NMEA Software Standard

Introduction

A software standard for commercial GPS receivers is NMEA 0183 (www.nmea.org). This is a serial protocol using ASCII *sentences* to convey information from the device, in this case, a GPS receiver. The standard baud rate is 4800 with a word length of 8 (bit 7 cleared), 1 stop bit, and no parity. PCM-GPS boards as shipped, default to this NMEA 0183 standard and begin transmitting NMEA sentences immediately after power-up.

The Trimble Lassen® IQ GPS module is also capable of transmitting and receiving serial data in a Trimble proprietary format known as TSIP, the factory standard. This is a binary protocol which ordinarily runs at 9600 baud, an 8-bit word, and odd parity. Users requiring a TSIP interface to the GPS should contact WinSystems Technical Support for details on converting the PCM-GPS to TSIP.

The sections that follow will document the NMEA sentences sent by the PCM-GPS and provide **C** source code examples for decoding and utilizing the NMEA data in an application.

NMEA 0183 Sentence Structure

The NMEA 0183 protocol covers a broad array of navigation data. This array of information is separated into discrete sentences which convey a specific set of data. The entire protocol encompasses over 50 sentences, but only a subset of the sentences apply to a GPS receiver like the Trimble unit on the PCM-GPS. The NMEA sentence structure is described below:

$IDMSG,D1,D2,D3,D4,....,Dn*CS[CR][LF]

Parameter	Description
$	The '$' signifies the start of a sentence.
ID	The talker identification is a two letter mnemonic which describes the source of the navigation information. The identification for GPS is 'GP'.
MSG	The sentence identification is a three letter mnemonic which describes the sentence content and the number and order of the data fields.
,	Commas serve as delimiters between the data fields.
Dn	Each sentence contains multiple data fields (Dn) delimited by the commas.
*****	The asterisk serves as the checksum delimiter.
CS	The checksum field contains two ASCII characters which indicate the hexadecimal value of the checksum.
[CR][LF]	The carriage return [CR] and the line feed [LF] combination terminate the sentence.

NMEA 0183 sentences vary in length, but each sentence is limited to 79 characters or less. This length limit excludes the **$** and the **[CR][LF]** characters. The data field block, including delimiters is limited to 74 characters or less.

Supported NMEA 0183 Sentences

The PCM-GPS supports up to seven unique NMEA 0183 data sentences. The factory default for the board is to output all seven sentences once each second. The supported sentences are documented in the sections that follow.

GGA - GPS Fix Data

The GGA sentence includes time, position, and fix related data for the GPS receiver. The GGA sentence structure is shown here :

$GPGGA,hhmmss.ss,llll.lll,a,nnnnn.nnn,b,t,uu,v.v,w.w,M,x.x,M,y.y,zzzz*hh<CR><LF>

Parameter	Description
hhmmss.ss	*hoursminutesseconds.decimal* : 2 Fixed digits for hours, 2 fixed digits for minutes, 2 fixed digits for seconds and a variable number of digits for decimal seconds. Leading zeros are always included for hours, minutes, and seconds to maintain fixed length. The decimal point and the associated decimal value are optional if full resolution is not needed.
llll.lll	*Degreesminutes.decimal* : Latitude value is 2 fixed digits of degrees, 2 fixed digits of minutes and a variable number of digits for decimal fractions of minutes. Leading zeros are always included for degrees and minutes to maintain fixed length. The decimal point and decimal fraction value are optional.
a	Hemispherical orientation *N* or *S*. Single character indicates latitude hemisphere.
nnnnn.nnn	*Degreesminutes.decimal* : Longitude value is 3 fixed digits of degrees, 2 fixed digits of minutes and a variable number of digits for decimal fractions of minutes. Leading zeros are always included for degrees and minutes to maintain fixed length. The decimal point and decimal fraction value are optional.
b	Hemispherical orientation *E* or *W*. Single character indicates longitude hemisphere.
t	GPS Quality indicator. *0* = No GPS, *1* = GPS, *2* = DGPS
uu	Number of satellites in use.
v.v	Horizontal Dilution of Precision (HDOP)
w.w	Antenna Altitude in Meters
M	Fixed character field for Meters *M*
x.x	Geoidal Separation in Meters. Geoidal separation is the difference between the WGS-84 earth ellipsoid and mean sea-level.

Parameter	Description
M	Fixed character field for Meters *M*
y.y	Age of differential GPS data. Time in seconds since last type 1 or type 9 update.
zzzz	Differential Reference Station ID (0000 to 1023)
hh	Checksum

GLL - Geographic Position - Latitude/Longitude
The GLL sentence contains the latitude and longitude, the time of the position fix and the status. The GLL sentence structure is shown here :

$GPGLL, llll.lll,a,yyyyy.yyy,b,hhmmss.ss,A,i,*hh<CR><LF>

Parameter	Description
llll.lll	*Degreesminutes.decimal* : Latitude value is 2 fixed digits of degrees, 2 fixed digits of minutes and a variable number of digits for decimal fractions of minutes. Leading zeros are always included for degrees and minutes to maintain fixed length. The decimal point and decimal fraction value are optional.
a	Hemispherical orientation *N* or *S*. Single character indicates latitude hemisphere.
yyyyy.yyy	*Degreesminutes.decimal* : Longitude value is 3 fixed digits of degrees, 2 fixed digits of minutes and a variable number of digits for decimal fractions of minutes. Leading zeros are always included for degrees and minutes to maintain fixed length. The decimal point and decimal fraction value are optional.
b	Hemispherical orientation *E* or *W*. Single character indicates longitude hemisphere.
hhmmss.ss	*hoursminutesseconds.decimal* : 2 Fixed digits for hours, 2 fixed digits for minutes, 2 fixed digits for seconds and a variable number of digits for decimal seconds. Leading zeros are always included for hours, minutes, and seconds to maintain fixed length. The decimal point and the associated decimal value are optional if full resolution is not needed.
A	Status Character. *A* = Valid, *V* = Invalid
i	Mode indicator character : *A* = Autonomous mode. *D* = Differential Mode *E* = Estimated (dead reckoning) mode *M* = Manual input mode *S* = Simulated mode *N* = Data not valid.
hh	Checksum

GSA - GPS DOP and Active Satellites
The GSA sentence indicates the GPS receiver's operating mode and lists the satellites used for navigation and the DOP values of the position solution. The GSA sentence structure is shown here:

$GPGSA,a,b,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,c.c,d.d,e.e*hh<CR><LF>

Parameter	Description
a	Mode indicator character : *A* = Automatic mode. *M* = Manual mode. In manual mode, the receiver is forced to opertae in either 2D or 3D mode. In automatic mode, the receiver is allowed to switch between 2D and 3D modes subject to the PDOP and satellite masks.
b	Hemispherical orientation *E* or *W*. Single character indicates longitude hemisphere.
X1-X2	PRN numbers of the satellites used in the position solution. When less than 12 satellites are used, the unused fields are empty.
c.c	Position dilution of precision value (PDOD)
d.d	Horizontal dilution of precision value (HDOP)
e.e	Vertical dilution of precision value (VDOP)
hh	Checksum

GSV - GPS Satellites in View
The GSV sentence identifies the GPS satellites in view, including their PRN number, elevation, azimuth, and SNR value. Each sentence contains data for four satellites. Second and third sentences are sent when more than 4 satellites are in view. Fields 1 and 2 indicate the total number of GSV sentences being sent and the number of each sentence respectively.

$GPGSV,a,b,c,d1,e1,f1,g1,d2,e2,f2,g2,d3,e3,f3,g3,d4,e4,f4,g4*hh<CR><LF>

Parameter	Description
a	Total number of GSV sentences to be sent.
b	This sentence number 1 to 3.
c	Total number of satellites in view.
d1-d4	Satellite PRN number
e1-e4	Satellite elevation in degrees (90º Maximum)
f1-f4	Satellite azimuth in degrees (000 to 359)
g1-g4	Satellite SNR (Null when not tracking)
hh	Checksum

RMC - Recommended Minimum Specific GPS/Transit Data
The RMC sentence contains the time, date, position, course and speed data provided by the GPS navigation receiver. A checksum is mandatory for this sentence and the transmission interval may not exceed 2 seconds. All data fields must be provided unless the data is temporarily unavailable.

$GPRMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,b,c.c,d.d,eeeeee,f.f,g,i*hh<CR><LF>

Parameter	Description
hhmmss.ss	*hoursminutesseconds.decimal* : 2 Fixed digits for hours, 2 fixed digits for minutes, 2 fixed digits for seconds and a variable number of digits for decimal seconds. Leading zeros are always included for hours, minutes, and seconds to maintain fixed length. The decimal point and the associated decimal value are optional if full resolution is not needed.
A	Status character. *A* = Valid, *V* = Navigation receiver warning
llll.lll	*Degreesminutes.decimal* : Latitude value is 2 fixed digits of degrees, 2 fixed digits of minutes and a variable number of digits for decimal fractions of minutes. Leading zeros are always included for degrees and minutes to maintain fixed length. The decimal point and decimal fraction value are optional.
a	Hemispherical orientation *N* or *S*. Single character indicates latitude hemisphere.
yyyy.yyy	*Degreesminutes.decimal* : Longitude value is 3 fixed digits of degrees, 2 fixed digits of minutes and a variable number of digits for decimal fractions of minutes. Leading zeros are always included for degrees and minutes to maintain fixed length. The decimal point and decimal fraction value are optional.
b	Hemispherical orientation *E* or *W*. Single character indicates longitude hemisphere.
c.c	Speed over the ground (SOG) in knots.
d.d	Track made good in degrees true.
eeeeee	Date : *ddmmyy*
f.f	Magnetic variation in degrees.
g	Variation direction *E* = East, *W* = West.

Parameter	Description
i	Position System Mode indicator character *A* = Autonomous Mode *D* = Differential Mode *E* = Estimated (Dead reckoning) Mode *M* = Manual Input Mode *S* = Simulation Mode *N* = Data not valid.
hh	Checksum

VTG - Track Made Good and Ground Speed
The VTG sentence conveys the actual track made good (COG) and the speed relative to the ground (SOG).

$GPVTG,a.a,T,b.b,M,c.c,N,d.d,K,i*hh<CR><LF>

Parameter	Description
a.a	Track made good in degrees true.
T	Character indicator for *True*
b.b	Track made good in degrees Magnetic.
M	Character indicator for *Magnetic*.
c.c	Speed over the ground in Knots.
N	Character indicator for Knots = ‘N’
d.d	Speed over the ground in kilometers per hour.
K	Character indicator for Kilometers = ‘K’
i	Mode indicator character. *A* = Autonomous Mode *D* = Differential Mode *E* = Estimated (Dead reckoning) Mode *M* = Manual Input Mode *S* = Simulation Mode *N* = Data not valid.
hh	Checksum

ZDA - Time and Date

The ZDA sentence contains the UTC time, the day of the month, the month, the year, and the local time zone.

$GPZDA,hhmmss.ss,dd,mm,yyyy,x1,x2*hh<CR><LF>

Parameter	Description
hhmmss.ss	*hoursminutesseconds.decimal* : 2 Fixed digits for hours, 2 fixed digits for minutes, 2 fixed digits for seconds and a variable number of digits for decimal seconds. Leading zeros are always included for hours, minutes, and seconds to maintain fixed length. The decimal point and the associated decimal value are optional if full resolution is not needed.
dd	Day of the month (01-31)
mm	Month of the year (01-12)
yyyy	Year
x1.xy	Unused (NULL). A GPS receiver cannot independently identify the local time zone offsets.
hh	Checksum

WARNING: **If the UTC time is not available, time output will be in GPS time until the UTC offset value is collected from the GPS satellites. When the offset becomes available, the time will jump to UTC time.**

NOTE: GPS time can be used as a time tag for the PPS output. The ZDA sentence comes out 100-500 ms after the PPS.

Sample Program

As has been illustrated by the previous section, there is a lot of information available to an application program within the NMEA 0183 sentences. Using GPS in application software usually revolves around one or more of the following capabilities of GPS.

1. To resolve time. The atomic clocks and the ground updates mean that GPS derived time is probably the most accurate time available to a standard embedded application program.

2. To resolve a position laterally and/or vertically. To know where an object is within a few meters on or above the surface of the earth is useful in a wide variety of tracking applications.

3. To navigate to a location. The position information, when combined with current course and speed, allows an application to control and recognize its relative position and distance to any other location within the sphere of the GPS constellation.

To this end, WinSystems provides a sample NMEA 0183 decoding function in **C** called *parse_nmea()* in the file GPS_FUNC.C This function is called with a string argument holding the string to decode, and then decodes the NMEA sentence type and loads the appropriate global variables with the results from that data sentence. Some variables are affected by more than one sentence type, in which case the variable(s) holds the result from the most recently parsed NMEA sentence. The following list gives all of the global variables, their data type, and the NMEA sentence type that will affect their value(s).

An application needs to simply extract the serial data into [CR][LF] terminated strings, call the *parse_nmea()* function, and then utilize the resultant variables as needed.

Variable Type/Name	Encoding	NMEA Sentences affecting
float gps_utc_time	hhmmss.ss	ZDA, GGA, GLL, RMC
int gps_day	(1-31)	ZDA
int gps_month	(1-12)	ZDA
int gps_year	yyyy	ZDA
float gps_latitude	llll.lll	GGA, GLL, RMC
char gps_lat_reference	*N* or *S*	GGA, GLL, RMC
float gps_longitude	yyyyy.yyy	GGA, GLL, RMC
char gps_long_reference	*E* or *W*	GGA, GLL, RMC
int gps_quality	(0-2)	GGA
int gps_satellite_count	nn	GGA
float gps_hdop	v.v	GGA, GSA
float gps_altitude	v.v	GGA
char gps_altitude_unit	*M*	GGA
float gps_separation	v.v	GGA
char gps_separation_unit	*M*	GGA
float gps_differential_age	v.v	GGA
int gps_differential_station_id	(000-1023)	GGA
char gps_status	*A* or *V*	GLL
char gps_mode_indicator	*ADEMSN*	GLL
char gps_op_mode	*A* or *M*	GSA
char gps_fix_mode	(1-3)	GSA
int gps_satellites_in_use[12]	xx	GSA
float gps_pdop	v.v	GSA
float gps_vdop	v.v	GSA
int gps_gsv_message_count	x	GSV

Variable Type/Name	Encoding	NMEA Sentences affecting
int gps_gsv_message_number	x	GSV
int gps_total_sats_in_view	xx	GSV
int gps_prn_number[3][4]	xx	GSV
int gps_elevation[3][4]	(0-90)	GSV
int gps_azimuth[3][4]	(000-359)	GSV
int gps_snr[3][4]	xx	GSV
char gps_rmc_status	*A* or *V*	RMC
float gps_sog	x.x	RMC, VTG
float gps_track	x.x	RMC, VTG
long gps_rmc_date	ddmmyy	RMC
float gps_mag_variation	x.x	RMC
char gps_variation_direction	*E* or *W*	RMC
char gps_sys_mode_indicator	*ADEMSN*	RMC
float gps_track_magnetic	x.x	VTG
float gps_sog_kilometers	x.x	VTG

In addition to the NMEA global variables there are several globals defined to support the serial port functions. These globals are listed here. Refer to the source code for the sample application NMEA3.C for example usage of the serial port support functions.

Parameter	Description
Unsigned port_address	The Base address of the UART
Unsigned int_number	The IRQ number (0-15)
Unsigned baud_rate	The Baud Rate (150-115200)
Unsigned vector	Vector value (internal use only)
Unsigned mask_val	PIC Mask (internal use only)
Long old_vector	Original Vector contents (internal use only)
Unsigned pic_base	Interrupt controller address (internal use only)

There are also the function prototypes for the functions contained in GPS_FUNC.C

Function Prototype
int parse_nmea(char * string)
int com_puts(char * string)
void com_gets(char *string)
int com_read(void)
void com_flush(void)
void com_close(void)
void com_init(void)
unsigned char com_getch(void)
void com_putch(char c)
char *my_token(char *source, char token)

All of these items are declared in the header file GPS_FUNC.H, which may be included by an application desiring to utilize these routines as supplied. Refer to the source code for GPS_FUNC.C or NMEA3.C for clarification of any usage details.

An actual MS-DOS sample application program is provided both in **C** source form and in a precompiled .EXE file. The program is invoked at the DOS command line with optional arguments such as:

nmea3 300 5 4800

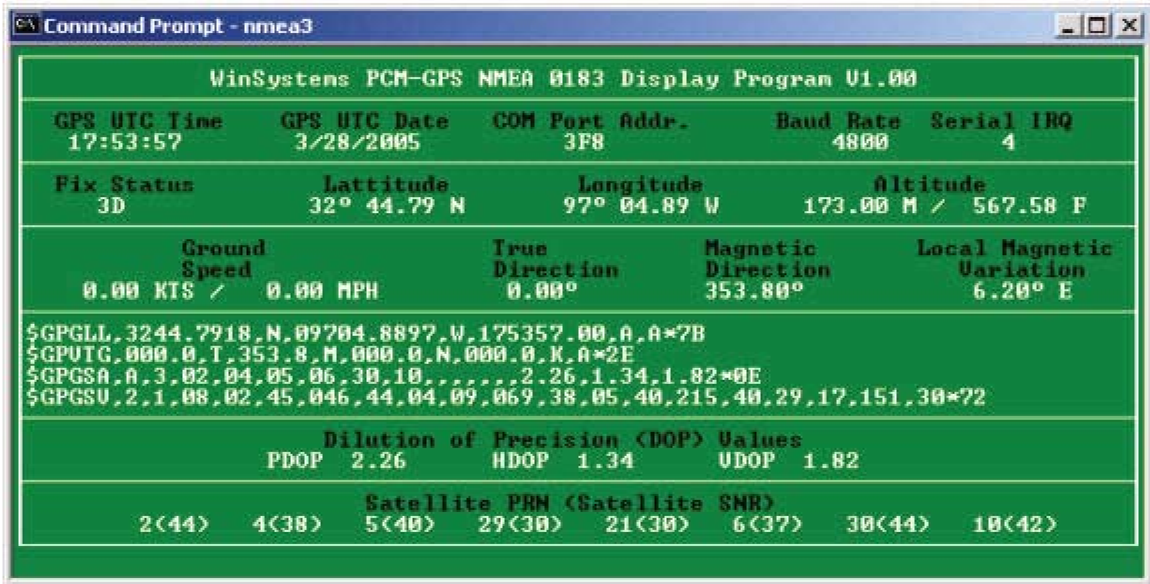
Where 300 is the hex address of the COM port configuration for the GPS, 5 is the IRQ level configuration for the GPS and the 4800 specifies the desired baud rate. If no arguments are given, the program defaults to COM1 values (i.e., 3F8 4 4800).

The application was built and tested using the Borland C/C++ compiler Version 3.1 with a build command line of :

bcc nmea3.c gps_func.c

NMEA3.EXE illustrates the usage of the global variables for display purposes and also serves as a test/diagnostic utility to show the current state of the GPS receiver and its satellite reception.

The following screen examples shows the program running.



The middle of the screen shows the raw NMEA 0183 sentences while the rest of the screen displays a number of the GPS global variables.

C Source Code Listings

```
/* GPS_FUNC.C Copyright 2005, WinSystems Inc. All Rights reserved */
/***************************************************************
*
*   Name    : GPS_FUNC.C
*
*   Project : PCM-GPS
*
*   Purpose : Sample NMEA 0183 Decoding Routines
*
*   Revision: 1.00
*
*   Date    : February 8, 2005
*
*   Author        : Steve Mottin
*
***************************************************************
*
*        Changes :
*
*   Revision        Date          Description
*   --------        ----------    ------------------------------
*   1.00            28/08/2005    Created
*
****************************************************************
*/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#define BUFFER_SIZE 32768

/* GPS Global variables */

/* These are derived from the ZDA message */

float gps_utc_time;
int   gps_day;
int   gps_month;
int   gps_year;

/* These are added by the GGA  message */

float gps_latitude;
char  gps_lat_reference;
float gps_longitude;
char  gps_long_reference;
int   gps_quality;
int   gps_satellite_count;
float gps_hdop;
float gps_altitude;
char  gps_altitude_unit;
float gps_separation;
char  gps_separation_unit;
float gps_differential_age;
int   gps_differential_station_id;

/* These are added by the GLL message */
```

```
char gps_status;
char gps_mode_indicator;

/* These are added by the GSA message */

char gps_op_mode;
char gps_fix_mode;
int  gps_satellites_in_use[12];
float gps_pdop;
float gps_vdop;

/* These are added by the GSV message */

int gps_gsv_message_count;
int gps_gsv_message_number;
int gps_total_sats_in_view;

int gps_prn_number[3][4];
int gps_elevation[3][4];
int gps_azimuth[3][4];
int gps_snr[3][4];

/* These are added by the RMC message */

char gps_rmc_status;
float gps_sog;
float gps_track;
long gps_rmc_date;
float gps_mag_variation;
char gps_variation_direction;
char gps_sys_mode_indicator;

/* These are added by the VTG message */

float gps_track_magnetic;
float gps_sog_kilometers;

/* Com Port Global values */

unsigned port_address;
unsigned int_number;
unsigned baud_rate;
unsigned vector;
unsigned mask_val;
long old_vector;
unsigned pic_base = 0x20;

/* Local function proto-types */

int parse_nmea(char *string);

int com_puts(char *str);
void com_gets(char *str);
int com_read(void);
void com_flush(void);
void com_close(void);
void com_init(void);
int com_check(void);
unsigned char com_getch(void);
void com_putch(char c);
char *my_token(char *source,char token);
```

```
char *field[50];


/* This function accepts a string believed to contain standard NMEA 0183 sentence
  data and parses those fields and loads the appropriate global variables with the results.
*/

int parse_nmea(char *string)
{
int field_count;
int x,y;

   field_count = 0;

#ifdef DEBUG
   printf("Parsing NMEA string : <%s>\n",string);
#endif

   /* NMEA 0183 fields are delimited by commas. The my_token function returns
   pointers to the fields.
   */

   /* Get the first field pointer */

   field[0] = my_token(string,',');
#ifdef DEBUG
   if(field[0])
      printf("Token = <%s>\n",field[0]);
#endif

   field_count++;
   while(1)
   {
      /* Contiue retrieving fields until there are no more (NULL) */

      field[field_count] = my_token(NULL,',');
      if(field[field_count] == NULL)
         break;
#ifdef DEBUG
      printf("Token = <%s>\n",field[field_count]);
#endif

      field_count++;
   }

#ifdef DEBUG
   printf("%d fields parsed\n",field_count);
#endif

   /* If we got at least ONE field */

   if(field_count)
   {
      /* Check the first field for the valid NMEA 0183 headers */

      if(strcmp(field[0],"$GPGGA") == 0)
      {
         /* Retrieve the values from the remaining fields */

         gps_utc_time = atof(field[1]);
         gps_latitude = atof(field[2]);
         gps_lat_reference = *(field[3]);
         gps_longitude = atof(field[4]);
```

```
          gps_long_reference = *(field[5]);
          gps_quality = atoi(field[6]);
          gps_satellite_count = atoi(field[7]);
          gps_hdop = atof(field[8]);
          gps_altitude = atof(field[9]);
          gps_altitude_unit = *(field[10]);
          gps_separation = atof(field[11]);
          gps_separation_unit = *(field[12]);
          gps_differential_age = atof(field[13]);
          gps_differential_station_id = atoi(field[14]);
#ifdef DEBUG
          printf("NMEA string GPGGA recognized\n");
          printf("Time = %9.2f\n",gps_utc_time);
          printf("Position : %8.3f %c %8.3f %c\n",gps_latitude,gps_lat_reference,
             gps_longitude,gps_long_reference);

          printf("GPS quality = %d, Satelite count = %d, HDOP = %4.2f\n",gps_quality,
             gps_satellite_count, gps_hdop);

          printf("GPS altitude = %9.2f %c, Geoidal Separation = %9.2f %c\n",gps_altitude,
             gps_altitude_unit, gps_separation, gps_separation_unit);

          printf("GPS differential update age = %9.2f.Station ID = %d\n",gps_differential_age,
             gps_differential_station_id);

#endif

       }

       if(strcmp(field[0],"$GPGLL") == 0)
       {
          /* Retrieve the values from the remaining fields */

          gps_latitude = atof(field[1]);
          gps_lat_reference = *(field[2]);
          gps_longitude = atof(field[3]);
          gps_long_reference = *(field[4]);
          gps_utc_time = atof(field[5]);
          gps_status = *(field[6]);
          gps_mode_indicator = *(field[7]);

#ifdef DEBUG
          printf("NMEA string GPGLL recognized\n");
          printf("Position : %8.3f %c %8.3f %c\n",gps_latitude,gps_lat_reference,
             gps_longitude,gps_long_reference);
          printf("Time = %9.2f\n",gps_utc_time);

          printf("GPS status = %c, GPS mode indicator = %c\n",gps_status,gps_mode_indicator);
#endif

       }



       if(strcmp(field[0],"$GPGSA") == 0)
       {
          /* Retrieve the values from the remaining fields */

          gps_op_mode = *(field[1]);
          gps_fix_mode = *(field[2]);
          gps_pdop = atof(field[15]);
          gps_hdop = atof(field[16]);
          gps_vdop = atof(field[17]);
```

```
#ifdef DEBUG
        printf("NMEA string GPGSA recognized\n");

        printf("Operation mode = %c, Fix mode = %c\n",gps_op_mode,gps_fix_mode);

        printf("Satelites in use : ");
#endif
        for(x=0; x<12; x++)
        {
          gps_satellites_in_use[x] = atoi(field[x+3]);
#ifdef DEBUG
          if(gps_satellites_in_use[x])
            printf("%d   ",gps_satellites_in_use[x]);
#endif
        }
#ifdef DEBUG
        printf("\n");

        printf("GPS precision %5.2f PDOP, %5.2f HDOP, %5.2f VDOP\n",
          gps_pdop,gps_hdop,gps_vdop);
#endif
      }

      if(strcmp(field[0],"$GPGSV") == 0)
      {

        /* Retrieve the data from the remaining fields */

        gps_gsv_message_count = atoi(field[1]);
        gps_gsv_message_number = atoi(field[2]);

        gps_total_sats_in_view = atoi(field[3]);

#ifdef DEBUG
        printf("NMEA string GPGSV recognized\n");
        printf("Total satelites in view = %d\n",gps_total_sats_in_view);
#endif
        if((gps_gsv_message_number > 0) && (gps_gsv_message_number < 4))
        {
          y = gps_gsv_message_number - 1;
          for(x=0; x< 4; x++)
          {
            gps_prn_number[y][x] = atoi(field[(x*4)+4]);
            gps_elevation[y][x] = atoi(field[(x*4)+5]);
            gps_azimuth[y][x] = atoi(field[(x*4)+6]);
            gps_snr[y][x] = atoi(field[(x*4)+7]);
#ifdef DEBUG
            printf("Satelite %d - Elev = %d Azim = %d  SNR = %d\n",
              gps_prn_number[y][x],gps_elevation[y][x],gps_azimuth[y][x],
              gps_snr[y][x]);
#endif

          }
        }

      }

      if(strcmp(field[0],"$GPRMC") == 0)
      {

        /* Retrieve the data from the remaining fields */
```

```
        gps_utc_time = atof(field[1]);
        gps_rmc_status = *(field[2]);
        gps_latitude = atof(field[3]);
        gps_lat_reference = *(field[4]);
        gps_longitude = atof(field[5]);
        gps_long_reference = *(field[6]);
        gps_sog = atof(field[7]);
        gps_track = atof(field[8]);
        gps_rmc_date = atol(field[9]);
        gps_mag_variation = atof(field[10]);
        gps_variation_direction = *(field[11]);
        gps_sys_mode_indicator = *(field[12]);

#ifdef DEBUG
        printf("NMEA string GPRMC recognized\n");

        printf("GPS UTC Time = %9.2f. RMC Status = %c\n",gps_utc_time,gps_rmc_status);


        printf("Position : %7.2f Deg %c  %7.2f Deg %c\n",gps_latitude,gps_lat_reference,
          gps_longitude,gps_long_reference);


        printf("GPS Track %7.2f degrees at %7.2f Knot groundspeed\n",gps_track,gps_sog);


        printf("GPS Date : %8ld Mag Variation %6.2f Deg %c GPS Mode %c\n",
          gps_rmc_date,gps_mag_variation,gps_variation_direction,gps_mode_indicator);
#endif
      }

      if(strcmp(field[0],"$GPVTG") == 0)
      {
        /* Retrieve the data from the remaining fields */

        gps_track = atof(field[1]);
        gps_track_magnetic = atof(field[3]);
        gps_sog = atof(field[5]);
        gps_sog_kilometers = atof(field[7]);
        gps_sys_mode_indicator = *(field[9]);

#ifdef DEBUG
        printf("NMEA string GPVTG recognized\n");


        printf("GPS Track : %7.2f True %7.2f Magnetic. Speed :  %9.2f Knots %9.2f Kilometer/Hour\n",
          gps_track,gps_track_magnetic,gps_sog,gps_sog_kilometers);
#endif
      }

      if(strcmp(field[0],"$GPZDA") == 0)
      {

        /* Retrieve the data from the remaining fields */

        gps_utc_time = atof(field[1]);
        gps_day = atoi(field[2]);
        gps_month = atoi(field[3]);
        gps_year = atoi(field[4]);
#ifdef DEBUG
        printf("NMEA string GPZDA recognized\n");
        printf("%9.2f %2d/%2d/%4d\n",gps_utc_time,gps_month,gps_day,gps_year);
#endif
```

```
      }

    }
    return field_count;
}


/* These variables and the function my_token are used to retrieve the comma
   delimited field pointers from the input string. Repeated calls to
   my_token return the next field until there are no more (NULL).
*/

static char stat_string[128];
char *current = NULL;

char *my_token(char *source,char token)
{
char *start;

    /* The source string is real only for the first call. Subsequent calls
    are made with the source string pointer as NULL
    */

    if(source != NULL)
    {
        /* If the string is empty return NULL */
        if(strlen(source) == 0)
            return NULL;
        strcpy(stat_string,source);
        /* Current is our 'current' position within the string */
        current = stat_string;
    }

    start = current;

    while(1)
    {
        /* If we're at the end of the string, return NULL */

        if((*current == '\0') && (current == start))
            return NULL;

        /* If we're at the end now, but weren't when we started, we need
           to return the pointer for the last field before the end of string
        */

        if(*current == '\0')
            return start;

        /* If we've located our specified token (comma) in the string
            load its location in the copy with an end of string marker
            so that it can be handled correctly by the calling program.
        */

        if(*current == token)
        {
            *current = '\0';
            current++;
            return start;
        }
        else
        {
            current++;
```

```
        }
    }
}


break_handler()
{
        return 1;
}


/* Assign buffer and pointers for port */

char com_buffer[BUFFER_SIZE];
unsigned com_head,com_tail;

/* Initialize COM for specified baud rate. Remaining communications
   parameters are hard-wired to 8-bit word, no parity, and 1 stop bit.
*/
void com_init(void)
{
long far *addr;
void interrupt far com_isr();
unsigned baud;

        /* calculate the necessary baud rate divisor given the desired baud
           rate and the input frequency to the counter timer
        */

        baud = (unsigned) (1843200L / ((long)baud_rate * 16L));

        /* Set up the baud rate generation */

        /* Set the DLAB bit to allow loading of the divisor values */

        outportb(port_address+3,inportb(port_address + 3) | 0x80);
        outportb(port_address+1, baud >> 8);
        outportb(port_address, baud & 0xff);
        outportb(port_address+3,inportb(port_address + 3) & 0x7f);

        /* Install the interrupt service routine for com input */

        if(int_number < 8)
                vector = int_number + 8;
        else
                vector = (int_number - 8) + 0x70;

        addr = (long far *) (vector * 4L);
        disable();

        /* Save the old vector for later restoration */

        old_vector = *addr;

        *addr = (long) com_isr;

        /* Initialize the receive buffer pointers */

        com_head = com_tail = 0;

        /* Initialize the UART for 8 bit word, no parity and one stop bit */

    outportb(port_address+3,3); /* 8 bits, 1 stop, no parity */
```

```
        outportb(port_address+1,1);    /* Enable, RX interrupts only */
        outportb(port_address+4,0x0b); /* Set,RTS,DTR and OUT2 */

        inportb(port_address);
        inportb(port_address);
        inportb(port_address);


        /* Unmask the interrupts */

        if(int_number < 8)
        {
                mask_val = 1 << int_number;
                outportb(pic_base + 1,inportb(pic_base + 1) & ~mask_val);
        }
        else
        {
                mask_val = 1 << (int_number - 8);
                outportb(0xa1,inportb(0xa1) & ~mask_val);
                outportb(pic_base +1,inportb(pic_base + 1) & 0xfb);
        }
        enable();
}

/* This is the comm_close routine to shut down the com port before
   exiting thus not leaving the interrupts hanging
*/

void com_close(void)
{
long far *addr;

        addr = (long far *) (vector * 4L);
        disable();
        *addr = old_vector;
        outportb(pic_base + 1,inportb(pic_base + 1) | mask_val);
        enable();
}

/* This is the COM receive character ISR. It places incoming characters
   into the receive buffer.
*/
void interrupt far com_isr()
{
        /* Get the character, store it in the buffer, update the buffer pointer */

        com_buffer[com_tail++] = inportb(port_address);

        /* If tail pointer points to end of buffer, reset it to the beginning */

        if(com_tail == BUFFER_SIZE)
                com_tail = 0;

        /* Issue non-specific EOI to interrupt controller */

        if(int_number > 8)
                outportb(0xa0,0x20);

        outportb(pic_base, 0x20);       /* Send non-specific EOI to PIC */
}

/* This function writes a string to COM utilizing the com_putch() function
   for outputting each character.
```

```
*/
int com_puts(char *str)
{
        while(*str)
                com_putch(*str++);
        return 0;
}

/* This function writes the specified character to the COM UART. This is
   a polled mode function and will wait until the UART is ready for the
   next character.
*/
void com_putch(char c)
{
unsigned retry;

        /* Wait till UART ready */
    retry = 0x8000;

        while((inportb(port_address+5) & 0x40) == 0)
    {
       if(--retry == 0)
          break;
    }

        /* Write the character */
        outportb(port_address,c);

}

int com_check(void)
{
        if(com_head == com_tail)
                return 0;
        else
                return 1;
}

unsigned char com_getch(void)
{
int c;

        while(!com_check())
                ;
        c = com_buffer[com_head++];
        if(com_head == BUFFER_SIZE)
                com_head = 0;
        return(c & 0xff);
}

void com_gets(char *str)
{
int c;

        while((c = com_getch()) != '\r')
        {
                com_putch(c);
                *str++ = c;
        }
        *str = '\0';
}

int com_read()
```

```
{
        if(inportb(port_address+5) & 1)
                return(inportb(port_address) & 0xff);
        else
                return -1;
}

void com_flush()
{
        disable();
        com_head = com_tail = 0;
        enable();
}
```

```
/* GPS_FUNC.H Copyright 2005, WinSystems Inc. All Rights reserved */
/***************************************************************
*
*   Name    : GPS_FUNC.H
*
*   Project : PCM-GPS
*
*   Purpose : Sample NMEA Decoding Routines
*
*   Revision: 1.00
*
*   Date    : February 8, 2005
*
*   Author  : Steve Mottin
*
***************************************************************
*
*        Changes :
*
*   Revision        Date          Description
*   --------        ----------    ------------------------------
*   1.00            28/08/2005    Created
*
***************************************************************
*/

extern unsigned port_address;
extern unsigned int_number;
extern unsigned baud_rate;
extern unsigned vector;
extern unsigned mask_val;
extern long old_vector;
extern unsigned pic_base;


/* GPS Global variables */

/* These are from the ZDA message */

extern float gps_utc_time;
extern int   gps_day;
extern int   gps_month;
extern int   gps_year;

/* These are added by the GGA  message */

extern float gps_latitude;
extern char  gps_lat_reference;
extern float gps_longitude;
extern char  gps_long_reference;
extern int   gps_quality;
extern int   gps_satellite_count;
extern float gps_hdop;
extern float gps_altitude;
extern char  gps_altitude_unit;
extern float gps_separation;
extern char  gps_separation_unit;
extern float gps_differential_age;
extern int   gps_differential_station_id;

/* These are added by the GLL message */

extern char gps_status;
```

```
extern char gps_mode_indicator;

/* These are added by the GSA message */

extern char gps_op_mode;
extern char gps_fix_mode;
extern int  gps_satellites_in_use[12];
extern float gps_pdop;
extern float gps_vdop;

/* These are added by the GSV message */

extern int gps_gsv_message_count;
extern int gps_gsv_message_number;
extern int gps_total_sats_in_view;

extern int gps_prn_number[3][4];
extern int gps_elevation[3][4];
extern int gps_azimuth[3][4];
extern int gps_snr[3][4];

/* These are added by the RMC message */

extern char gps_rmc_status;
extern float gps_sog;
extern float gps_track;
extern long gps_rmc_date;
extern float gps_mag_variation;
extern char gps_variation_direction;
extern char gps_sys_mode_indicator;

/* These are added by the VTG message */

extern float gps_track_magnetic;
extern float gps_sog_kilometers;

int parse_nmea(char *string);
int com_puts(char *str);
void com_gets(char *str);
int com_read(void);
void com_flush(void);
void com_close(void);
void com_init(void);
int com_check(void);
unsigned char com_getch(void);
void com_putch(char c);
char *my_token(char *source,char token);
```

```
/* NMEA3.C Copyright 2005, WinSystems Inc. All Rights reserved */
/**************************************************************
*
*   Name    : NMEA3.C
*
*   Project : PCM-GPS
*
*   Purpose : Sample NMEA Decoding Routines
*
*   Revision: 1.00
*
*   Date    : February 8, 2005
*
*   Author        : Steve Mottin
*
**************************************************************
*
*        Changes :
*
*   Revision        Date          Description
*   --------        ----------    ------------------------------
*   1.00            28/08/2005    Created
*
***************************************************************
*/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#include „gps_func.h"

#define VERSION „1.00"

/* Local function proto-types */

void update_screen(void);
void screen_init(void);
void close_screen(void);
void draw_box(void);
void center_text(int line, char *text);
void display_raw(char *string);
void put_blanks(int count);

/* Local global variables for handling strings */

char line_buffer[128];
char temp_string[80];

/* Command line arguments specify the com port I/O address, the com port IRQ
   number and the com port baud rate respectively as in :

        nmea3 3f8 4 4800

   which specifies a com port at 3f8 (hex) IRQ 4 (decimal) and 4800 baud (decimal).
   When no arguments are given the default is 3f8 4 4800.
*/

main(int argc, char *argv[])
{
```

```
unsigned char c;
int line_index;
int count;

   /* First priority is to initialize the com port. If there are
      command line arguments, they must be recognized.
   */


         port_address = 0x3f8;
         int_number = 4;
   baud_rate = 4800;

         if(argc > 1)
         {
                 sscanf(argv[1],"%x",&port_address);
                 if(port_address < 0x100 || port_address > 0x3f8)
                 {
                         printf("\nInvalid port address %04x specified\n",port_address);
                         exit(2);
                 }
                 if(port_address == 0x2f8)
                         int_number = 3;
                 if(port_address == 0x3f8)
                         int_number = 4;
         }

         if(argc > 2)
         {
                 sscanf(argv[2],"%d",&int_number);
                 if(int_number < 2 || int_number > 15)
                 {
                         printf("\nInvalid interrupt number %d specified\n");
                         exit(2);
                 }
         }

         if(argc > 3)
         {
                 sscanf(argv[3],"%d",&baud_rate);
                 if(baud_rate < 110 || baud_rate > 38400)
                 {
                         printf("\nInvalid baud rate %d specified\n",baud_rate);
                         exit(2);
                 }
         }

   /* Now that all of the com port values are known we can call com_init();
      which will initialize the port.
   */

         com_init();

   /* Clear the screen and display the main screen */

   screen_init();


   /* Get ready to start handling the incoming characters from the NMEA
      stream. line_index is the current string position for the incoming
      characters. Since this is the beginning, that's where it starts.
   */
```

```
line_index = 0;

/* This application does nothing but parse NMEA strings and display
   the provided data. It loops eternal until a Ctrl-A keypress occurs.
*/

     while(1)
     {
   /* Check for any characters in the serial receive buffer */
             if(com_check())
             {
      /* Yes, retrieve the character */
                     c = com_getch();

      /* We ignore carriage return characters */
      if(c == '\r')
         continue;

      /* If we have a line feed and have some characters stored
         we will parse the line to see what we have.
      */
      if((c == '\n') && (line_index > 0))
      {
         line_buffer[line_index] = 0;

         /* This displays the raw NMEA data in a window in the
            middle of the screen display
         */

         display_raw(line_buffer);

         /* Call the parse_name function in gps_func.c
            It decodes the string and loads decoded data values
            into the appropriate global variables for use by
            the application.
         */

         parse_nmea(line_buffer);

         /* All of our global variable values are now displayed
            on the screen.
         */

         update_screen();

         /* Start the counter over for the next string */

         line_index = 0;
         continue;
      }

      /* As long as we haven't overflowed the buffer. Put the character
         into the line_buffer for later parsing. Update the index.
      */

      if(line_index < 80)
         line_buffer[line_index++] = c;

             }
   /* Check for a keyboard character */
             if(kbhit())
             {
                     c = getch();
```

```
        /* If the character is a Ctr-A shut down the com-port,
           clear the screen, and exit.
        */
                        if(c == 0x01)
                        {
                                com_close();
           close_screen();
                                exit(0);
                        }
        /* All other keystrokes are ignored */
                }
        }

}

/* This function displays the RAW NMEA 0183 strings, or whatever comes in over the
   comm port, in a small window in the center of the screen.
*/

void display_raw(char *string)
{
   window(2,14,79,17);
   gotoxy(1,4);
   cprintf("\n");
   gotoxy(1,4);
   cprintf("%s",string);
   window(1,1,80,25);
}

/* This function sets up the screen and displays the data block titles */

void screen_init()
{
   _setcursortype(_NOCURSOR);
   window(1,1,80,25);
   textbackground(GREEN);
   textcolor(WHITE);
   clrscr();
   draw_box();
   sprintf(temp_string,"WinSystems PCM-GPS NMEA 0183 Display Program V%s",VERSION);
   center_text(2,temp_string);

   textcolor(BLACK);
   gotoxy(4,4);
   cputs("GPS UTC Time");
   gotoxy(20,4);
   cputs("GPS UTC Date");
   gotoxy(35,4);
   cputs("COM Port Addr.");
   gotoxy(55,4);
   cputs("Baud Rate");
   gotoxy(66,4);
   cputs("Serial IRQ");

   gotoxy(4,7);
   cputs("Fix Status");
   gotoxy(23,7);
   cputs("Lattitude");
   gotoxy(41,7);
   cputs("Longitude");
   gotoxy(62,7);
   cputs("Altitude");
```

```
    gotoxy(13,10);
    cputs("Ground");
    gotoxy(13,11);
    cputs("Speed");

    gotoxy(35,10);
    cputs("True");
    gotoxy(35,11);
    cputs("Direction");

    gotoxy(50,10);
    cputs("Magnetic");
    gotoxy(50,11);
    cputs("Direction");

    gotoxy(65,10);
    cputs("Local Magnetic");
    gotoxy(68,11);
    cputs("Variation");


    center_text(19,"Dilution of Precision (DOP) Values");

    center_text(22,"Satellite PRN (Satellite SNR)");
    textcolor(WHITE);
    gotoxy(39,5);
    cprintf("%4X",port_address);
    gotoxy(58,5);
    cprintf("%5d",baud_rate);
    gotoxy(70,5);
    cprintf("%2d",int_number);

}


/* This function sets the screen back to normal before exiting */

void close_screen()
{

    _setcursortype(_NORMALCURSOR);
    window(1,1,80,25);
    textbackground(BLACK);
    textcolor(WHITE);
    clrscr();
}

/* This function uses the line characters to draw the screen box and
   the data block areas.
*/

void draw_box(void)
{
int x;

    gotoxy(1,1);
    putch('Ú');
    for(x=1; x<79; x++)
        putch('Ä');

    putch('¿');

    for(x=2; x<24; x++)
```

```
{
   gotoxy(1,x);
   putch(‘³’);
   gotoxy(80,x);
   putch(‘³’);
}

gotoxy(1,24);
putch(‘À’);
for(x=1; x<79; x++)
   putch(‘Ä’);
putch(‘Ù’);

gotoxy(1,3);
putch(‘Ã’);

for(x=1; x<79; x++)
   putch(‘Ä’);

putch(‘´’);

gotoxy(1,6);
putch(‘Ã’);

for(x=1; x<79; x++)
   putch(‘Ä’);

putch(‘´’);

gotoxy(1,9);
putch(‘Ã’);

for(x=1; x<79; x++)
   putch(‘Ä’);

putch(‘´’);



gotoxy(1,13);
putch(‘Ã’);

for(x=1; x<79; x++)
   putch(‘Ä’);

putch(‘´’);




gotoxy(1,18);
putch(‘Ã’);

for(x=1; x<79; x++)
   putch(‘Ä’);

putch(‘´’);

gotoxy(1,21);
putch(‘Ã’);

for(x=1; x<79; x++)
   putch(‘Ä’);
```

```
    putch(' ');


}

/* This helper function centers a text string on the desired line */

void center_text(int line, char *text)
{
int column;

    column = 40 - (strlen(text) / 2);
    gotoxy(column,line);
    cputs(text);
}

/* This function is used to blank out a specific number of character
   positions
*/

void put_blanks(int count)
{
int x;

    if((count <= 0) || (count > 78))
        return;

    for(x=0; x<count; x++)
        temp_string[x] = ' ';

    temp_string[x] = '\0';
    cputs(temp_string);
}

/* This function is the workhorse of this application. It takes the globale
   variables that have been loaded by call to parse_nmea and displays them
   in readable form across the display screen.
*/


void update_screen()
{
int x,y;
int hour,minute,second;
unsigned long temp;
float ftemp;
double bearing;
double distance;
float lat_mul,lon_mul;

    gotoxy(5,5);        // Middle of top line

    /* Display the current UTC time from the GPS */

    if(gps_utc_time)
    {
        temp = (unsigned long)gps_utc_time;
        hour = (int)(temp /10000l);
        temp = temp - (hour * 10000l);
        minute = (int)(temp / 100);
        temp = temp - (minute * 100);
```

```
    second = (int) temp;
    cprintf("%2d:%02d:%02d ",hour,minute,second);
}

gotoxy(20,5);

/* Display the current GPS date information */

cprintf("%2d/%02d/%04d",gps_month,gps_day,gps_year);

gotoxy(7,8);
put_blanks(69);
gotoxy(7,8);

/* Display the current state of our navigational fix */

if(gps_fix_mode == '1')
{
    /* We don't have a location yet */
    cprintf("N/A");
}
else
{
    /* We have at least a 2-dimensional location fix, we can display
       basic position information
    */

    if(gps_fix_mode == '2')
        cprintf("2D");
    if(gps_fix_mode == '3')
        cprintf("3D");

    gotoxy(21,8);

    /* Display latitude of current position. Adjust for readability */

    x = (int) (gps_latitude / 100.0);
    ftemp = (gps_latitude - (x * 100.0));
    cprintf("%3dø %05.2f %c",x,ftemp,gps_lat_reference);

    gotoxy(39,8);

    /* Display longitude of current position. Also adjust to be readable */

    x = (int) (gps_longitude / 100.0);
    ftemp = (gps_longitude - (x * 100.0));
    cprintf("%3dø %05.2f %c",x,ftemp,gps_long_reference);

    gotoxy(55,8);
    if(gps_fix_mode == '3')
    {
        /* If we have a 3-dimensional fix, we can also display GPS
           altitude in both meters and feet.
        */

        cprintf("%8.2f %c",gps_altitude,gps_altitude_unit);

        cprintf(" /%8.2f F",gps_altitude * 3.28083);

    }

}
gotoxy(10,23);
```

```
    put_blanks(69);
    gotoxy(10,23);

    /* This code sequence displays the currently used satelite numbers (PRN)
       and their corresponding signal to noise ratio (SNR).
    */

    for(y=0; y<2; y++)
    {
       for(x=0; x<4; x++)
       {
          if(gps_prn_number[y][x])
             printf("%d(%d)   ",gps_prn_number[y][x],gps_snr[y][x]);

       }
    }

    gotoxy(3,12);

    /* Display the current speed over the ground (SOG) in both knots and
       Mile per hour.
    */

    cprintf("%7.2f KTS /%7.2f MPH",gps_sog,gps_sog * 1.150779);

    /* Display both the True track direction as well as the magnetic
       track direction.
    */

    gotoxy(35,12);
    cprintf("%5.2fø",gps_track);

    gotoxy(50,12);
    cprintf("%5.2fø",gps_track_magnetic);

    /* Display the local magnetic variation. The deviation from true
       north.
    */

    gotoxy(68,12);
    cprintf("%5.2fø %c",gps_mag_variation,gps_variation_direction);

    /* Display the Dilution of Precision values */

    sprintf(temp_string,"PDOP %5.2f      HDOP %5.2f      VDOP %5.2f",gps_pdop,gps_hdop,gps_vdop);
    center_text(20,temp_string);
}
```