AN1808 ATK-S1216F8 GPS/北斗模块使用说明

本应用文档（AN1808，对应 **NANO STM32 开发板扩展实验 7**）将教大家如何在 ALIENTEK STM32 开发板上使用 ATK-S1216F8-BD GPS/北斗模块，并实现 GPS/北斗定位。

本文档分为如下几部分：

1， ATK-S1216F8-BD GPS/北斗模块简介

2， 硬件连接

3， 软件实现

4， 验证

1、ATK-S1216F8-BD GPS/北斗模块简介

ATK-S1216F8-BD-V23 模块，是 ALIENTEK 生产的一款高性能 GPS/北斗模块，模块核心采用 SkyTraq 公司的 S1216F8-BD 模组，具有 167 个通道，追踪灵敏度高达-165dBm，测量输出频率最高可达 20Hz。ATK-S1216F8-BD GSP/北斗模块具有以下特点：

1， 模块采用 S1216F8-BD 模组，体积小巧，性能优异。

2， 模块可通过串口进行各种参数设置，并可保存在内部 FLASH，使用方便。

3， 模块自带 IPX 接口，可以连接各种有源天线，适应能力强。

4， 模块兼容 3.3V/5V 电平，方便连接各种单片机系统。

5， 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/北斗星历数据的保存，以支持温启动或热启动，从而实现快速定位。

ATK-S1216F8-BD GPS/北斗模块非常小巧（25mm*27mm），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-S1216F8-BD GPS/北斗模块外观图

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	VCC	电源（3.3V~5.0V）
2	GND	地
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	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
5	PPS	时钟脉冲输出脚

表 1.1 ATK-S1216F8-BDGPS/北斗模块各引脚功能描述

其中，PPS 引脚同时连接到了模块自带了的状态指示灯：PPS，该引脚连接在 UBLOX NEO-6M 模组的 1PPS 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（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，用来外接一个有源天线，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-S1216F8-BD 模块默认采用 NMEA-0183 协议输出 GPS/北斗定位数据，并可以通过 SkyTraq 协议对模块进行配置，NMEA-0183 协议详细介绍请参考《ATK-S1216F8-BD GPS/北斗用户手册.pdf》，SkyTraq 配置协议，请参考《Binary Messages of SkyTraq Venus 8 GNSS Receiver.pdf》。

通过 ATK-S1216F8-BD GPS/北斗模块，任何单片机（3.3V/5V 电源）都可以很方便的实现 GPS/北斗定位，当然他也可以连接电脑，利用电脑软件实现定位。ATK-S1216F8-BD GPS/北斗模块的原理图如图 1.2 所示：

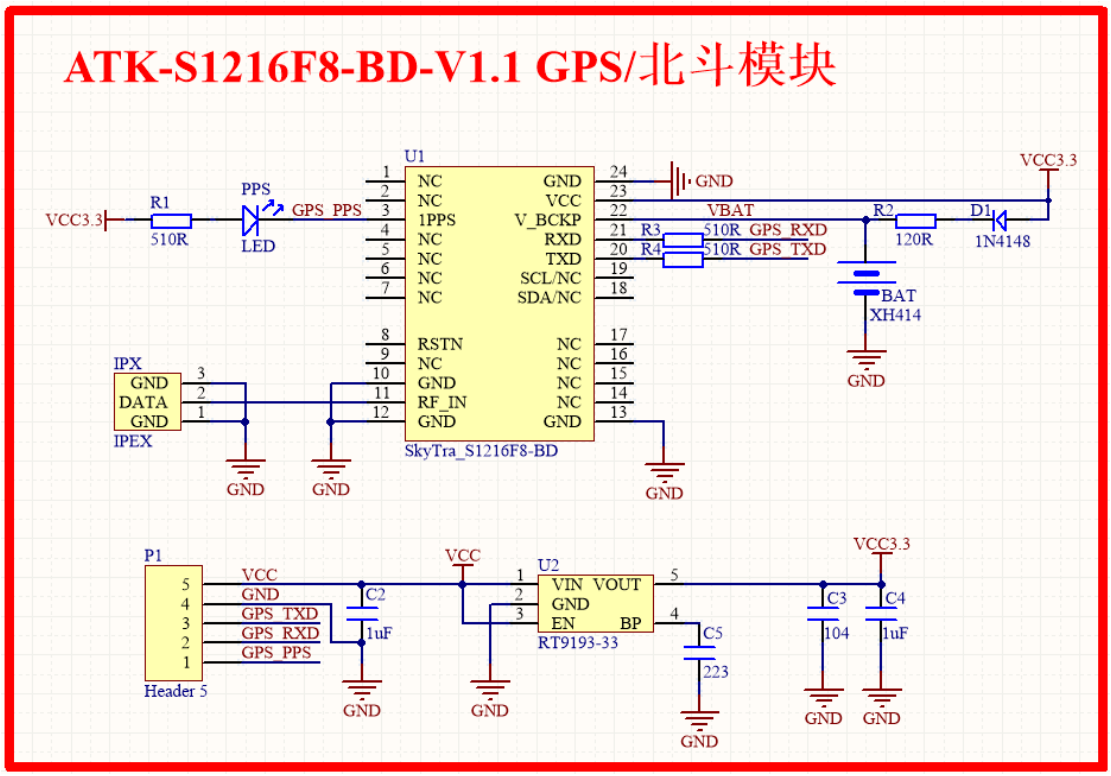

图 1.2 ATK-S1216F8-BD GPS/北斗模块原理图

2、硬件连接

本实验功能简介：通过串口 2 连接 ATK-S1216F8-BD GPS/北斗模块，然后可通过 KEYUP 按键控制串口 1 显示 GPS/北斗信息，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，通过 KEYUP 按键开启 USMART 工具，设置 GPS/北斗模块的刷新速率（最大支持 20Hz 刷新率）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 2

4， TFTLCD 模块

5， ATK-S1216F8-BD GPS/北斗模块

接下来，我们看看 ATK-S1216F8-BD GPS/北斗模块同 ALIENTEK STM32 开发板的连接，前面我们介绍了 ATK-S1216F8-BD 模块的接口，我们通过杜邦线连接模块和开发板的相应端口，连接关系如表 2.1 所示：

ATK-S1216F8-BD GPS/北斗模块与开发板连接关系				
ATK-S1216F8-BD GPS/北斗模块	VCC	GND	TXD	RXD
ALIENTEK STM32 开发板	3.3V/5V	GND	PA3	PA2

表 2.1 ATK-S1216F8-BD GPS/北斗模块同 ALIENTEK STM32 开发板连接关系表

表中 ATK-S1216F8-BD GPS/北斗模块的 VCC，因为我们的模块是可以 3.3V 或 5V 供电的，所以可以接开发板的 3.3V 电源，也可以接开发板的 5V 电源，这个随便大家自己选择。另外，这里我们没有用到模块的 PPS 引脚，所以没有和单片机进行连接。

3、软件实现

本实验在扩展例程：**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"
////////////////////////////////////////////////////////////////////////////////
const u32 BAUD_id[9]={4800,9600,19200,38400,57600,115200,230400,460800,921600};
//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--)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 BDGSV 信息
    //gpsx:nmea 信息结构体
    //buf:接收到的北斗数据缓冲区首地址
    void NMEA_BDGSV_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,"$BDGSV");
        len=p1[7]-'0';                                  //得到 BDGSV 的条数
        posx=NMEA_Comma_Pos(p1,3);                      //得到可见北斗卫星总数
        if(posx!=0XFF)gpsx->beidou_svnum=NMEA_Str2num(p1+posx,&dx);
        for(i=0;i<len;i++)
        {
            p1=(u8*)strstr((const char *)p,"$BDGSV");
            for(j=0;j<4;j++)
            {
                posx=NMEA_Comma_Pos(p1,4+j*4);

    if(posx!=0XFF)gpsx->beidou_slmsg[slx].beidou_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->beidou_slmsg[slx].beidou_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->beidou_slmsg[slx].beidou_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->beidou_slmsg[slx].beidou_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 BDGSV 信息
        }
}
//分析 GNGGA 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS/北斗数据缓冲区首地址
void NMEA_GNGGA_Analysis(nmea_msg *gpsx,u8 *buf)
{
    u8 *p1,dx;
    u8 posx;
    p1=(u8*)strstr((const char *)buf,"$GNGGA");
    posx=NMEA_Comma_Pos(p1,6);                          //得到 GPS 状态
    if(posx!=0XFF)gpsx->gpssta=NMEA_Str2num(p1+posx,&dx);
    posx=NMEA_Comma_Pos(p1,7);
    //得到用于定位的卫星数
    if(posx!=0XFF)gpsx->posslnum=NMEA_Str2num(p1+posx,&dx);
    posx=NMEA_Comma_Pos(p1,9);                          //得到海拔高度
    if(posx!=0XFF)gpsx->altitude=NMEA_Str2num(p1+posx,&dx);
}
//分析 GNGSA 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS/北斗数据缓冲区首地址
void NMEA_GNGSA_Analysis(nmea_msg *gpsx,u8 *buf)
{
    u8 *p1,dx;
    u8 posx;
    u8 i;
    p1=(u8*)strstr((const char *)buf,"$GNGSA");
    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 GNRMC 信息
    //gpsx:nmea 信息结构体
    //buf:接收到的 GPS/北斗数据缓冲区首地址
    void NMEA_GNRMC_Analysis(nmea_msg *gpsx,u8 *buf)
    {
        u8 *p1,dx;
        u8 posx;
        u32 temp;
        float rs;
        p1=(u8*)strstr((const char *)buf,"$GNRMC");
        //"$GNRMC",经常有&和 GNRMC 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,去掉 ms
            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 GNVTG 信息
//gpsx:nmea 信息结构体
//buf:接收到的 GPS/北斗数据缓冲区首地址
void NMEA_GNVTG_Analysis(nmea_msg *gpsx,u8 *buf)
{
        u8 *p1,dx;
        u8 posx;
        p1=(u8*)strstr((const char *)buf,"$GNVTG");
        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/北斗数据缓冲区首地址
void GPS_Analysis(nmea_msg *gpsx,u8 *buf)
{
        NMEA_GPGSV_Analysis(gpsx,buf);   //GPGSV 解析
        NMEA_BDGSV_Analysis(gpsx,buf);   //BDGSV 解析
        NMEA_GNGGA_Analysis(gpsx,buf);  //GNGGA 解析
        NMEA_GNGSA_Analysis(gpsx,buf);   //GNGSA 解析
```

```
        NMEA_GNRMC_Analysis(gpsx,buf);  //GNRMC 解析
        NMEA_GNVTG_Analysis(gpsx,buf);  //GNVTG 解析
    }
    ////////////////////////////////////////SkyTraq 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 SkyTra_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<100)          //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]==0X83)break;
                else if(USART3_RX_BUF[i]==0X84)
                {
                    rval=3;
                    break;
                }
            }
            if(i==len)rval=2;               //没有找到同步字符
        }else rval=1;                       //接收超时错误
        USART3_RX_STA=0;                    //清除接收
        return rval;
    }
    //配置 SkyTra_GPS/北斗模块波特率
    //baud_id:0~8，对应波特率,4800/9600/19200/38400/57600/115200/230400/460800/921600
    //返回值:0,执行成功;其他,执行失败(这里不会返回 0 了)
    u8 SkyTra_Cfg_Prt(u32 baud_id)
    {
        SkyTra_baudrate *cfg_prt=(SkyTra_baudrate *)USART3_TX_BUF;
        cfg_prt->sos=0XA1A0;        //引导序列(小端模式)
        cfg_prt->PL=0X0400;             //有效数据长度(小端模式)
        cfg_prt->id=0X05;           //配置波特率的 ID
```

```
    cfg_prt->com_port=0X00;                 //操作串口 1
    cfg_prt->Baud_id=baud_id;               ////波特率对应编号
    cfg_prt->Attributes=1;                  //保存到 SRAM&FLASH
    cfg_prt->CS=cfg_prt->id^cfg_prt->com_port^cfg_prt->Baud_id^cfg_prt->Attributes;
    cfg_prt->end=0X0A0D;                    //发送结束符(小端模式)
    SkyTra_Send_Date((u8*)cfg_prt,sizeof(SkyTra_baudrate));//发送数据给 SkyTraF8
    delay_ms(200);                          //等待发送完成
    usart3_init(36,BAUD_id[baud_id]);       //重新初始化串口 3
    return SkyTra_Cfg_Ack_Check();
//这里不会返回 0,因为 SkyTra 发回来的应答在串口重新初始化的时候已经被丢弃了.
}
//配置 SkyTra_GPS/北斗模块的时钟脉冲宽度
//width:脉冲宽度 1~100000(us)
//返回值:0,发送成功;其他,发送失败.
u8 SkyTra_Cfg_Tp(u32 width)
{
    u32 temp=width;
    SkyTra_pps_width *cfg_tp=(SkyTra_pps_width *)USART3_TX_BUF;
    temp=(width>>24)|((width>>8)&0X0000FF00)|((width<<8)&
        0X00FF0000)|((width<<24)&0XFF000000);  //小端模式
    cfg_tp->sos=0XA1A0;                 //cfg header(小端模式)
    cfg_tp->PL=0X0700;                  //有效数据长度(小端模式)
    cfg_tp->id=0X65   ;                 //cfg tp id
    cfg_tp->Sub_ID=0X01;                //数据区长度为 20 个字节.
    cfg_tp->width=temp;                 //脉冲宽度,us
    cfg_tp->Attributes=0X01;            //保存到 SRAM&FLASH
    cfg_tp->CS=cfg_tp->id^cfg_tp->Sub_ID^(cfg_tp->width>>24)^(cfg_tp->width>>16)
    &0XFF^(cfg_tp->width>>8)&0XFF^cfg_tp->width&0XFF^cfg_tp->Attributes;
    cfg_tp->end=0X0A0D;                 //发送结束符(小端模式)
    SkyTra_Send_Date((u8*)cfg_tp,sizeof(SkyTra_pps_width));     //发送数据给 SkyTra
    return SkyTra_Cfg_Ack_Check();
}
//配置 SkyTraF8-BD 的更新速率
//Frep:（取值范围:1,2,4,5,8,10,20）测量时间间隔，单位为 Hz，最大不能大于 20Hz
//返回值:0,发送成功;其他,发送失败.
u8 SkyTra_Cfg_Rate(u8 Frep)
{
    SkyTra_PosRate *cfg_rate=(SkyTra_PosRate *)USART3_TX_BUF;
    cfg_rate->sos=0XA1A0;               //cfg header(小端模式)
    cfg_rate->PL=0X0300;                //有效数据长度(小端模式)
    cfg_rate->id=0X0E;                  //cfg rate id
    cfg_rate->rate=Frep;                //更新速率
    cfg_rate->Attributes=0X01;          //保存到 SRAM&FLASH     .
    cfg_rate->CS=cfg_rate->id^cfg_rate->rate^cfg_rate->Attributes;//脉冲间隔,us
```

```
    cfg_rate->end=0X0A0D;          //发送结束符(小端模式)
    SkyTra_Send_Date((u8*)cfg_rate,sizeof(SkyTra_PosRate));     //发送数据给 SkyTra
    return SkyTra_Cfg_Ack_Check();
}
//发送一批数据给 SkyTraF8-BD，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 SkyTra_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 SkyTraq 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$GPGSV 的开头，然后固定输出格式，不论是否有数据输出，逗号是肯定会有的，而且都会以‘*’作为有效数据的结尾，所以，我们了解了 NMEA-0183 协议的数据格式（在 ATK-S1216F8-BD GPS/北斗模块的用户手册有详细介绍）之后，就可以通过数逗号的方法，来解析数据了。本代码实现了对 NMEA-0183 协议的$GNGGA、$GPGSA、$GNGSV、$BDGSV、$GNRMC 和$GNVTG 等 6 类帧的解析，结果存放在通过 gps.h 定义的 nmea_msg 结构体内。

SkyTraq 协议控制部分，此部分我们只实现了 SkyTraF8-BD 模组常用的 3 个配置：串口波特率设置、PPS 输出脉冲宽度设置、输出频率设置。

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

PPS 输出脉冲宽度设置，通过函数 SkyTra_Cfg_Tp 实现，可以设置脉冲宽度（1us~100ms）。

输出频率设置，通过函数 SkyTraq_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 20Hz 的测量输出频率。

最后 SkyTraq_Send_Date 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 3 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;
//北斗 NMEA-0183 协议重要参数结构体定义
//卫星信息
__packed typedef struct
{
    u8 beidou_num;     //卫星编号
    u8 beidou_eledeg;  //卫星仰角
    u16 beidou_azideg;//卫星方位角
    u8 beidou_sn;      //信噪比
}beidou_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 GPS 卫星数
    u8 beidou_svnum;            //可见北斗卫星数
    nmea_slmsg slmsg[12];       //最多 12 颗 GPS 卫星
    beidou_nmea_slmsg beidou_slmsg[12];//暂且算最多 12 颗北斗卫星
    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 GPS 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;
    ////////////////////////////////////////////////////////////////////////////////////////////////
//SkyTra S1216F8 配置波特率结构体
__packed typedef struct
{
    u16 sos;            //启动序列，固定为 0XA0A1
    u16 PL;             //有效数据长度 0X0004;
    u8 id;              //ID，固定为 0X05
    u8 com_port;        //COM 口，固定为 0X00，即 COM1
    u8 Baud_id;
    //波特率（0~8,4800,9600,19200,38400,57600,115200,230400,460800,921600）
    u8 Attributes;
    //配置数据保存位置 ,0 保存到 SRAM，1 保存到 SRAM&FLASH，2 临时保存
    u8 CS;              //校验值
    u16 end;            //结束符:0X0D0A
}SkyTra_baudrate;
////////////////////////////////////////////////////////////////////////////////////////////////
//SkyTra S1216F8-BD 配置输出信息结构体
__packed typedef struct
{
    u16 sos;            //启动序列，固定为 0XA0A1
    u16 PL;             //有效数据长度 0X0009;
    u8 id;              //ID，固定为 0X08
    u8 GGA;             //1~255（s）,0:disable
    u8 GSA;             //1~255（s）,0:disable
    u8 GSV;             //1~255（s）,0:disable
    u8 GLL;             //1~255（s）,0:disable
    u8 RMC;             //1~255（s）,0:disable
    u8 VTG;             //1~255（s）,0:disable
    u8 ZDA;             //1~255（s）,0:disable
    u8 Attributes;
     //配置数据保存位置 ,0 保存到 SRAM，1 保存到 SRAM&FLASH，2 临时保存
    u8 CS;              //校验值
    u16 end;            //结束符:0X0D0A
}SkyTra_outmsg;
////////////////////////////////////////////////////////////////////////////////////////////////
//SkyTra S1216F8-BD 配置位置更新率结构体
__packed typedef struct
{
```

```
    u16 sos;                //启动序列，固定为 0XA0A1
    u16 PL;                 //有效数据长度 0X0003；
    u8 id;                  //ID，固定为 0X0E
    u8 rate;                //取值范围:1, 2, 4, 5, 8, 10, 20, 25, 40, 50
    u8 Attributes;
    //配置数据保存位置 ,0 保存到 SRAM，1 保存到 SRAM&FLASH，2 临时保存
    u8 CS;                  //校验值
    u16 end;                //结束符:0X0D0A
}SkyTra_PosRate;
////////////////////////////////////////////////////////////////////////////////////////////
//SkyTra S1216F8-BD 配置输出脉冲(PPS)宽度结构体
__packed typedef struct
{
    u16 sos;                //启动序列，固定为 0XA0A1
    u16 PL;                 //有效数据长度 0X0007；
    u8 id;                  //ID，固定为 0X65
    u8 Sub_ID;              //0X01
    u32 width;              //1~100000(us)
    u8 Attributes;
    //配置数据保存位置 ,0 保存到 SRAM，1 保存到 SRAM&FLASH，2 临时保存
    u8 CS;                  //校验值
    u16 end;                //结束符:0X0D0A
}SkyTra_pps_width;
////////////////////////////////////////////////////////////////////////////////////////////
//SkyTra S1216F8-BD  ACK 结构体
__packed typedef struct
{
    u16 sos;                //启动序列，固定为 0XA0A1
    u16 PL;                 //有效数据长度 0X0002；
    u8 id;                  //ID，固定为 0X83
    u8 ACK_ID;
    u8 CS;                  //校验值
    u16 end;                //结束符
}SkyTra_ACK;
////////////////////////////////////////////////////////////////////////////////////////////
//SkyTra S1216F8 -BD NACK 结构体
__packed typedef struct
{
    u16 sos;                //启动序列，固定为 0XA0A1
    u16 PL;                 //有效数据长度 0X0002；
    u8 id;                  //ID，固定为 0X84
    u8 NACK_ID;
    u8 CS;                  //校验值
    u16 end;                //结束符
```

```
}SkyTra_NACK;

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_BDGSV_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GNGGA_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GNGSA_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GNGSA_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GNRMC_Analysis(nmea_msg *gpsx,u8 *buf);
void NMEA_GNVTG_Analysis(nmea_msg *gpsx,u8 *buf);
u8 SkyTra_Cfg_Prt(u32 baud_id);
u8 SkyTra_Cfg_Tp(u32 width);
u8 SkyTra_Cfg_Rate(u8 Frep);
void SkyTra_Send_Date(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 main.c 里面，修改代码如下：

```
u8 USART1_TX_BUF[USART2_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;
    tp=gpsx.longitude;
    sprintf((char *)dtbuf,"Longitude:%.5f %1c     ",tp/=100000,gpsx.ewhemi);    // 得到经度字符串
    printf("%s\r\n",dtbuf);
    tp=gpsx.latitude;
    sprintf((char *)dtbuf,"Latitude:%.5f %1c     ",tp/=100000,gpsx.nshemi);    // 得到纬度字符串
    printf("%s\r\n",dtbuf);
    tp=gpsx.altitude;
    sprintf((char *)dtbuf,"Altitude:%.1fm        ",tp/=10);                    // 得到高度字符串
    printf("%s\r\n",dtbuf);
    tp=gpsx.speed;
    sprintf((char *)dtbuf,"Speed:%.3fkm/h        ",tp/=1000);                  // 得到速度字符串
```

```
        printf("%s\r\n",dtbuf);
        if(gpsx.fixmode<=3)
    //定位状态
        {
            sprintf((char *)dtbuf,"Fix Mode:%s",fixmode_tbl[gpsx.fixmode]);
            printf("%s\r\n",dtbuf);
        }
        sprintf((char *)dtbuf,"GPS+BD Valid satellite:%02d",gpsx.posslnum);          // 用 于
定位的 GPS 卫星数
        printf("%s\r\n",dtbuf);
        sprintf((char *)dtbuf,"GPS Visible satellite:%02d",gpsx.svnum%100);          // 可 见
GPS 卫星数
        printf("%s\r\n",dtbuf);
        sprintf((char *)dtbuf,"BD Visible satellite:%02d",gpsx.beidou_svnum%100);//可见北斗
卫星数
        printf("%s\r\n",dtbuf);
        sprintf((char                *)dtbuf,"UTC                Date:%04d/%02d/%02d
",gpsx.utc.year,gpsx.utc.month,gpsx.utc.date);//显示 UTC 日期
        printf("%s\r\n",dtbuf);
        sprintf((char                *)dtbuf,"UTC                Time:%02d:%02d:%02d
",gpsx.utc.hour,gpsx.utc.min,gpsx.utc.sec);//显示 UTC 时间
        printf("%s\r\n",dtbuf);
    }

    int main(void)
    {

        u16 i,rxlen;
        u16 lenx;
        u8 key=0XFF;
        u8 upload=0;
        u8 stop=0;

        HAL_Init();                                 //初始化 HAL 库
        Stm32_Clock_Init(RCC_PLL_MUL9); //设置时钟,72M
        delay_init(72);                             //初始化延时函数
        uart_init(115200);                          //初始化串口 115200
        USART2_Init(38400);                         //串口 2 初始化
        LED_Init();                                 //初始化 LED
        KEY_Init();                                 //初始化按键
        usmart_init(72);                            //USMART 初始化
        printf("ALIENTEK NANO STM32\r\n");
        printf("SkyTraF8-BD TEST\r\n");
        if(SkyTra_Cfg_Rate(5)!=0)
```

```
        //设置定位信息更新速度为 5Hz,顺便判断 GPS 模块是否在位.
    {
        printf("SkyTraF8-BD 配置中...、\r\n");
        do
        {
            USART2_Init(9600);              //初始化串口 3 波特率为 9600
            SkyTra_Cfg_Prt(3);              //重新设置模块的波特率为 38400
            USART2_Init(38400);             //初始化串口 3 波特率为 38400
            key=SkyTra_Cfg_Tp(100000);      //脉冲宽度为 100ms
        }while(SkyTra_Cfg_Rate(5)!=0&&key!=0);
        //配置 SkyTraF8-BD 的更新速率为 5Hz
        printf("SkyTraF8-BD 设置完成\r\n");
    }
    while(1)
    {
         delay_ms(1);
         if(USART2_RX_STA&0X8000) //接收到一次数据了
         {
            rxlen=USART2_RX_STA&0X7FFF;//得到数据长度
            for(i=0;i<rxlen;i++)USART1_TX_BUF[i]=USART2_RX_BUF[i];
            USART2_RX_STA=0;                //启动下一次接收
            USART1_TX_BUF[i]=0;             //自动添加结束符

             if(!stop)
             {
                if(upload) printf("\r\n%s\r\n",USART1_TX_BUF);
                                //发送接收到的数据到串口 1
                else
                {
                    GPS_Analysis(&gpsx,(u8*)USART1_TX_BUF);//分析字符串
                    Gps_Msg_Show(); //显示信息
                }
             }
         }
         key=KEY_Scan(0);
         if(key==KEY1_PRES)//控制 GPS 数据是否上传上位机
         {
            upload=!upload;
         }else if(key==WKUP_PRES)//控制 USMART 调试
         {
            stop=!stop;
            printf("USMART ON!\r\n");
         }
         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 SkyTraq_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/北斗数据。通过 KEY_UP 按键，使能或失能 USMART 调试，使能 USMART 调试，则 GPS/北斗数据不串口输出，失能则可数据输出。（不建议在 GPS/北斗数据串口输出时进行 USMART 的调试）

在 USMART 调试失能下，每次接收到 GPS/北斗模块发送过来的数据，通过 KEY1 按键可切换是执行数据解析还是将原始的数据串口输出，原始的数据可发送给上位机解析显示。

至此，整个 ATK-S1216F8-BD GPS/北斗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 NANO STM32 开发板上（假设 ATK-S1216F8-BD GPS/北斗模块已经正确连接到开发板，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），则串口一直显示：

SkyTraF8-BD 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-S1216F8-BD GPS/北斗模块→ATK-S1216F8-BD GPS/北斗模块问题汇总.pdf）。排除问题根源后，就会进入到下一步，正常到模块以后，串口调试助手显示如图 4.1 所示内容：

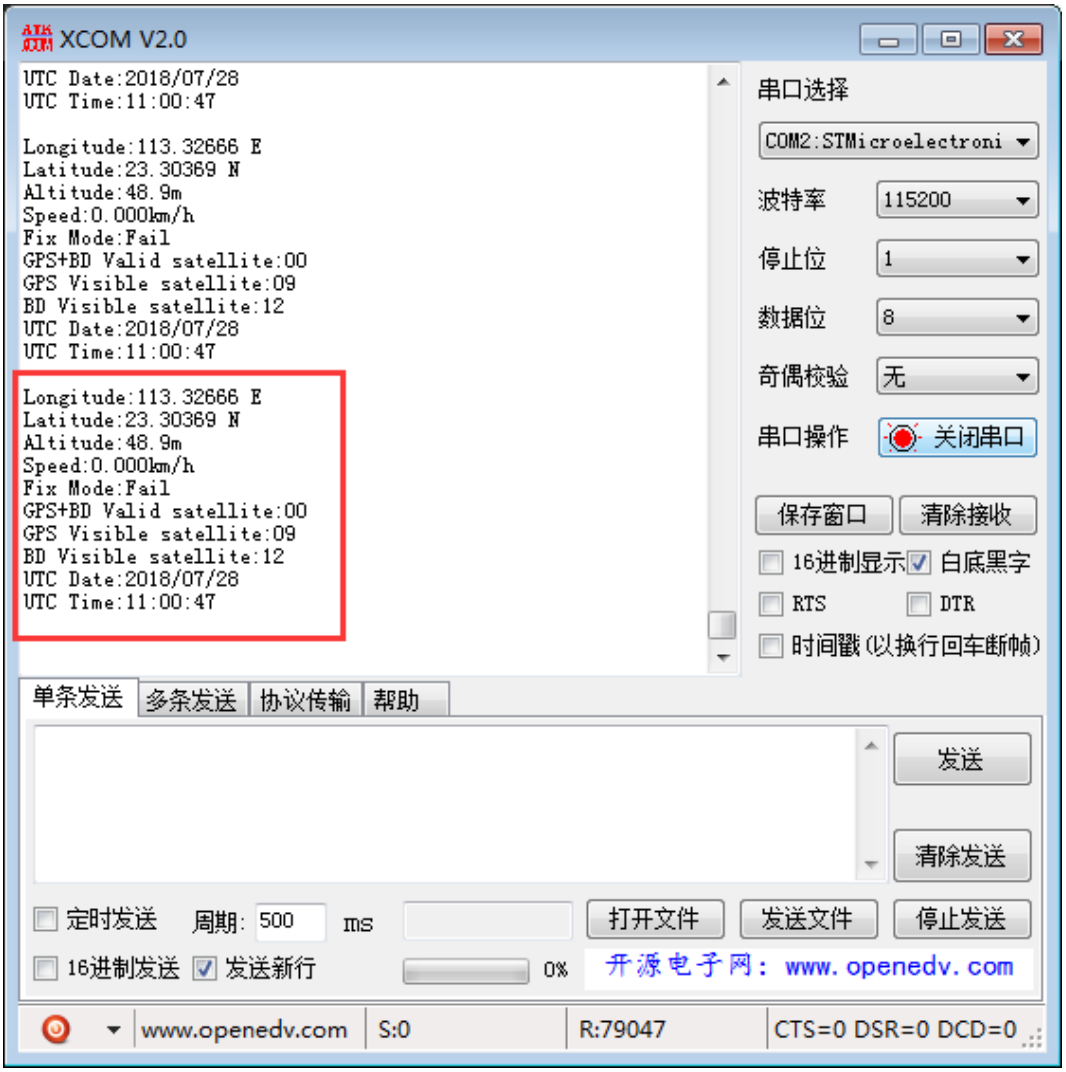

图 4.1 串口调试助手显示界面

上图是我们的 GPS/北斗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-S1216F8-BD GPS/北斗模块，用于定位的卫星达到 9 颗，可见的 GPS 和北斗卫星一共 21 颗！

我们打开 GNSS_Viewer 软件，连接开发板，并按一下开发板的 KEY1 按键，程序上传 NMEA 数据到电脑，可以看到 GNSS_Viewer 软件显示如图 4.2 所示

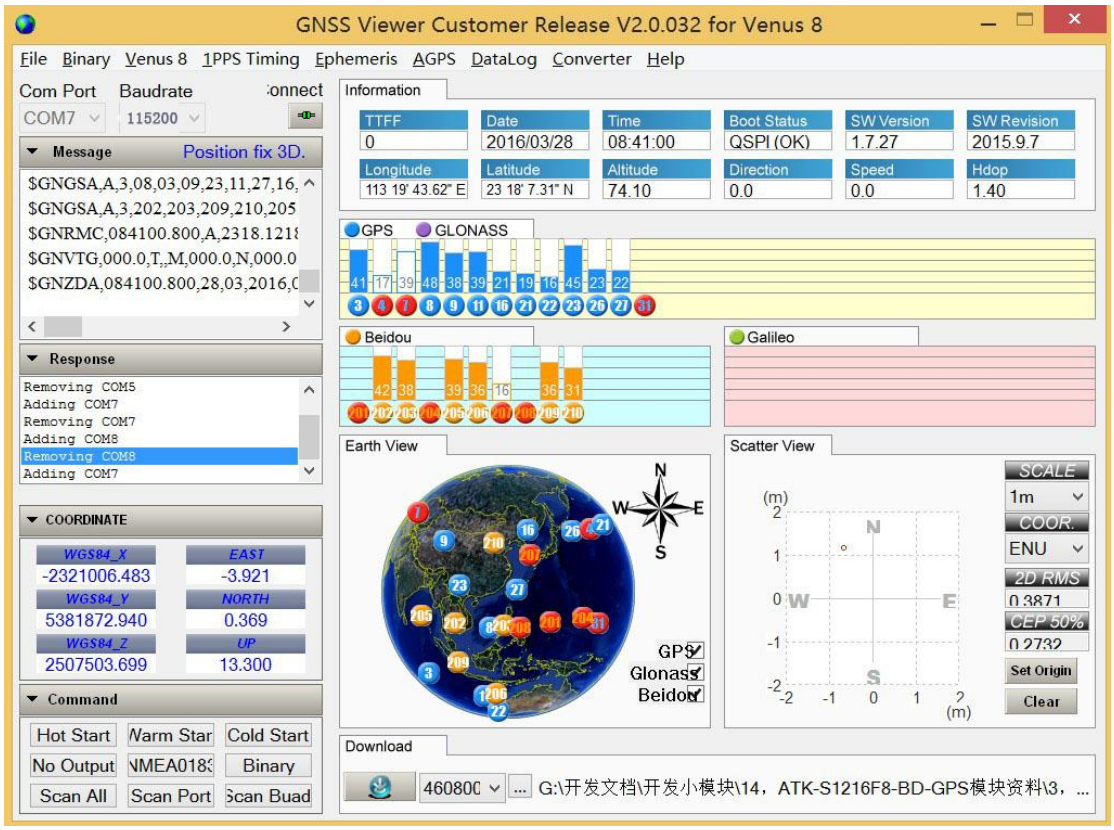

图 4.2 GNSS_Viewer 显示 ATK-S1216F8-BD GPS/北斗模块信息

可以看到，此时用于定位的卫星数更多了，有 16 颗，其中，GPS 用于定位的 10 颗，北斗用于定位的卫星 6 颗卫星，可见 ATK-S1216F8-BD GPS/北斗模块配合有源天线实现定位的效果挺好的。

模块在定位成功后，可以看到 ATK-S1216F8-BD GPS/北斗模块的蓝色灯开始闪烁。模块默认的 NMEA 数据输出速度为 5Hz；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： SkyTraq_Cfg_Tp、SkyTraq_Cfg_Rate 等 2 个两个函数，来改变 ATK-S1216F8-BD GPS/北斗模块的配置参数。如图 4.3 所示（注意断开 GNSS_Viewer 的连接，按下 KEY_UP 按键）：

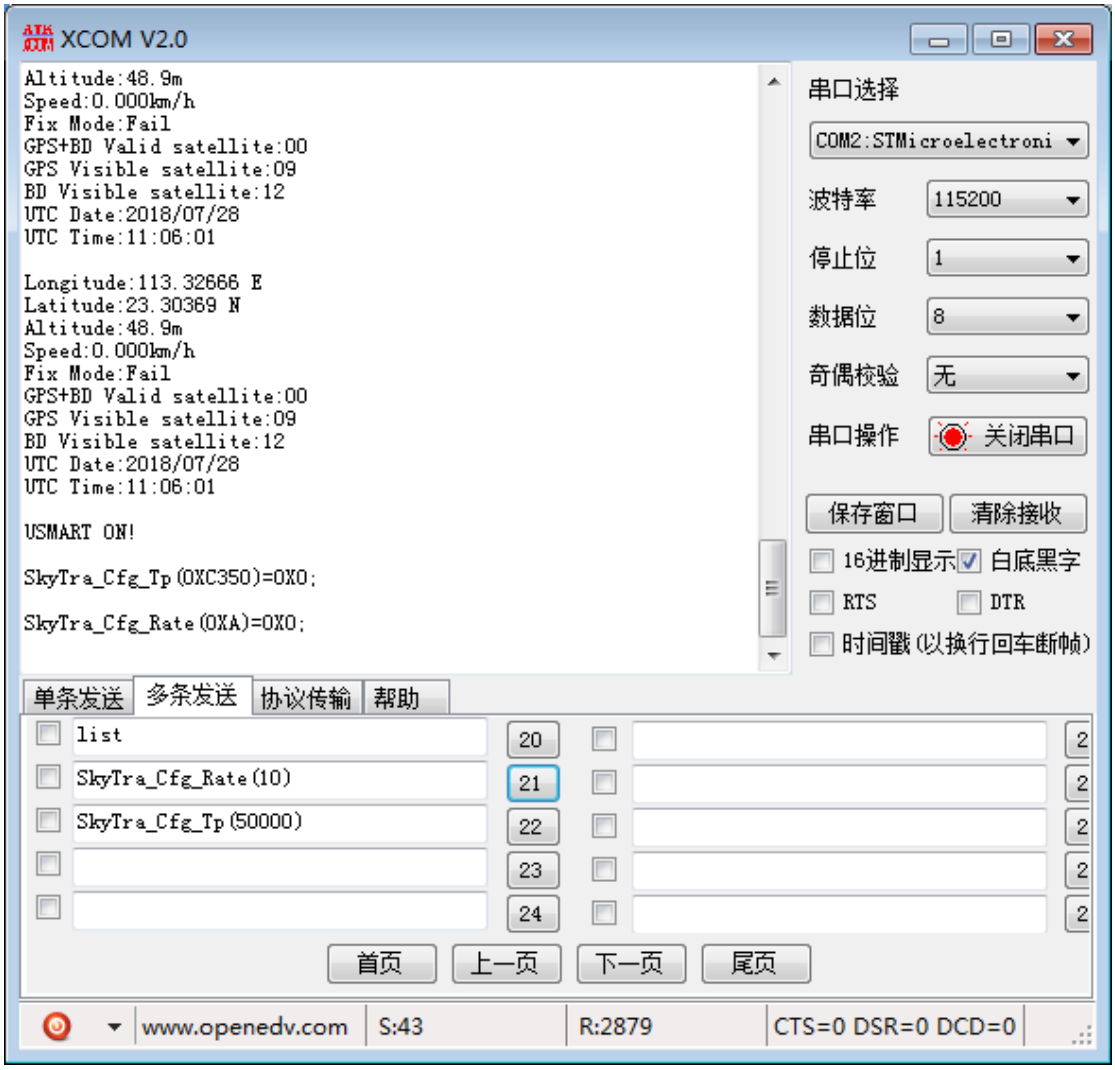

图 4.3 通过 usmart 改变模块默认设置状况

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。

SkyTraq_Cfg_Tp(50000)，这个函数，用于设置模块的 PPS 输出脉冲宽度为 50000us，也就是 50ms。

SkyTraq_Cfg_Rate(10)，这个函数，用于设置模块的定位信息输出频率为 10Hz。

以上三个函数，设置完以后，PPS 脉冲宽度为 50ms，输出信息更新速率为 10Hz；

至此，关于 ATK-S1216F8-BD GPS/北斗模块的介绍，我们就讲完了，通过本文档的学习，相信大家可以很快学会 ATK-S1216F8-BD GPS/北斗模块的使用。

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