Date: 051031
Rev: 1.0

GPS Library for CCP XS

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the CoDeSys GPS library which enables the user to receive GPS data in a CoDeSys PLC program.

1.2 References

http://www.nmea.org/

1.3 Revision history

Rev	Date	Author	Comments
1.0	2005-10-30	Fredrik Löwenhielm, CCS	Initial version

2 Background and principles

The PLC library described in this document enables the user to read GPS data in a CoDeSys PLC program if a CCP XS is used as target hardware. The CCP XS is equipped with a GPS module that communicates by the NMEA standard. Therefore, the GPS data/variables supplied in the GPS library are very NMEA oriented. Knowledge about the NMEA standard will help understanding the GPS data received. Information about the NMEA standard can be found and downloaded on the NMEA web site http://www.nmea.org/.

The library will give the user one function to initialise the library and a global variable to read GPS data from. The GPS data is updated every time a NMEA message is received from the GPS unit in CCP XS. The NMEA messages are sent once every second.

3 Installation

In order to work, the GPS library needs to have proper GPS drivers installed on the target hardware. The GPS driver dll used by the target runtime is called GPS.dll. This file has to be located in the same directory as the runtime executable. In addition, the GPS dll has to be registered in order to be loaded by the runtime. This is done by modifying the runtime configuration file rts3s.cfg also located in the runtime directory. Below are instructions for installing the GPS driver.

1. Locate your runtime directory on the target file system. The path is usually \SoftPLC\. If there is no such directory, search for the file PLCCEARM.exe. The directory where you find the file is your runtime directory.

2. Copy the GPS.dll file to the runtime directory. If the file is already there, your GPS driver is already installed and you can skip the rest of this section.

3. Open the rts3s.cfg file in a text editor. It should look something like this:

```
[IODRIVERLIST]
MODULE0=CCPXS
INITFCT0=IODrvInit
MODULE1=3SCanFrame
INITFCT1=IODrvInit
[PLC]
SETTINGS_FROM_CFG=Yes
USE SYSINTR_TIMING AS TICK=Yes
Files=
BootMode=Run
```

Now, insert these two lines in the IO driver list:

```
MODULE2=GPS
INITFCT2=IODrvInit
```

If there are more than two drivers present in the IO driver list, just increase the module number. Your file should now look something like this:

```
[IODRIVERLIST]
MODULE0=CCPXS
INITFCT0=IODrvInit
MODULE1=3SCanFrame
INITFCT1=IODrvInit
MODULE2=GPS
INITFCT2=IODrvInit
[PLC]
SETTINGS_FROM_CFG=Yes
USE SYSINTR_TIMING AS TICK=Yes
Files=
BootMode=Run
```

4. Now you can check if your runtime loads the GPS dll. If your runtime is already started, shut it down. Then, start it by double clicking on the file PLCCEARM.exe. Login to your PLC from CoDeSys. Go to the 'resources' tab and choose the PLC Browser. Enter 'rtsinfo' in the command line prompt. In the information you receive from the PLC, the loading of the GPS driver should be present. If not, check that you have followed the steps above correctly.

5. Now, place the library files GPS_XS.lib and GPS_DRV.lib in your CCP XS target library directory in CoDeSys. Your library directory can be located by starting CoDeSys, creating a new project with CCP XS as target. Then, open the Project->Options dialog. Choose 'Directories'. There, you can see the library paths used by CoDeSys.

6. Now you are ready to use the GPS library.

4 Using the GPS library

The following section will describe how you create a project where you use the GPS library.

1. Create a new project in CoDeSys. Choose the proper target (CCP XS).

2. Go the Resources tab. Choose Library manager.

3. Choose Insert in the main menu. Choose the file GPS_XS.lib. The GPS_DRV.lib will automatically be loaded too.

4. Choose Insert again. Choose the file SysLibSem.lib. This library should already be installed with your CCP XS target installation.

5. In your program code call the function GPS_Init(). The function is described further down in this document. GPS_Init() has been added by the GPS_XS library. It should not cause any trouble if the function is called more than one time, but it is recommended that the function is called once in the PLC program. The GPS data will not be valid before this function is called.

6. After the GPS_Init() function has been called the GPS data is readable in the global variable “gpsData” which has been added by the GPS_XS library. The gpsData variable is an instance of the type GPS_DATA. The GPS_DATA type is described in detail in a later chapter. Note that it is recommended that the function SysSemEnter and SysSemLeave is called when reading the gpsData variable. This is described in the section “Using semaphores”.

5 Functions

5.1 GPS_Init

Declaration of the GPS_Init function:

```
FUNCTION GPS_Init : DWORD
VAR_INPUT
END_VAR
VAR
END_VAR
```

The GPS_Init function has no input parameters. If initialisation of the GPS module succeeds it returns a handle to a semaphore which can be used for safe reading of the variable gpsData. If initialisation fails the function will return zero.

6 Using a semaphore

In the runtime, a thread is created that updates the gpsData variable. In order to be sure that the PLC program does not get data that currently is being updated by the thread, a semaphore can be used. If a semaphore is not used it is theoretically possible that some data will contain erroneous information. This should not happen with data of the most common types, like DWORD, WORD, BYTE etc. However, when the GPS thread is updating strings, it is possible that the PLC program reads from the string before it is completely updated. In the section below there are code examples of how to use a semaphore.

7 Code example

This section gives a example of code for using the GPS library. In order to use the code the libraries GPS_XS.lib and the SysLibSem.lib has to be added to the project as described in previous sections.

```
(* Variable declaration*)
VAR
        b_init: BOOL := FALSE;
        hSem: DWORD;
        latitude: DWORD;
        longitude: DWORD;
END_VAR
```

```
(*First loop, call the GPS_Init() function.  Save semaphore handle*)
IF b_init = FALSE THEN
        hSem := GPS_Init();
        b_init := TRUE;
END_IF

(*Read the wanted variables from gpsData, use the semaphore*)
SysSemEnter(hSem);
latitude := gpsData.gps_gga.latitude;
longitude := gpsData.gps_gga.longitude;
SysSemLeave(hSem);  (*Now the gpsData has been read, and we signal this to the semaphore*)
```

In the code example above it is actually not necessary to use the semaphore. The reason for this is that both the "gpsData.gps_gga.latitude" and the "gpsData.gps_gga.longitude" variables are of type DWORD. This means that they will be updated during one single operation by the CPU. However, if the semaphore is not used it is possible that one of the variables is newly updated and the other has an older value.

8 GPS data types

After you have completed the steps in the previous sections you are ready to start using the PLC data in your PLC program. This section describes the data types of the variables contained in the gpsData variable. As mentioned earlier in this document, the data types are very NMEA oriented and therefore it is recommended that the reader have some knowledge about the NMEA standard and the NMEA messages.

8.1 GPS_DATA type

The only variable that is added by the GPS library is gpsData which is of type GPS_DATA.

This is the declaration of GPS_DATA:

```
TYPE GPS_DATA :
STRUCT
                gps_gga:GPS_GGA;
                gps_gsa:GPS_GSA;
                gps_gsv:GPS_GSV;
                gps_rmc:GPS_RMC;
END_STRUCT
END_TYPE
```

The four variables contained in the GPS_DATA type relate to the NMEA messages:

- GGA - Global Positioning System Fix Data
- GSA - DOP and Active Satellites
- GSV – Satellites in view
- RMC – Recommended Minimum Specific GNSS Data

The NMEA messages mentioned above are the messages that the GPS driver receives from the GPS module in the CCP XS. The variables gps_gga, gps_gsa, gps_gsv and gps_rmc are updated every time the corresponding NMEA message is received by the driver. The data is updated approximately once every second. The GSV message actually consists of several messages depending on how many satellites the GPS unit have in its view. Therefore the GSV message counter can tick faster than the other message counters.

8.2 GPS_GGA type

Below is the declaration of the GPS_GGA data type:

```
TYPE GPS_GGA :
STRUCT
                utc_time:STRING(20);
                latitude :DWORD;
                northSouth : STRING(1);
                longitude : DWORD;
                eastWest : STRING(1);
                fixValid : USINT;
                numberOfSatellitesUsed : UINT;
                horizontalDilutionOfPres :REAL;
                altitude : REAL;
                diffWGS84ref : REAL;
                numberOfReceivedMsgs:DWORD;
END_STRUCT
END_TYPE
```

Below is a description of the variables:

utc_time – This variable contains the UTC time as a string in the format “hhmmss.dd” where hh is hours, mm is minutes, ss is seconds, and dd is the decimal part of seconds.

latitude - This is the latitude as a DWORD. The format is xxmmdddd where xx is degrees, mm is minutes, and dddd is the decimal part of minutes. In the NMEA message this value is supplied as a decimal value xxmm.dddd. The value received is multiplied by 10,000 and inserted in the latitude variable.

northSouth – This is the latitude north/south indicator as a string. The possible values are ‘N’ (north) or ‘S’ (south).

longitude – This is the longitude as a DWORD. The format is yyymmdddd where yyy is degrees, mm is minutes, and dddd is the decimal part of minutes. In the NMEA message this value is supplied as a decimal value yyymm.dddd. The value received is multiplied by 10,000 and inserted in the longitude variable.

eastWest – This is the east/west indicator as a string. The possible values are ‘E’ (east) or ‘W’ (west).

fixValid – This variable tells if the fix is valid or not. Possible values are 1 (valid) or 0 (not valid). If the fix is not valid, this means that the GPS receiver does not have enough signals from the satellites in order to calculate a fix.

numberOfSatellitesUsed – This is the number of satellites used when calculating the current fix. Maximum number of satellites are 12 for the GPS receiver.

horizontalDilutionOfPres – This is the precision of the horizontal dilution. The lower this value is, the better. Generally < 2.0 is good.

altitude – This is the mean-sea-level of the GPS antenna given in meters.

diffWGS84ref – The difference between the WGS-84 reference ellipsoid surface and the mean-sea-level altitude.

numberOfReceivedMsgs – The number of GGA messages that has been received since the GPS_Init() function was called for the first time. The value is reset when the runtime is shut down.

8.3 GPS_GSA type

Below is the declaration of the GPS_GSA data type:

```
TYPE GPS_GSA :
STRUCT
            mode1:STRING(1);
            mode2:USINT;
            satelliteIDs : ARRAY[1..12] OF USINT;
            PDOP:REAL;
            HDOP:REAL;
            VDOP:REAL;
            numberOfReceivedMsgs:DWORD;

END_STRUCT
END_TYPE
```

Below is a description of the variables:

mode1 – A string that can contain either 'M' or 'A'. 'M' means that the GPS unit is forced to operate in either 2D or 3D. 'A' means that the GPS unit automatically switch between 2D and 3D.

mode2 – The values 1,2 and 3 are possible. 1 means that fix is not available. 2 means that the GPS is in 2D mode. 3 means that the GPS is in 3D mode.

satelliteIDs – This is an array of 12 bytes that contains the PRN numbers of the satellites currently used by the GPS. If for example 5 satellites are used by the GPS, the first 5 bytes will contain the satellites PRN numbers, and the rest of the array will contain zeros.

PDOP – Position Dilution of Precision. A small value means good precision.

HDOP – Horizonal Dilution of Precision. A small value means good precision.

VDOP – Vertical Dilution of Precision. A small value means good precision.

numberOfReceivedMsgs – The number of GSA messages that has been received since the GPS_Init() function was called for the first time. The value is reset when the runtime is shut down.

8.4 GPS_GSV type

Below is the declaration of the GPS_GSV data type:

```
TYPE GPS_GSV :
STRUCT
            totalSatellitesInView : UINT;
            satelliteIds : ARRAY[1..36] OF USINT;
            satelliteElevations : ARRAY[1..36] OF USINT;
            satelliteAzimuths : ARRAY[1..36] OF UINT;
            satelliteSNRs : ARRAY[1..36] OF USINT;
            numberOfReceivedMsgs:DWORD;

END_STRUCT
END_TYPE
```

Below is a description of the variables:

totalSatellitesInView – The total number of satellites in view. The maximum number is 36. Note that the receiver can only use up to 12 satellites for calculation of fix.

satelliteIds – An array of 36 bytes that contains the PRN ids of the satellites in view.

satelliteElevations – An array of 36 bytes that contains the elevation of the satellites in view. Maximum is 90 degrees.

satelliteAzimuths – An array of 36 bytes that contains the azimuth of the satellites in view. Maximum is 359 degrees.

satelliteSNRs – An array of 36 bytes that contains the SNR (C/No) value of the satellites in view. Value 0-99 dB-Hz. Zero when not tracking.

numberOfReceivedMsgs – The number of GSV messages that has been received since the GPS_Init() function was called for the first time. The value is reset when the runtime is shut down.

8.5 GPS_RMC type

Below is the declaration of the GPS_RMC data type:

```
TYPE GPS_RMC :
STRUCT
            utc_Time:STRING(20);
            valid:USINT;
            latitude:DWORD;
            northSouth:STRING(1);
            longitude:DWORD;
```

```
                eastWest:STRING(1);
                speed:REAL;
                heading:REAL;
                gps_Date:STRING(20);
                magneticVariation:REAL;
                declination:STRING(1);
                mode:STRING(1);
                numberOfReceivedMsgs:DWORD;

END_STRUCT
END_TYPE
```

Below is a description of the variables:

utc_time – This variable contains the UTC time as a string in the format “hhmmss.dd” where hh is hours, mm is minutes, ss is seconds, and is the decimal part of seconds.

valid – This variable tells if the fix is valid or not. Possible values are 1 (valid) or 0 (not valid). If the fix is not valid, this means that the GPS receiver does not have enough signals from the satellites in order to calculate a fix.

latitude - This is the latitude as a DWORD. The format is xxmmdddd where xx is degrees, mm is minutes, and dddd is the decimal part of minutes. In NMEA message this value is supplied as a decimal value xxmm.dddd. The value received is multiplied by 10,000 and inserted in the latitude variable.

northSouth – This is the latitude north/south indicator as a string. The possible values are ‘N’ (north) or ‘S’ (south).

longitude – This is the longitude as a DWORD. The format is yyymmdddd where yyy is degrees, mm is minutes, and dddd is the decimal part of minutes. In NMEA message this value is supplied as a decimal value yyymm.dddd. The value received is multiplied by 10,000 and inserted in the longitude variable.

eastWest – This is the east/west indicator as a string. The possible values are ‘E’ (east) or ‘W’ (west).

speed – Speed over ground in knots.

Heading – Heading in degrees.

*gps_Dat*e – String that contains date, month and year in the format ddmmyy. dd – Date, mm – month, yy – year.

magneticVariation – Magnetic variation in degrees. Subtracts from heading.

declination – String that contains the direction of the magnetic variation. Either ‘W’ (west) or ‘E’ (east).

mode – String that contains either ‘A’ (autonomous mode), or ‘N’ (data not valid’).