View Single Post Thread: Discussion FrSky S-Port telemetry library - easy to use and configurable

Sep 14, 2014

pawelsky

FrSky S-Port telemetry library - easy to use and configurable

Hi,

for those having Tarains radios or other system capable of receiving FrSky S-Port telemetry data I've created a library, that allows to emulate and/or decode the FrSky S-Port sensors or using Arduino compatible Teensy 3.x board or 5V/16MHz 328P based boards (e.g. ProMini, Nano, Uno).

If you still use old FrSky telemetry, there is also a library for that as well. You ca fin it here:
http://www.rcgroups.com/forums/showthread.php?t=2465555

The library initially created to work together with the NazaDecoder or NazaCanDecoder libraries that read telemetry data coming out of DJI Naza Lite/v1/v2 controllers (examples can be found in post #36), but it can be used for other purposes as well.

The main goals that I had when creating this library was to have it:
1) easy to use
2) easy to configure
3) easy to add new sensors

The library consists of 3 main classes:

1. FrSkySportTelemetry - which is the main class responsible for collecting the data and sending it via one of the Teensy serial ports or SoftwareSerial ports on 328P based borads.
2. FrSkySportSingleWireSerial - which is responsible for S-Port-like single wire serial transmission
3. FrSkySportSensor - which is a base class for all the sensors. So far following sensors have been implemented (with the exception of the last one all of them can both emulate and decode the data):
    - FrSkySportSensorAss - which emulates ASS-70/ASS-100 airspeed sensors
    - FrSkySportSensorFcs - which emulates FCS-40A/FCS-150A current sensors
    - FrSkySportSensotFlvss - which emulates FLVSS LiPo voltage monitor sensor
    - FrskySportSensorGps - which emulates the GPS v2 sensor
    - FrskySportSensorRpm - which emulates the RPM/temperature sensor
    - FrSkySportSensorSp2uart - which emulates the S.Port to UART Converter type B sensor (analog ADC3/ADC4 inputs only)
    - FrskySportSensorVario - which emulates the high precision variometer sensor
    - FrskySportSensorXjt- which is a special sensor that is only capable of decoding the additional (i.e. other than from the above sensors) data stream coming from the XJT transmitter module's S.Port. This data includes the old type (hub) telemetry and the special data such as ADC1, ADC2, SWR, RSSI and RxBatt.

The library is very easy to use, you can define which sensors to include, you can change their default IDs (e.g. to avoid conflicts or to use 2 FLVSS sensors to 12S batteries). The library makes sure that the sensors respond to correct ID being polled/transmitted (not just any ID or a fixed list of IDs as in some of

the existing libraries) so you can even mix the virtual sensors with the real ones without conflicts. You can also define which of the Teensy serial ports to use. In fact you can send/decode different data on different ports by creating multiple telemetry objects.

The library can be used for both emulating the S.Port sensors and decoding the S.Port data, but **never at the same time!**

Attached you'll find the library itself and a picture showing the encoder code in action. As you can see the data displayed for FCS and FLVSS sensor matches the data set in code.

ENCODER - emulating the S.Port sensor

Here is a quick example on how the library is used to emulate a sensor

Code:

```
#include "FrSkySportSensorFlvss.h"
#include "FrSkySportSensorFcs.h"
#include "FrSkySportSingleWireSerial.h"
#include "FrSkySportTelemetry.h"

FrSkySportSensorFlvss flvss1;
FrSkySportSensorFlvss flvss2(FrSkySportSensor::ID15);
FrSkySportSensorFcs fcs;
FrSkySportTelemetry telemetry;

void setup()
{
  telemetry.begin(FrSkySportSingleWireSerial::SERIAL_3, &flvss1, &flvss
}

void loop()
{

  /* DO YOUR STUFF HERE */

  flvss1.setData(4.10, 4.11, 4.12, 4.13, 4.14, 4.15);
  flvss2.setData(3.10, 3.11, 3.12, 3.13, 3.14, 3.15);
  fcs.setData(25.3, 12.6);
  telemetry.send();
}
```

As you can see it is super simple, all you need to do is to:
1) create the sensors you want to use and the telemetry object
2) call the begin method of the telemetry object defining the port and pointers to used sensors (up to 28)
3) update the sensor data when necessary
4) call the send method of the telemetry object periodically

For simplicity it is not possible to remotely change data send frequency which every sensor has defined, the values are as specified by FrSky.

Have a look at the FrSkySportTelemetryExample included in the library to have a better understanding on what parameters can be set for each of the sensors.

DECODER - decoding the S.Port data

Here is a quick example on how the library is used to decode a S.Port Sensor data.

Code:

```
#include "FrSkySportSensorFlvss.h"
#include "FrSkySportSensorFcs.h"
#include "FrSkySportSingleWireSerial.h"
#include "FrSkySportDecoder.h"

FrSkySportSensorFlvss flvss1;
FrSkySportSensorFlvss flvss2(FrSkySportSensor::ID15);
FrSkySportSensorFcs fcs;
FrSkySportDecoder decoder;

void setup()
{
  decoder.begin(FrSkySportSingleWireSerial::SERIAL_3, &flvss1, &flvss2,
}

void loop()
{
  // Call this on every loop
  decoder.decode();

  // Make sure that all the operations below are short or call them per
  flvss1.getCell1();  // Read cell 1 voltage from the standard FLVSS se
  flvss2.getCell2();  // Read cell 2 voltage from the ID15 FLVSS sensor
  fcs.getCurrent();  // Read current from the fcs sensor and do somethi

   /* DO YOUR STUFF HERE */

}
```

As you can see it is super simple, all you need to do is to:
1) create the sensors you want to use and the decoder object
2) call the begin method of the decoder object defining the port and pointers to used sensors (up to 28)
4) call the decode method of the decoder object on every loop
3) read the data from sensors and do something with it

You can decode data from real sensors, my telemetry library, from Taranis serial port that can be found in the battery compartment (make sure it is configured to mirror S.Port data in radio menu)or S-Port connector in the Taranis JR TX module bay.

Note that if you want to connect to a S.Port sensor directly you also need to daisy chain an S.Port receiver as there must be something polling the S.Port telemetry data from sensors. The library does not poll by itself, just listens.

You'll find more detailed example in the FrSkySportDecoderExample and FrSkySportXjtDecoderExample directory.

Connections

Below you can also find attached connection diagrams for both encoder and decoder (note that they are different). Pick one of the Teensy 3.x TX (for encoder)/RX (for decoder) or 328P based bard 2-12 pins (one that is not used

for other purposes) and tell the library to use it in ***telemetry.begin*** function. Depending on the board used you should chose:

- **SERIAL_1**, **SERIAL_2** or **SERIAL_3** for Teensy 3.x boards (note that the additional SERIAL_USB shall only be used for debug purposes if you know what you are doing)
- **SOFT_SERIAL_PIN_2** to **SOFT_SERIAL_PIN_12** for 328P based boards (e.g. Pro Mini, Nano, Uno)

Note that for 328P based boards you need to include the SoftwareSerial library in your main sketch:

Code:

```
#include "SoftwareSerial.h"
```

There were reports that for some reason the library does not work when used with 1.6.0 version of Arduino IDE. This is most likely due to the changed toolchain and modified timings. If you experience problems please upgrade to a newer version where the timing calculation has been modified (1.6.3 has been confirmed working) - that shall help.

Also note that to avoid interrupt conflict with the mentioned SoftwareSerial library you need to disable attitude (roll/pitch) sensing in the NazaDecoder library by uncommenting the following line in NazaDecoder.h file:

Code:

```
//#define ATTITUDE_SENSING_DISABLED
```

Make sure you understand Teensy 3.x/328P based board powering options before choosing how to power your board!

When creating my library I used some of the information that I found on this webpage:
https://code.google.com/p/telemetry-...ySPortProtocol

FrSky S.Port Telemetry library changelog

Version 20160324
[NEW] Increased the max number of sensors from 10 to 28 (to cover all possible sensor IDs)

Version 20160313
[FIX] Added missing getAccX, getAccY and getAccZ function names to keywords file for syntax highlighting
[FIX] Added missing getCell7 - getCell12 function names to keywords file for syntax highlighting
[FIX] Changed the FrSkySportSensorTaranisLegacy decoder class name to FrSkySportSensorXjt
[NEW] Added usage example for the FrSkySportSensorXjt decoder class (FrSkySportXjtDecoderExample)
[NEW] Added decoding of ADC1, ADC2, RSSI, SWR and RxBatt data to FrSkySportSensorXjt
[NEW] Added decoding of vertical speed from FVAS sensor (not documented in FrSky spec, added based on OpenTX sources)

Version 20151108
[FIX] For combined values decoded by the FrSkySportSensorTaranisLegacy sensor the data ID is only returned when complete value has been decoded
[FIX] Fixed handling empty FLVSS data message that was crashing the decoder
[NEW] Added defines for FrSkySportSensorTaranisLegacy sensor decoded values
[NEW] Added decoded value IDs to keywords file for syntax highlighting
[NEW] Added clarification about different IDs used by FCS-40A and FCS-150A sensors

in examples and sensor class header file

Version 20151020
[FIX] Stuffing was not used for CRC, fixed
[FIX] Fixed lat/lon calculation in FrSkySportSensorTaranisLegacy sensor

Version 20151018
[NEW] Added Taranis legacy telemetry decoder class (which decodes data that Taranis spits out on teh S.Port mirror when it receives old FrSky Telemetry Hub data)
[NEW] Added CRC checking in decoder class
[NEW] Added return result (decoded appId) to decode methods and updated the decoder example
[FIX] Removed redundant CRC calculations in FrSkySportSingleWireSerial class
[FIX] Simplified cell voltage decoding

Version 20151008
[NEW] Added Decoder class and decoding functions to sensors (plus the decoder usage example)
[FIX] In RPM sensor changed the rpm value type to integer, t1/t2 are now properly rounded
[FIX] Minor editorial corrections

Version 20150921
[NEW] Added airspeed (ASS-70/ASS-100) sensor.

Version 20150725
[NEW] Added data transmission periods to ensure telemetry receiver is not flooded with data
[NEW] Added SP2UART (type B) sensor. Note that only analog ports ADC3 and ADC4 are implemented, not the UART part.

Version 20150319
[FIX] corrected setting the 328p serial pin to the right mode when transmitting/receiving. This shall help when chaining the adapter with other sensors. Note that a 4.7kohm resistor is recommended to protect the S.Port data line.

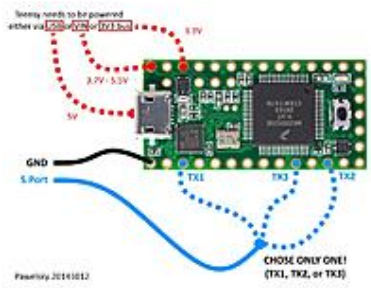
Version 20141129
[FIX] fixed incorrect display of GPS coordinates on 328p platform (caused by wrong usage of abs function)

Version 20141120
[NEW] added support for 328P based boards (e.g. Pro Mini, Nano, Uno)
[NEW] added connection diagrams

Version 20140914
initial version of the library

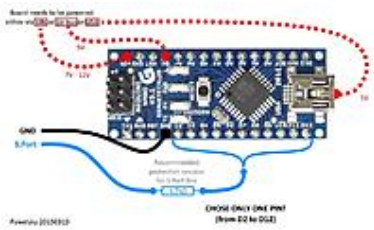
Images View all Images in thread

Views: 1219



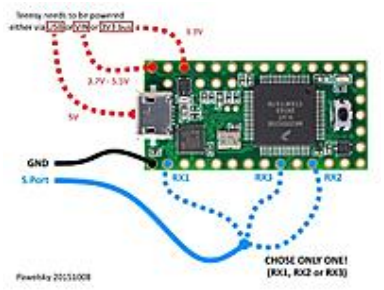
Views: 3455

ENCODER connection diagram for Teensy 3.x boards



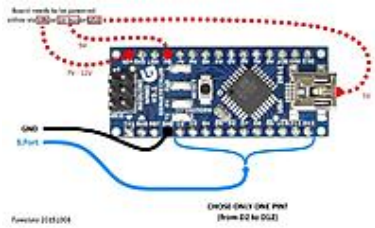
Views: 1904

ENCODER connection diagram for 328P based boards (e.g. Pro Mini, Nano, Uno). Nano used as an example here. Although the resistor is not absolutely necessary, due to the voltage difference between Arduino and S.Port (5V vs 3.3V) it is strongly recommended



Views: 258

DECODER connection diagram for Teensy 3.x boards



Views: 309

DECODER connection diagram for 328P based boards (e.g. Pro Mini, Nano, Uno). Nano used as an example here.

Files

View all Files in thread

FrSkySportTelemetry_201603 v20160324 (451.7 KB) 314 views

Last edited by pawelsky; Mar 24, 2016 at 12:15 PM.

Quote