FALCOM
WIRELESS COMMUNICATIONS GMBH

SiRFstarII System

Development Kit

SiRFgps User's Guide

Preliminary

Version 1.00, 09/02/2005

Index of contents

Version history:

Version number	Author	Changes
1.00	Fadil Beqiri	Initial version

0 DOCUMENT OVERVIEW

This document provides user guidance to the SiRFstarII-based portable GPS receiver system option.

The Falcom assumes no responsibility for any errors, which may appear in this document, nor does it make a commitment to update the information contained herein.

The Falcom reserves a right to make changes to this document at any time, without a notice. This manual applies to SiRFgps library version: 1.0 2.04.047 4.49.

1 SIRFGPS GPS LIBRARY

SiRFgps Library is a heart of the SiRFstarII GPS system. It controls GPS hardware tracker, collects tracker data, computes position, time and other parameters, and communicates with user software.

Host system typically runs following processes/tasks:

- User Applications -GPS consumer (mapping, routing, etc.) and other user's tasks
- GPS Component -interfaces with User Applications and controls SiRFgps Library
- eCos Operating system

GPS Component is grouped into following main functional blocks:

- GPS Engine - the core of the SiRFgps
- GPS Component Interface - GPS device driver or direct link to User Application
- User Interface
- Tracker ASIC Interface
- eCos Operating System Interface

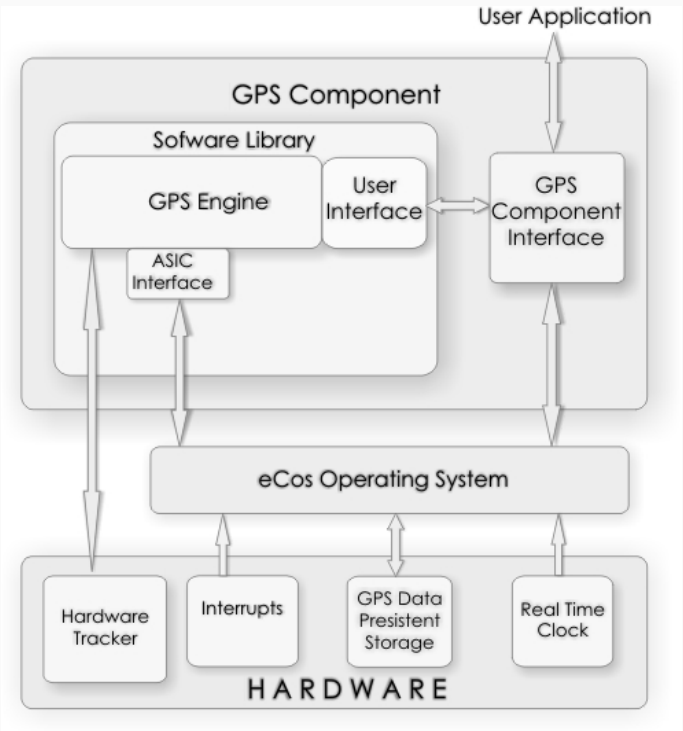

Figure 1: SiRFgps Software Architecture

The GPS Component Interface can be a device driver, other eCos OS interface or a direct link to the User Application. GPS Component Interface can be developed by the user or in joint development with Falcom. Information provided through the User Interface is in SiRF data structure format. This data (latitude, longitude, time, etc.) can be consumed directly or used to generate NMEA or other required data format.

1.1 Files

SiRFgps Library:

- SiRFgpsServer.a – pre-compiled SiRFgps Library for GCC compiler and eCos OS

Header files:

- gps_types.h – defines standard types used in all GPS interfaces
- gps_ctrl.h – control functions of the User Interface
- gps_interface.h – communication functions of the User Interface
- gps_messages.h –messages used in GPS communication and configuration
- gps_trk_asic.h – functions of the Tracker ASIC Interface
- gps_rtos.h – functions of the eCos OS Interface

Source files for eCos OS and ASIC Interfaces (can be used to re-build SiRFgps Library):

- gps_rtos.c – eCos Operating System Interface
- gps_trk_asic.c – Tracker ASIC Interface

2 INTERFACES

SiRFgps Library interacts with the host system through the following three interfaces:

User Interface -called by user to start, configure, get data and stop GPS Engine. GPS Engine calls user's UI_event() function to deliver GPS information (position, time, and other synchronous and asynchronous (data)

Tracker ASIC Interface

ECos Operating System Interface -tracker interrupt handlers and ASIC initialization

-calls eCos OS services for task scheduling, synchronization, RTC and storage access

2.1 User Interface

GPS engine control

GPS_Start() *This function initializes and starts GPS engine's tasks and communication interface. Function should be called by client's application to start GPS activity.*

GPS_Stop() *This function suspends GPS engine activity or deletes GPS engine's threads and communication interface. Function should be called by client's application to stop GPS activity.*

Module Interface events processing

UI_Event(`MI_EVENT Event, tGPS_UINT32 Time`)

This function is called periodically (every 1 ms) from the GPS library. The user-application can be adapted, depending on the generated events listed below, which will be passed to the application layer (user interface). All of these events signalling the status of GPS core.

This Function should be implemented by the user, and automatically called when GPS is started.

One of the following values (events)can be sent out to the user interface.

```
MODULE INTERFACE EVENTS:
typedef enum
{
   MI_EV_NONE                  = 0x0000,
   MI_EV_MEASUREMENT_RCVD      = 0x0001,
   MI_EV_NAV_COMPLETE          = 0x0002,
   MI_EV_NEW_VISIBLE_LIST      = 0x0004,
   MI_EV_NEW_50BPS             = 0x0008,
   MI_EV_NEW_ALMANAC           = 0x0010,
   MI_EV_NEW_EPHEMERIS         = 0x0020,
   MI_EV_INITIAL_ACQ_COMPLETE  = 0x0040,
   MI_EV_KRAUSE_COMPLETE       = 0x0080,
   MI_EV_WAIT_INITIAL_ACQ      = 0x0100,
   MI_EV_INITIAL_POSITION      = 0x0200,
```

```
   MI_EV_MEASUPDATE            = 0x0400,
   MI_EV_NL_INIT_DONE          = 0x0800,
   MI_EV_USER                  = 0x1000,
   MI_EV_DR_HI_RATE            = 0x1000,
   MI_EV_DR_1HZ_RATE           = 0x2000,
   MI_EV_PPS_MESSAGE           = 0x4000,
   MI_EV_ALL                   = 0xFFFF
} MI_EVENT;
```

Geodetic navigation output message, message ID = 41

```
      typedef     struct
      {
tGPS_UINT16   nav_valid;   //GPS validity flags (bitmap)
tGPS_UIKT16   nav_mode ;   //GPS navigation mode flags (bitmap)
tGPS_UINT16   gps_week;    //GPS week number
tGPS_UINT32   gps_tow;     //GPS time of the week in 1 ms
tGPS_UINT16   utc_year;    //UTC year
tGPS_UIKT8    utc_month;   //UTC month
tGPS_UINT8    utc_day;     //UTC day
tGPS_UINT8    utc_hour;    //UTC hour
tGPS_UINT8    utc_min;     //UTC minute
tGPS_UINT16   utc_sec;     //UTC second in 1 ms
tGPS_UINT32   sv_used;     //Satellites used in i.'.i x (bitmap)
tGPS_SINT32   lat;         //Latitude in le-7 degrees
tGPS_SINT32   lon;         //Longitude in le-7 degrees
tGPS_SINT32   alt_ellipse;//Altitude from ellipsoid in 0.01 meters
tGPS_SINT32   alt_msl;     //Altitude lrom Mean Sea Level in 0.01 meter;
tGPS_UINT8    datum;       //Datum code
tGPS_UINT16   sog;         //Speed Over Ground in 0.01 meters/sec
tGPS_UIKT16   cog;         //Course Over Ground,from True North in 0.01
                             degrees
tGPS_SINT16   reserved;
tGPS_SINT16   reserved;
tGPS_SINT16   reserved;
tGPS_UINT32   ehpe;        //Estimated horizontal position error in
                             meters
tGPS_UIKT32   evpe;        //Estimated vertical position error in
                             meters
tGPS_UINT32   reserved;
tGPS_UINT16   reserved;
tGPS_SINT32   clk_bias;    //GPS clock bias
tGPS_UINT32   clk_bias_error; //Estimated GPS clock bias error
tGPS_SINT32   clk_offset;  //GPS clock offset
tGPS_OINT32   clk_offset_error; //Estimated GPS clock offset error
tGPS_UINT32   reserved;
tGPS_UINT16   reserved;
tGPS_UINT16   reserved;
tGPS_UINT8    sv_used_cnt; //Number of satellites used in solution
```

```
tGPS_UINT8    hdop;        //Horizontal Dillution Of Precision
tGPS UINT8    reserved;
      }  tGPS_NAV_GEODETIC_NAVIGATION;
```

Tracker data output message, message ID = 4

```
      typedef struct
      {
tGPS_SINT16 gps_week;     //GPS week number
tGPS_UINT32 gps_tow;      //GPS time of the week in 10 ms
tGPS_UINT8  chnl_cnt;     //Channels count
struct
{
  tGPS_UINT8  svid;       //Satellite ID
  tGPS_UINT8  azimuth;    //Satellite azimuth in 1.5 degs
  tGPS_UINT8 elevation;   //Satellite elevation in 0.5 degs
  tGPS_UINT16 state;      //Satellite tracking state (bitmap)
  tGPS_UINT8  cno[10];    //Satellite C/No in dB-Hz
} chnl[12];
      }  tGPS_NAV_MEASURED_TRACKER;
```

GPS receiver initialization input message, message ID = 428

```
      typedef struct
      {
tGPS_SINT32 lat;          //Latitude in degrees
tGPS_SINT32 lon;          //Longitude in degrees
tGPS_SINT32 alt_ellips;   //Altitude in meters from ellipsoid
tGPS_UINT16 heading;      //Heading in degrees
tGPS_SINT32 clk_offset;   //Clock Offset in Hz
tGPS_UINT32 gps_tow;      //GPS Time of Week in 10 ms
tGPS_UINT16 gps_week;     //GPS Week number
tGPS_UINT8  chnl_cnt;     //Number of channels
tGPS_UINT8  restart_flags //Reset Configuration {bitmap)
      }  tGPS_DR_SET_NAV_INIT;
```

2.2 Tracker ASIC Interface

Following functions are implemented by SiRF and provided in open source. Availability and exact functionality of these functions depends on the operating system.

ASIC ←→ GPS Tracker Engine

`GPS_ASIC_Initialize()`	*Initializes GPS ASIC*
`GPS_ASIC_INTR_lms_Install()`	*Installs tracker's lms interrupt handlers (low and high level handler)*
`GPS_ASIC_INTR_100ms_Install()`	*Installs tracker's 100ms interrupt handlers (low and high level handler)*

`GPS_ASIC_INTR_1ms_Activate_HISR()`	*Activates lms high level interrupt handler*
`GPS_ASIC_INTR_100ms_Activate_HISR()`	*Activates 100ms high level interrupt handler*
`GPS_ASIC_INTR_1ms_Control()`	*Enables and disables lms interrupt*
`GPS_ASIC_INTR_100ms_Control{)`	*Enables and disables 100ms interrupt*
`GPS_ASIC_INTR_All_Disable()`	*Disables all interrupts*
`GPS_ASIC_INTR_All_Restore()`	*Restores all interrupts state*

2.3 Operating System Interface

Following functions are implemented by SiRF and provided in open source. Availability and exact functionality of these functions depends on the operating system and hardware configuration.

`OS_Thread_Create()`
`OS_Thread_Deletef)`
`OS_Thread_Sleep()`
	Thread startup and suspension functions

`OS_Mutex_Create()`
`OS_Mutex_Delete()`
`OS_Mutex_Enter()`
`OS_Mutex_Exit()`
	Mutex synchronization functions

`OS_Semaphore_Create()`
`OS_Semaphore_Delete()`
`OS_Semaphore_Wait()`
`OS_Semaphore_Release()`
	Signaling synchronization functions

`OS_Storage_Open()`
`OS_Storage_Close()`
`OS_Storage_Write()`
`03_SL-oraqe_Read ()`
	Persistent storage functions

`OS_RTC_Read()`	*Reads time from host RTC*

3 GPS DEMO

A simple application that outputs the current position in NMEA format (NMEA protocols) to a serial port.

Settings:

Output: NMEA (GGA, GSV, GSA and RMC)

Input: PSRF101, PRF102

CLASS:

comm_handler: Basis class that communicates with device driver e.g. with serial interface.

nmea_handler: the class inherits directly from *comm_handler*, that serves to output the GPS protocols in NMEA format. The PSRF commands can be read and correspondingly implemented.

sirf_handler: the class inherits directly from *comm_handler*, that serves to output the GPS protocols in SiRF Binary format.

PROCEDURE:

To read and write from/to the serial interface, the *read(buffer)* and *write(buffer)* methode is available for use.

The *input ()* und *output ()* will be called from the higher-level application, means from UI_Event(). The UI_Event() is called every 1 ms from the GPS Library.

All instances of the *com_port* class are kept statically in the *privat_instances[]* list and will be through the *threadcommunicationTask()* periodically called.

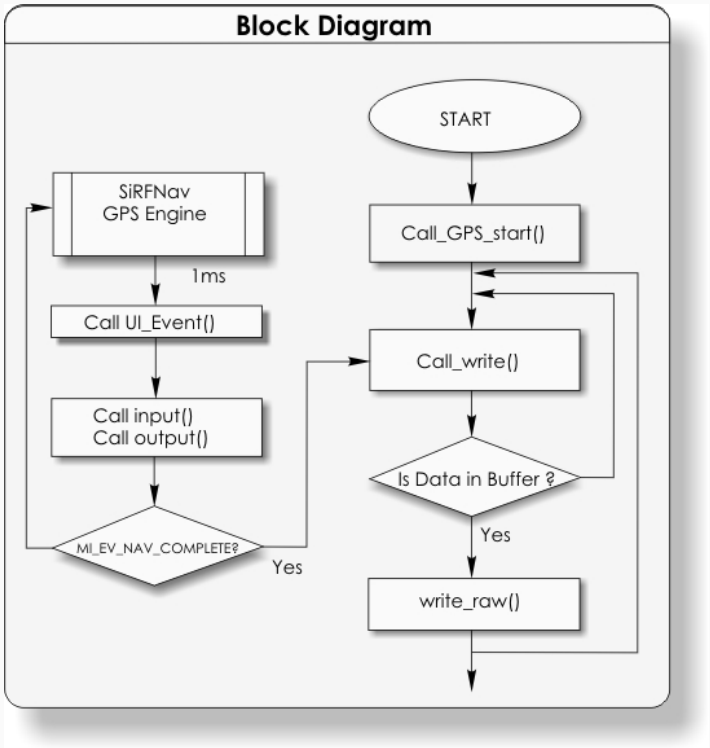

Figure 2: Typical User Task Flowchart