_Pos(p1,15);          //得到 PDOP 位置精度因子
```

```

    if(posx!=0XFF)gpsx->pdop=NMEA_Str2num(p1+posx,&dx);
    posx=NMEA_Comma_Pos(p1,16);           //得到 HDOP 位置精度因子
    if(posx!=0XFF)gpsx->hdop=NMEA_Str2num(p1+posx,&dx);
    posx=NMEA_Comma_Pos(p1,17);           //得到 VDOP 位置精度因子
    if(posx!=0XFF)gpsx->vdop=NMEA_Str2num(p1+posx,&dx);
}
//分析 GPRMC 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS 数据缓冲区首地址
void NMEA_GPRMC_Analysis(nmea_msg *gpsx,u8 *buf)
{
    u8 *p1,dx;
    u8 posx;
    u32 temp;
    float rs;
    p1=(u8*)strstr((const char *)buf,"GPRMC");
    //"$GPRMC",经常有&和 GPRMC 分开的情况,故只判断 GPRMC.
    posx=NMEA_Comma_Pos(p1,1);           //得到 UTC 时间
    if(posx!=0XFF)
    {
        temp=NMEA_Str2num(p1+posx,&dx)/NMEA_Pow(10,dx);//得到 UTC 时间
        gpsx->utc.hour=temp/10000;
        gpsx->utc.min=(temp/100)%100;
        gpsx->utc.sec=temp%100;
    }
    posx=NMEA_Comma_Pos(p1,3);           //得到纬度
    if(posx!=0XFF)
    {
        temp=NMEA_Str2num(p1+posx,&dx);
        gpsx->latitude=temp/NMEA_Pow(10,dx+2);    //得到°
        rs=temp%NMEA_Pow(10,dx+2);               //得到'
        gpsx->latitude=gpsx->latitude*NMEA_Pow(10,5)+(rs*NMEA_Pow(10,5-dx))/60;
        //转换为°
    }
    posx=NMEA_Comma_Pos(p1,4);           //南纬还是北纬
    if(posx!=0XFF)gpsx->nshemi=*(p1+posx);
    posx=NMEA_Comma_Pos(p1,5);           //得到经度
    if(posx!=0XFF)
    {
        temp=NMEA_Str2num(p1+posx,&dx);
        gpsx->longitude=temp/NMEA_Pow(10,dx+2);    //得到°
        rs=temp%NMEA_Pow(10,dx+2);               //得到'
        gpsx->longitude=gpsx->longitude*NMEA_Pow(10,5)+(rs*NMEA_Pow(10,5-dx))
        /60;//转换为°
    }
}

```



```
    }
    posx=NMEA_Comma_Pos(p1,6);           //东经还是西经
    if(posx!=0XFF)gpsx->ewhemi=*(p1+posx);
    posx=NMEA_Comma_Pos(p1,9);           //得到 UTC 日期
    if(posx!=0XFF)
    {
        temp=NMEA_Str2num(p1+posx,&dx);   //得到 UTC 日期
        gpsx->utc.date=temp/10000;
        gpsx->utc.month=(temp/100)%100;
        gpsx->utc.year=2000+temp%100;
    }
}
//分析 GPVTG 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS 数据缓冲区首地址
void NMEA_GPVTG_Analysis(nmea_msg *gpsx,u8 *buf)
{
    u8 *p1,dx;
    u8 posx;
    p1=(u8*)strstr((const char *)buf,"$GPVTG");
    posx=NMEA_Comma_Pos(p1,7);           //得到地面速率
    if(posx!=0XFF)
    {
        gpsx->speed=NMEA_Str2num(p1+posx,&dx);
        if(dx<3)gpsx->speed*=NMEA_Pow(10,3-dx); //确保扩大 1000 倍
    }
}
//提取 NMEA-0183 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS 数据缓冲区首地址
void GPS_Analysis(nmea_msg *gpsx,u8 *buf)
{
    NMEA_GPGSV_Analysis(gpsx,buf); //GPGSV 解析
    NMEA_GPGGA_Analysis(gpsx,buf); //GPGGA 解析
    NMEA_GPGSA_Analysis(gpsx,buf); //GPGSA 解析
    NMEA_GPRMC_Analysis(gpsx,buf); //GPRMC 解析
    NMEA_GPVTG_Analysis(gpsx,buf); //GPVTG 解析
}

//GPS 校验和计算
//buf:数据缓存区首地址
//len:数据长度
//cka,ckb:两个校验结果.
void Ublox_CheckSum(u8 *buf,u16 len,u8* cka,u8*ckb)
```

```
{
    u16 i;
    *cka=0;*ckb=0;
    for(i=0;i<len;i++)
    {
        *cka=*cka+buf[i];
        *ckb=*ckb+*cka;
    }
}

//////////////////////////////////UBLOX 配置代码//////////////////////////////////
//检查 CFG 配置执行情况
//返回值:0,ACK 成功
//      1,接收超时错误
//      2,没有找到同步字符
//      3,接收到 NACK 应答
u8 Ublox_Cfg_Ack_Check(void)
{
    u16 len=0,i;
    u8 rval=0;
    while((USART3_RX_STA&0X8000)==0 && len<100)//等待接收到应答
    {
        len++;
        delay_ms(5);
    }
    if(len<250)        //超时错误.
    {
        len=USART3_RX_STA&0X7FFF;    //此次接收到的数据长度
        for(i=0;i<len;i++)if(USART3_RX_BUF[i]==0XB5)break;//查找同步字符 0XB5
        if(i==len)rval=2;                //没有找到同步字符
        else if(USART3_RX_BUF[i+3]==0X00)rval=3;//接收到 NACK 应答
        else rval=0;                    //接收到 ACK 应答
    }else rval=1;                //接收超时错误
    USART3_RX_STA=0;            //清除接收
    return rval;
}
//配置保存
//将当前配置保存在外部 EEPROM 里面
//返回值:0,执行成功;1,执行失败.
u8 Ublox_Cfg_Save(void)
{
    u8 i;
    _ublox_cfg_cfg *cfg_cfg=(_ublox_cfg_cfg *)USART3_TX_BUF;
    cfg_cfg->header=0X62B5;    //cfg header
    cfg_cfg->id=0X0906;        //cfg id
```

```
    cfg_cfg->dlength=13;        //数据区长度为 13 个字节.
    cfg_cfg->clearmask=0;        //清除掩码为 0
    cfg_cfg->savemask=0XFFFF;    //保存掩码为 0XFFFF
    cfg_cfg->loadmask=0;         //加载掩码为 0
    cfg_cfg->devicemask=4;        //保存在 EEPROM 里面
    Ublox_CheckSum((u8*)&cfg_cfg->id,sizeof(_ublox_cfg_cfg)-4,&cfg_cfg->cka,
                  &cfg_cfg->ckb);
    Ublox_Send_Date((u8*)cfg_cfg,sizeof(_ublox_cfg_cfg));//发送数据给 NEO-6M
    for(i=0;i<6;i++)if(Ublox_Cfg_Ack_Check()==0)break;
    //EEPROM 写入需要比较久时间,所以连续判断多次
    return i==6?1:0;
}
//配置 NMEA 输出信息格式
//msgid:要操作的 NMEA 消息条目,具体见下面的参数表
//      00,GPGGA;01,GPGLL;02,GPGSA;
//      03,GPGSV;04,GPRMC;05,GPVTG;
//      06,GPGRS;07,GPGST;08,GPZDA;
//      09,GPGBS;0A,GPDTM;0D,GPGNS;
//uart1set:0,输出关闭;1,输出开启.
//返回值:0,执行成功;其他,执行失败.
u8 Ublox_Cfg_Msg(u8 msgid,u8 uart1set)
{
    _ublox_cfg_msg *cfg_msg=(_ublox_cfg_msg *)USART3_TX_BUF;
    cfg_msg->header=0X62B5;      //cfg header
    cfg_msg->id=0X0106;          //cfg msg id
    cfg_msg->dlength=8;          //数据区长度为 8 个字节.
    cfg_msg->msgclass=0XF0;      //NMEA 消息
    cfg_msg->msgid=msgid;        //要操作的 NMEA 消息条目
    cfg_msg->iicset=1;           //默认开启
    cfg_msg->uart1set=uart1set;  //开关设置
    cfg_msg->uart2set=1;         //默认开启
    cfg_msg->usbset=1;           //默认开启
    cfg_msg->spiset=1;           //默认开启
    cfg_msg->ncset=1;            //默认开启
    Ublox_CheckSum((u8*)&cfg_msg->id,sizeof(_ublox_cfg_msg)-4,&cfg_msg->cka,
                  &cfg_msg->ckb);
    Ublox_Send_Date((u8*)cfg_msg,sizeof(_ublox_cfg_msg));//发送数据给 NEO-6M
    return Ublox_Cfg_Ack_Check();
}
//配置 NMEA 输出信息格式
//baudrate:波特率,4800/9600/19200/38400/57600/115200/230400
//返回值:0,执行成功;其他,执行失败(这里不会返回 0 了)
u8 Ublox_Cfg_Prt(u32 baudrate)
{

```

```
_ublox_cfg_prt *cfg_prt=(_ublox_cfg_prt *)USART3_TX_BUF;
cfg_prt->header=0X62B5;    //cfg header
cfg_prt->id=0X0006;        //cfg prt id
cfg_prt->dlength=20;        //数据区长度为 20 个字节.
cfg_prt->portid=1;          //操作串口 1
cfg_prt->reserved=0;        //保留字节,设置为 0
cfg_prt->txready=0;         //TX Ready 设置为 0
cfg_prt->mode=0X08D0;       //8 位,1 个停止位,无校验位
cfg_prt->baudrate=baudrate; //波特率设置
cfg_prt->inprotomask=0X0007;//0+1+2
cfg_prt->outprotomask=0X0007;//0+1+2
cfg_prt->reserved4=0;        //保留字节,设置为 0
cfg_prt->reserved5=0;        //保留字节,设置为 0
Ublox_CheckSum((u8*)&cfg_prt->id,sizeof(_ublox_cfg_prt)-4,&cfg_prt->cka,
                &cfg_prt->ckb);
Ublox_Send_Date((u8*)cfg_prt,sizeof(_ublox_cfg_prt));//发送数据给 NEO-6M
delay_ms(200);              //等待发送完成
usart3_init(42,baudrate);    //重新初始化串口 3
return Ublox_Cfg_Ack_Check();
//这里不会返回 0,因为 UBLOX 发回来的应答在串口重新初始化时已经被丢弃了.
}

//配置 UBLOX NEO-6 的时钟脉冲输出
//interval:脉冲间隔(us)
//length:脉冲宽度(us)
//status:脉冲配置:1,高电平有效;0,关闭;-1,低电平有效.
//返回值:0,发送成功;其他,发送失败.
u8 Ublox_Cfg_Tp(u32 interval,u32 length,signed char status)
{
    _ublox_cfg_tp *cfg_tp=(_ublox_cfg_tp *)USART3_TX_BUF;
    cfg_tp->header=0X62B5;    //cfg header
    cfg_tp->id=0X0706;        //cfg tp id
    cfg_tp->dlength=20;        //数据区长度为 20 个字节.
    cfg_tp->interval=interval; //脉冲间隔,us
    cfg_tp->length=length;     //脉冲宽度,us
    cfg_tp->status=status;     //时钟脉冲配置
    cfg_tp->timeref=0;         //参考 UTC 时间
    cfg_tp->flags=0;           //flags 为 0
    cfg_tp->reserved=0;        //保留位为 0
    cfg_tp->antdelay=820;      //天线延时为 820ns
    cfg_tp->rfdelay=0;         //RF 延时为 0ns
    cfg_tp->userdelay=0;       //用户延时为 0ns
    Ublox_CheckSum((u8*)&cfg_tp->id,sizeof(_ublox_cfg_tp)-4,&cfg_tp->cka,
                    &cfg_tp->ckb);
    Ublox_Send_Date((u8*)cfg_tp,sizeof(_ublox_cfg_tp));//发送数据给 NEO-6M
```

```

        return Ublox_Cfg_Ack_Check();
    }
    //配置 UBLOX NEO-6 的更新速率
    //measrate:测量时间间隔, 单位为 ms, 最少不能小于 200ms (5Hz)
    //reftime:参考时间, 0=UTC Time; 1=GPS Time (一般设置为 1)
    //返回值:0,发送成功;其他,发送失败.
    u8 Ublox_Cfg_Rate(u16 measrate,u8 reftime)
    {
        _ublox_cfg_rate *cfg_rate=(_ublox_cfg_rate *)USART3_TX_BUF;
        if(measrate<200)return 1;    //小于 200ms, 直接退出
        cfg_rate->header=0X62B5;    //cfg header
        cfg_rate->id=0X0806;        //cfg rate id
        cfg_rate->dlength=6;        //数据区长度为 6 个字节.
        cfg_rate->measrate=measrate;//脉冲间隔,us
        cfg_rate->navrate=1;        //导航速率 (周期), 固定为 1
        cfg_rate->timeref=reftime;  //参考时间为 GPS 时间
        Ublox_CheckSum((u8*)(&cfg_rate->id),sizeof(_ublox_cfg_rate)-4,&cfg_rate->cka,
                        &cfg_rate->ckb);
        Ublox_Send_Date((u8*)cfg_rate,sizeof(_ublox_cfg_rate));//发送数据给 NEO-6M
        return Ublox_Cfg_Ack_Check();
    }
    //发送一批数据给 Ublox NEO-6M, 这里通过串口 3 发送
    //dbuf: 数据缓存首地址
    //len: 要发送的字节数
    void Ublox_Send_Date(u8* dbuf,u16 len)
    {
        u16 j;
        for(j=0;j<len;j++)//循环发送数据
        {
            while((USART3->SR&0X40)==0);//循环发送,直到发送完毕
            USART3->DR=dbuf[j];
        }
    }
}

```

这部分代码可以分为 2 个部分, 第一部分是 NMEA-0183 数据解析部分, 另外一部分则是 UBX 协议控制部分。

NMEA-0183 协议解析部分, 这里利用了一个简单的数逗号方法来解析。我们知道 NMEA-0183 协议都是以类似\$GPGSV 的开头, 然后固定输出格式, 不论是否有数据输出, 逗号是肯定会有, 而且都会以 ‘*’ 作为有效数据的结尾, 所以, 我们了解了 NMEA-0183 协议的数据格式 (在 ATK-NEO-6M 的用户手册有详细介绍) 之后, 就可以通过数逗号的方法, 来解析数据了。本代码实现了对 NMEA-0183 协议的\$GPGGA、\$GPGSA、\$GPGSV、\$GPRMC 和\$GPVTG 等 5 类帧的解析, 结果存放在通过 gps.h 定义的 nmea_msg 结构体内。

UBX 协议控制部分, 此部分我们只实现了 NEO-6M 模组常用的 5 个配置: 保存设置、输出信息设置、串口波特率设置、时钟脉冲设置和输出频率设置。

保存设置, 通过函数 Ublox_Cfg_Cfg_Save 实现, 调用该函数, 可以实现将当前 NEO-6M

模块的配置信息保存到 EEPROM 里面。

输出信息设置，通过函数 `Ublox_Cfg_Msg` 实现，该函数可以开启/关闭 NMEA 消息的某个条目。

串口波特率设置，通过函数 `Ublox_Cfg_Prt` 实现，该函数可以设置模块的波特率。

时钟脉冲设置，通过函数 `Ublox_Cfg_Tp` 实现，可以设置脉冲间隔，脉冲宽度等信息。

输出频率设置，通过函数 `Ublox_Cfg_Rate` 实现，该函数可以设置模块的测量输出频率，最快可以达到 5Hz 的测量输出频率。

最后 `Ublox_Send_Data` 函数，用于发送一批设置好的数据给串口 3，完成对 GPS 模块的配置。

我们将这 5 个函数都加入 USMART 控制，方便大家测试。另外要在 `usart3.h` 里面，将 `USART3_MAX_RECV_LEN` 的值设置为 800。然后在 `gps.h` 里面，我们输入如下代码：

```
#ifndef __GPS_H
#define __GPS_H
#include "sys.h"
//GPS NMEA-0183 协议重要参数结构体定义
//卫星信息
__packed typedef struct
{
    u8 num;        //卫星编号
    u8 eledeg;     //卫星仰角
    u16 azideg;    //卫星方位角
    u8 sn;         //信噪比
}nmea_slmsg;
//UTC 时间信息
__packed typedef struct
{
    u16 year; //年份
    u8 month;  //月份
    u8 date;  //日期
    u8 hour;  //小时
    u8 min;   //分钟
    u8 sec;   //秒钟
}nmea_utc_time;
//NMEA 0183 协议解析后数据存放结构体
__packed typedef struct
{
    u8 svnum;           //可见卫星数
    nmea_slmsg slmsg[12]; //最多 12 颗卫星
    nmea_utc_time utc;   //UTC 时间
    u32 latitude;        //纬度 分扩大 100000 倍,实际要除以 100000
    u8 nshemi;           //北纬/南纬,N:北纬;S:南纬
    u32 longitude;       //经度 分扩大 100000 倍,实际要除以 100000
    u8 ewhemi;           //东经/西经,E:东经;W:西经
    u8 gpssta;           //GPS 状态:0,未定位;1,非差分定位;2,差分定位;6,正在估算.
```



```

    u8 posslnum;           //用于定位的卫星数,0~12.
    u8 possl[12];          //用于定位的卫星编号
    u8 fixmode;             //定位类型:1,没有定位;2,2D 定位;3,3D 定位
    u16 pdop;               //位置精度因子 0~500,对应实际值 0~50.0
    u16 hdop;               //水平精度因子 0~500,对应实际值 0~50.0
    u16 vdop;               //垂直精度因子 0~500,对应实际值 0~50.0
    int altitude;           //海拔高度,放大了 10 倍,实际除以 10.单位:0.1m
    u16 speed;              //地面速率,放大了 1000 倍,实际除以 10.单位:0.001 公里/小时
}nmea_msg;
////////////////////////////////////
//UBLOX NEO-6M 配置(清除,保存,加载等)结构体
__packed typedef struct
{
    u16 header;             //cfg header,固定为 0X62B5(小端模式)
    u16 id;                  //CFG CFG ID:0X0906 (小端模式)
    u16 dlength;             //数据长度 12/13
    u32 clearmask;           //子区域清除掩码(1 有效)
    u32 savemask;            //子区域保存掩码
    u32 loadmask;            //子区域加载掩码
    u8 devicemask;           //目标器件选择掩码 b0:BK RAM;b1:FLASH;b2,EEPROM;b4,SPI FLASH
    u8 cka;                  //校验 CK_A
    u8 ckb;                  //校验 CK_B
}_ublox_cfg_cfg;

//UBLOX NEO-6M 消息设置结构体
__packed typedef struct
{
    u16 header;             //cfg header,固定为 0X62B5(小端模式)
    u16 id;                  //CFG MSG ID:0X0106 (小端模式)
    u16 dlength;             //数据长度 8
    u8 msgclass;             //消息类型(F0 代表 NMEA 消息格式)
    u8 msgid;                //消息 ID
                             //00,GPGGA;01,GPGLL;02,GPGSA;
                             //03,GPGSV;04,GPRMC;05,GPVTG;
                             //06,GPGRS;07,GPGST;08,GPZDA;
                             //09,GPGBS;0A,GPDTM;0D,GPGNS;
    u8 iicset;               //IIC 消输出设置 0,关闭;1,使能.
    u8 uart1set;             //UART1 输出设置 0,关闭;1,使能.
    u8 uart2set;             //UART2 输出设置 0,关闭;1,使能.
    u8 usbset;               //USB 输出设置 0,关闭;1,使能.
    u8 spiset;               //SPI 输出设置 0,关闭;1,使能.
    u8 ncset;                //未知输出设置 默认为 1 即可.
    u8 cka;                  //校验 CK_A

```

```
    u8  ckb;                //校验 CK_B
}_ublox_cfg_msg;

//UBLOX NEO-6M UART 端口设置结构体
__packed typedef struct
{
    u16 header;              //cfg header,固定为 0X62B5(小端模式)
    u16 id;                  //CFG PRT ID:0X0006 (小端模式)
    u16 dlength;             //数据长度 20
    u8  portid;              //端口号,0=IIC;1=UART1;2=UART2;3=USB;4=SPI;
    u8  reserved;            //保留,设置为 0
    u16 txready;             //TX Ready 引脚设置,默认为 0
    u32 mode;                //串口工作模式设置,奇偶校验,停止位,字节长度等的设置.
    u32 baudrate;            //波特率设置
    u16 inprotomask;         //输入协议激活屏蔽位 默认设置为 0X07 0X00 即可.
    u16 outprotomask;       //输出协议激活屏蔽位 默认设置为 0X07 0X00 即可.
    u16 reserved4;          //保留,设置为 0
    u16 reserved5;          //保留,设置为 0
    u8  cka;                //校验 CK_A
    u8  ckb;                //校验 CK_B
}_ublox_cfg_prt;

//UBLOX NEO-6M 时钟脉冲配置结构体
__packed typedef struct
{
    u16 header;              //cfg header,固定为 0X62B5(小端模式)
    u16 id;                  //CFG TP ID:0X0706 (小端模式)
    u16 dlength;             //数据长度
    u32 interval;            //时钟脉冲间隔,单位为 us
    u32 length;              //脉冲宽度,单位为 us
    signed char status;      //时钟脉冲配置:1,高电平有效;0,关闭;-1,低电平有效.
    u8 timeref;              //参考时间:0,UTC 时间;1,GPS 时间;2,当地时间.
    u8 flags;                //时间脉冲设置标志
    u8 reserved;             //保留
    signed short antdelay;    //天线延时
    signed short rfdelay;     //RF 延时
    signed int userdelay;     //用户延时
    u8 cka;                  //校验 CK_A
    u8 ckb;                  //校验 CK_B
}_ublox_cfg_tp;

//UBLOX NEO-6M 刷新速率配置结构体
__packed typedef struct
{
```

```

    u16 header;           //cfg header,固定为 0X62B5(小端模式)
    u16 id;               //CFG RATE ID:0X0806 (小端模式)
    u16 dlength;          //数据长度
    u16 measrate;         //测量时间间隔,单位为 ms,最少不能小于 200ms(5Hz)
    u16 navrate;          //导航速率(周期),固定为 1
    u16 timeref;          //参考时间: 0=UTC Time; 1=GPS Time;
    u8  cka;              //校验 CK_A
    u8  ckb;              //校验 CK_B
} _ublox_cfg_rate;

int NMEA_Str2num(u8 *buf,u8*dx);
void GPS_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GPGSV_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GPGGA_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GPGSA_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GPRMC_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GPVTG_Analysis(nmea_msg *gpsx,u8 *buf);
u8 Ublox_Cfg_Cfg_Save(void);
u8 Ublox_Cfg_Msg(u8 msgid,u8 uart1set);
u8 Ublox_Cfg_Prt(u32 baudrate);
u8 Ublox_Cfg_Tp(u32 interval,u32 length,signed char status);
u8 Ublox_Cfg_Rate(u16 measrate,u8 reftime);
void Ublox_Send_Data(u8* dbuf,u16 len);
#endif

```

gps.h 里面的内容, 都有非常详细的备注, 这里就不多说了。

最后, 在 test.c 里面, 修改代码如下:

```

u8 USART1_TX_BUF[USART3_MAX_RECV_LEN]; //串口 1,发送缓存区
nmea_msg gpsx;                          //GPS 信息
__align(4) u8 dtbuf[50];                //打印缓存器
const u8*fixmode_tbl[4]={ "Fail","Fail"," 2D "," 3D "}; //fix mode 字符串
//显示 GPS 定位信息
void Gps_Msg_Show(void)
{
    float tp;
    POINT_COLOR=BLUE;
    tp=gpsx.longitude;
    sprintf((char *)dtbuf,"Longitude:%.5f %1c    ",tp/=100000,gpsx.ewhemi);//得到经度
    LCD_ShowString(30,130,200,16,16,dtbuf);
    tp=gpsx.latitude;
    sprintf((char *)dtbuf,"Latitude:%.5f %1c    ",tp/=100000,gpsx.nshemi); //得到纬度
    LCD_ShowString(30,150,200,16,16,dtbuf);
    tp=gpsx.altitude;
    sprintf((char *)dtbuf,"Altitude:%.1fm      ",tp/=10);//得到高度字符串
}

```

```
LCD_ShowString(30,170,200,16,16,dtbuf);
tp=gpsx.speed;
sprintf((char *)dtbuf,"Speed:%.3fkm/h      ",tp/=1000);//得到速度字符串
LCD_ShowString(30,190,200,16,16,dtbuf);
if(gpsx.fixmode<=3)//定位状态
{
    sprintf((char *)dtbuf,"Fix Mode:%s",fixmode_tbl[gpsx.fixmode]);
    LCD_ShowString(30,210,200,16,16,dtbuf);
}
sprintf((char *)dtbuf,"Valid satellite:%02d",gpsx.posslnum);//用于定位的卫星数
LCD_ShowString(30,230,200,16,16,dtbuf);
sprintf((char *)dtbuf,"Visible satellite:%02d",gpsx.svnum%100);//可见卫星数
LCD_ShowString(30,250,200,16,16,dtbuf);
sprintf((char *)dtbuf,"UTC Date:%04d/%02d/%02d      ",gpsx.utc.year,gpsx.utc.month,
    gpsx.utc.date); //显示 UTC 日期
//printf("year2:%d\r\n",gpsx.utc.year);
LCD_ShowString(30,270,200,16,16,dtbuf);
sprintf((char *)dtbuf,"UTC Time:%02d:%02d:%02d      ",gpsx.utc.hour,gpsx.utc.min,
    gpsx.utc.sec); //显示 UTC 时间
LCD_ShowString(30,290,200,16,16,dtbuf);
}
int main(void)
{
    u16 i,rxlen;
    u16 lenx;
    u8 key=0XFF;
    u8 upload=0;
    Stm32_Clock_Init(336,8,2,7);//设置时钟,168Mhz
    delay_init(168);           //延时初始化
    uart_init(84,115200);      //初始化串口波特率为 115200
    usart3_init(42,38400);     //初始化串口 3 波特率为 38400
    usmart_dev.init(84);       //初始化 USMART
    LED_Init();                //初始化 LED
    KEY_Init();                //初始化按键
    LCD_Init();                //初始化 LCD
    usmart_dev.init(72);       //初始化 USMART
    POINT_COLOR=RED;
    LCD_ShowString(30,20,200,16,16,"ALIENTEK STM32F4 ^_^");
    LCD_ShowString(30,40,200,16,16,"NEO-6M GPS TEST");
    LCD_ShowString(30,60,200,16,16,"ATOM@ALIENTEK");
    LCD_ShowString(30,80,200,16,16,"KEY0:Upload NMEA Data SW");
    LCD_ShowString(30,100,200,16,16,"NMEA Data Upload:OFF");
    if(Ublox_Cfg_Rate(1000,1)!=0)
        //设置定位信息更新速度为 1000ms,顺便判断 GPS 模块是否在位.
```

```

{
    LCD_ShowString(30,120,200,16,16,"NEO-6M Setting...");
    while((Ublox_Cfg_Rate(1000,1)!=0)&&key)
        //持续判断,直到可以检查到 NEO-6M,且数据保存成功
    {
        usart3_init(42,9600);
        //初始化串口 3, 9600(EEPROM 没有保存数据的时候,波特率为 9600.)
        Ublox_Cfg_Prt(38400);           //重新设置模块的波特率为 38400
        usart3_init(42,38400);          //重新设置 38400
        Ublox_Cfg_Tp(1000000,100000,1); //设置 PPS 为 1 秒钟输出,脉宽为 100ms
        key=Ublox_Cfg_Save();           //保存配置
    }
    LCD_ShowString(30,120,200,16,16,"NEO-6M Set Done!!");
    delay_ms(500);
    LCD_Fill(30,120,30+200,120+16,WHITE); //清除显示
}
while(1)
{
    delay_ms(1);
    if(USART3_RX_STA&0X8000)           //接收到一次数据了
    {
        rxlen=USART3_RX_STA&0X7FFF;    //得到数据长度
        for(i=0;i<rxlen;i++)USART1_TX_BUF[i]=USART3_RX_BUF[i];
        USART3_RX_STA=0;                //启动下一次接收
        USART1_TX_BUF[i]=0;             //自动添加结束符
        GPS_Analysis(&gpsx,(u8*)USART1_TX_BUF); //分析字符串
        Gps_Msg_Show();                 //显示信息
        if(upload)printf("\r\n%s\r\n",USART1_TX_BUF); //发送数据到串口 1
    }
    key=KEY_Scan(0);
    if(key==KEY0_PRES)
    {
        upload=!upload;
        POINT_COLOR=RED;
        if(upload)LCD_ShowString(30,100,200,16,16,"NMEA Data Upload:ON ");
        else LCD_ShowString(30,100,200,16,16,"NMEA Data Upload:OFF");
    }
    if((lenx%500)==0)LED0=!LED0;
    lenx++;
}
}

```

此部分代码比较简单, main 函数初始化硬件之后, 通过 Ublox_Cfg_Rate 函数判断模块是否在位, 如果不在位, 则尝试去设置模块的波特率为 38400, 直到检测到模块在位为止。

然后, 进入死循环, 等待串口 2 接收 GPS 数据。每次接收到 GPS 模块发送过来的数据,

就执行数据解析，数据解析后执行 GPS 定位数据的显示，并可以根据需要（通过 KEY0 按键开启/关闭），将收到的数据通过串口 1 发送给上位机。

至此，整个 ATK-NEO-6M GPS 模块测试代码就介绍完了，我们接下来看代码验证。

4、验证

在代码编译成功之后，下载代码到我们的 STM32 开发板上（假设 ATK-NEO-6M GPS 模块已经正确连接到开发板，并且再次提醒大家，必须把 ATK-NEO-6M GPS 模块置于窗户旁、阳台或者室外，否则可能搜不到卫星!!!），如果模块和开发板的连接不正确（比如 TXD，RXD 接反了），或者模块的波特率设置有问题（不是 9600 或 38400），则液晶模块会一直显示：

NEO-6M Setting...

如果出现这种情况，请检查问题原因（参见光盘：增值资料→ALIENTEK 产品资料→ATK-NEO-6M GPS 模块→ATK-NEO-6M GPS 模块问题汇总.pdf）。排除问题根源后，就会进入到下一步，正常到模块以后，开发板的 LCD 显示如图 4.1 所示界面：

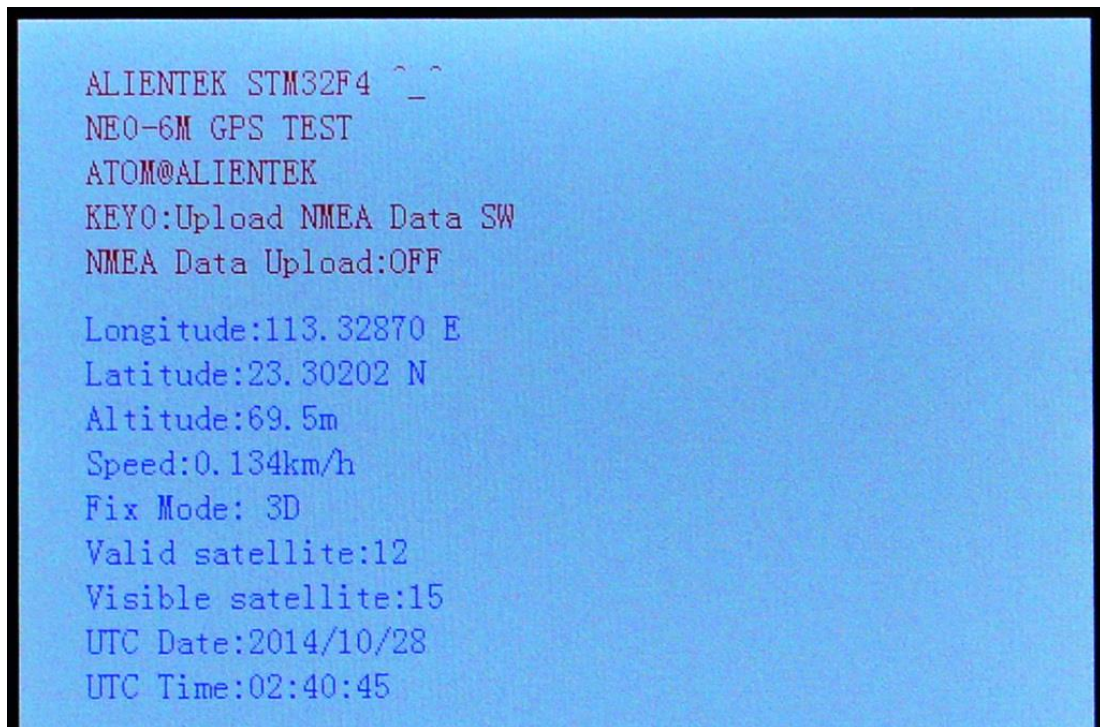


图 4.1 LCD 显示界面

上图是我们的 GPS 模块成功定位后的照片，可以得到当前地点的经纬度、高度、速度、定位模式、用于定位卫星数、可见卫星数和 UTC 日期时间等信息。此时，我们的 ATK-NEO-6M GPS 模块，用于定位的卫星达到 12 颗，可见的卫星数多达 15 颗！

我们打开 u-center 软件，连接开发板，并按一下开发板的 KEY0，程序上传 NMEA 数据到电脑，可以看到 u-center 软件显示如图 4.2 所示（这里提醒大家，u-center 会控制 DTR/RTS，将 B0 拉高，导致 u-center 连接开发板以后，按开发板的复位开发板不会运行代码。所以必须先断开 u-center 的连接，再按开发板的复位，待开发板程序启动以后，再连接!）：

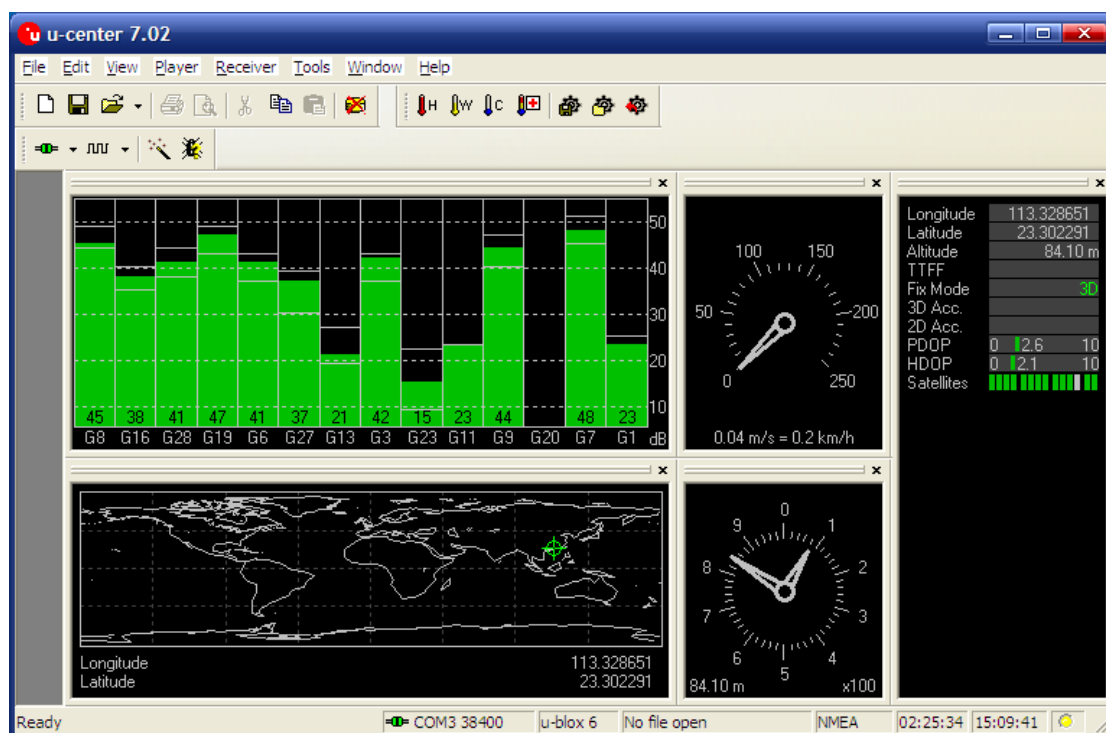


图 4.2 u-center 显示 ATK-NEO-6M GPS 模块信息

可以看到，此时用于定位的卫星数更多了，有 13 颗，其中，信噪比 40 以上的有 7 颗！信号强度非常不错，可见 ATK-NEO-6M GPS 模块自带的无源天线加上 LNA 后，起搜星能力不弱于有源天线了。

模块在定位成功后，可以看到 ATK-NEO-6M GPS 模块的蓝色灯开始闪烁。模块默认的 NMEA 数据输出速度为 1Hz；默认的 PPS 蓝灯闪烁情况为 100ms 灭，900ms 亮。

我们还可以通过 usmart 调用：Ublox_Cfg_Cfg_Save、Ublox_Cfg_Msg、Ublox_Cfg_Prt、Ublox_Cfg_Rate、Ublox_Cfg_Tp 等 5 个两个函数，来改变 ATK-NEO-6M 模块的配置参数。如图 4.3 所示（注意先断开 u-center 的连接）：



图 4.3 通过 usmart 改变模块默认设置状况

通过如图 4.3 所示的几个函数调用，我们可以改变模块的配置。

Ublox_Cfg_Tp(1000000,100000,0XFF)，这个函数，用于设置模块的 PPS 信号，设置 PPS 信号周期为 1000ms，脉冲宽度为 100ms，低电平有效，也就是 PPS 灯每秒钟只亮 100ms。

Ublox_Cfg_Rate(200,1)，这个函数，用于设置模块的输出时间间隔为 200ms，也就是输出速度为 5Hz。

Ublox_Cfg_Msg(3,0)，这个函数，用于设置模块的输出消息中禁止 GPGSV 消息的输出。

以上三个函数，设置完以后，我们可以按开发板的 KEY0 按键，让 STM32 输出 NMEA-0183 数据，如图 4.4 所示：

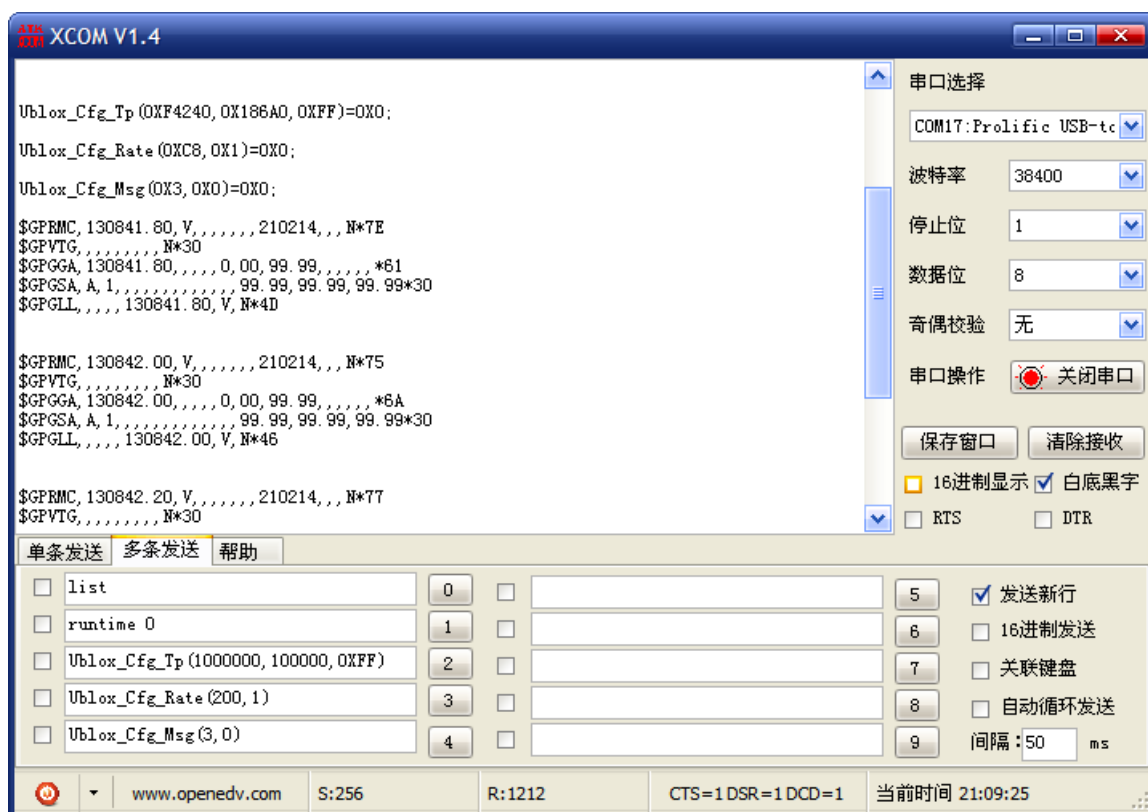


图 4.4 修改后的 NMEA-0183 输出数据

可见，输出的 NMEA-0183 协议数据中，没有了 GPGSV 这帧数据了。

至此，关于 ATK-NEO-6M GPS 模块的介绍，我们就讲完了，通过本文档的学习，相信大家很快学会 ATK-NEO-6M GPS 模块的使用。

正点原子@ALIENTEK

公司网址: www.alientek.com

技术论坛: www.openedv.com

电话: 020-38271790

传真: 020-36773971

