

[Forums](#)[Magazine](#)[More](#)[Search](#)[Sign Up](#) | [Log In](#)[Aircraft - General](#)[Radios](#)[Discussion](#) FrSky S-Port telemetry library - easy to use and configurable

Page 1 of 85

[Next](#)[Last](#)

Thread Tools

Sep 14, 2014

#1

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/4.0/LC](#) board, ESP8266 or 5V/16MHz ATmega328P/ATmega2560 based boards (e.g. ProMini, Nano, Uno, Mega).

If you still use old FrSky telemetry, there is also a library for that as well. You can find it here:

<https://www.rcgroups.com/forums/show....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 5 main classes:

1. FrSkySportTelemetry - which is responsible for sending the S.Port data via one of the Teensy serial ports or SoftwareSerial ports on ESP8266 and ATmega328P/ATmega2560 based boards.
2. FrSkySportDecoder - which is responsible for decoding the data coming from the S.Port line via one of the Teensy serial ports or SoftwareSerial ports on ESP8266 and ATmega328P/ATmega2560 based boards.
3. FrSkySportPolling - which is a base class for emulating data polling (useful when there is no device actively polling the S.Port data such as the receiver). There are following two polling methods available:
 - FrSkySportPollingSimple - which simply polls each sensor one after another in a loop (default)
 - FrSkySportPollingDynamic - which polls responding sensors more often (just like FrSky receivers do)

4. FrSkySportSingleWireSerial - which is responsible for S-Port-like single wire serial transmission.
5. 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
 - FrSkySportSensorEsc - which emulates [Neuron ESC](#) sensors (including SBEC)
 - FrSkySportSensorFcs - which emulates [FCS-40A/FCS-150A](#) current sensors
 - FrSkySportSensorFlvss - which emulates [FLVSS /MLVSS](#) LiPo voltage monitor sensor
 - FrskySportSensorGasSuite - which emulates the [Gas Suite](#) 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.

This library is for **non-commercial use only**, and no, **I do not plan to put this code on Github**.

ENCODER - emulating the S.Port sensor

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

Code:

```
#include "FrSkySportSensor.h"
#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, &flvss2, &fcs);
}

void loop()
{

    /* DO YOUR STUFF HERE */
    /* Make sure you do not do any blocking calls in
the loop, e.g. delay()!!! */

```

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 (i.e. within milliseconds from previous call)

Sensors will use their default physical IDs, but this can be changed by specifying different ID when the sensor object is created (see flvss2 sensor in the example above).

Make sure you DO NOT BLOCK THE LOOP (milliseconds matter) with calls to methods such as delay() or other time consuming calls to other libraries!!!

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 "FrSkySportSensor.h"
#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, &fcs);
}

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

    // Make sure that all the operations below are
    short or call them periodically otherwise you'll be
    losing telemetry data
    flvss1.getCell1(); // Read cell 1 voltage from
    the standard FLVSS sensor and do something with it
}

```

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 (i.e. within milliseconds from previous call)

Sensors will use their default physical IDs, but this can be changed by specifying different ID when the sensor object is created (see flvss2 sensor in the example above). When you want the sensor data to be decoded regardless of which physical ID it comes from use the special

FrSkySportSensor::ID_IGNORE ID, but be aware that in this case you won't be able to differentiate which particular sensor this data came from.

Make sure you DO NOT BLOCK THE LOOP (milliseconds matter) with calls to methods such as delay() or other time consuming calls to other libraries!!!

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.

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

Data polling

To send the data the S.Port sensors must be actively polled. Normally that is done by the receiver that the sensor is connected to. If for whatever reason you decide to use this library in a configuration that does not have the receiver (or any other device actively polling) in the S.Port chain you have to enable library's internal polling mechanisms. The library offers two types of polling:

- **simple** (legacy) implemented by class FrSkySportPollingSimple, where each sensor is polled one after another in a loop. This means that regardless of whether the sensor is active or not it will have to wait its turn to report until the whole loop repeats
- **dynamic** (FrSky-like) implemented by class FrSkySportPollingDynamic, where active sensors are polled more often, without waiting for all the inactive ones, improving the data refresh rate. This is similar to how the original FrSky receivers do polling, e.g. if there is one active sensor it will be polled every 24ms

instead of 336ms, when there are two active sensors they will be polled every 36ms instead of 336ms, etc.

Table below shows an example comparison on how sensors are polled with Simple and Dynamic polling when sensors ID1, ID3 and ID4 are responding (marked with asterisk *)

Code:

Time	Simple	Dynamic
12ms	ID1*	ID1*
24ms	ID2	ID1*
36ms	ID3*	ID2
48ms	ID4*	ID1*
60ms	ID5	ID3*
72ms	ID6	ID1*
84ms	ID7	ID3*
96ms	ID8	ID4*
108ms	ID9	ID1*
120ms	ID10	ID3*
132ms	ID11	ID4*
144ms	ID12	ID5
156ms	ID13	ID1*
168ms	ID14	ID3*
180ms	ID15	ID4*
192ms	ID16	ID6
204ms	ID17	ID1*
216ms	ID18	ID3*
228ms	ID19	ID4*
240ms	ID20	ID7
252ms	ID21	ID1*
264ms	ID22	ID3*
276ms	ID23	ID4*
288ms	ID24	ID8
300ms	ID25	ID1*
312ms	ID26	ID3*
324ms	ID27	ID4*

To enable polling you need to instantiate the FrSkySportTelemetry or FrSkySportDecoder class object passing the instance of and **FrSkySportPollingSimple** or **FrSkySportPollingDynamic** class object as a parameter. If you don't do that **NULL** (polling disabled) will be used as a default. Don't forget to #include an appropriate header file.

Code:

```
FrSkySportTelemetry telemetry(new
FrSkySportPollingDynamic()); // Create telemetry
```

Code:

```
FrSkySportDecoder decoder(new
FrSkySportPollingSimple()); // Create decoder object
```

In examples, to make things easier polling can be enabled by uncommenting the following line

Code:

```
//#define POLLING_ENABLED
```

For the telemetry to be properly decoded on the Taranis radio RSSI data is required, so when polling is enabled the library also sends out RSSI data at regular intervals, accompanied by the RxBatt data (both can be set by the user)

As both encoder and decoder have the polling capability, if you use both in your setup make sure that only one or the other has the polling enabled, not both at the same time.

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/LC serial TX or 328P based board 2-12 pins (one that is not used for other purposes) and tell the library to use it in ***telemetry.begin*** or ***decoder.begin*** function. Depending on the board used you should chose (NOTE: **SERIAL_x** and **SOFT_SERIAL_PIN_x** means inverted, single-wire connection, while **SERIAL_x_EXTINV** means uninverted two-wire connection and expects signal to be inverted externally):

- **SERIAL_1, SERIAL_2, SERIAL_3, SERIAL_1_EXTINV, SERIAL_2_EXTINV or SERIAL_3_EXTINV** for Teensy 3.0/3.1/3.2/LC boards (note that the additional SERIAL_USB shall only be used for debug purposes if you know what you are doing)
- **SERIAL_1, SERIAL_2, SERIAL_3, SERIAL_4, SERIAL_5, SERIAL_6, SERIAL_1_EXTINV, SERIAL_2_EXTINV, SERIAL_3_EXTINV, SERIAL_4_EXTINV, SERIAL_5_EXTINV or SERIAL_6_EXTINV** for Teensy 3.5/3.6 boards (note that the additional SERIAL_USB shall only be used for debug purposes if you know what you are doing)
- **SERIAL_1, SERIAL_2, SERIAL_3, SERIAL_4, SERIAL_5, SERIAL_6, SERIAL_7, SERIAL_1_EXTINV, SERIAL_2_EXTINV, SERIAL_3_EXTINV, SERIAL_4_EXTINV, SERIAL_5_EXTINV, SERIAL_6_EXTINV or SERIAL_7_EXTINV** for Teensy 4.0/4.1 board (note that the additional SERIAL_USB shall only be used for debug purposes if you know what you are doing). Note that for Teensy 4.1 there is an additional Serial8 available for which you can use **SERIAL_8** or **SERIAL_8_EXTINV**
- **SOFT_SERIAL_PIN_4/SOFT_SERIAL_PIN_D2, SOFT_SERIAL_PIN_5/SOFT_SERIAL_PIN_D1, SOFT_SERIAL_PIN_12/SOFT_SERIAL_PIN_D6, SOFT_SERIAL_PIN_13/SOFT_SERIAL_PIN_D7, SOFT_SERIAL_PIN_14/SOFT_SERIAL_PIN_D5, SOFT_SERIAL_PIN_15/SOFT_SERIAL_PIN_D8 or SERIAL_EXTINV** for ESP8266 based boards
- **SOFT_SERIAL_PIN_2 to SOFT_SERIAL_PIN_12 or SERIAL_EXTINV** for ATmega328P based boards (e.g. Pro Mini, Nano, Uno)
- **SOFT_SERIAL_PIN_10, SOFT_SERIAL_PIN_11, SOFT_SERIAL_PIN_12, SOFT_SERIAL_PIN_13, SOFT_SERIAL_PIN_14, SOFT_SERIAL_PIN_15, SOFT_SERIAL_PIN_50, SOFT_SERIAL_PIN_51, SOFT_SERIAL_PIN_52, SOFT_SERIAL_PIN_53, SERIAL_EXTINV, SERIAL_1_EXTINV, SERIAL_2_EXTINV or SERIAL_3_EXTINV** for ATmega2560 based boards (e.g. Mega)

Note that for ESP8266 and ATmega328P/ATmega2560 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/4.0/LC/ESP8266/ATmega328P/ATmega2560 based board powering options before choosing how to power your board!

Supported Application IDs and defaults

The table below shows all Application IDs supported by the sensor classes (except FrskySportSensorXjt), together with default Physical IDs and data poll times

Code:

Sensor class	Phys.	App ID	
#define	Data type		Poll
time			
	ID		
	[ms]		
-----+-----+-----+-----			
-----+-----+-----+-----			

FrSkySportSensorAss	ID10	0x0A00	
ASS_SPEED_DATA_ID		air speed	
500			
-----+-----+-----+-----			
-----+-----+-----+-----			

FrSkySportSensorEsc	ID17	0x0B50	
ESC_POWER_DATA_ID		ESC voltage/current	
300			
		0x0B60	
ESC_RPM_CONS_DATA_ID		RPM/consumption	
300			
		0x0B70	
ESC_TEMP_DATA_ID		temperature	
300			
		0x0E50	
ESC_SBEC_DATA_ID		SBEC voltage/current	
300			
-----+-----+-----+-----			
-----+-----+-----+-----			

FrSkySportSensorEsc	ID3	0x0200	

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

Other projects using this library

Here are references to some other projects using this library. **I do not provide support for these - in case of problems contact their authors.**

- Arduino Voltmeter compatible to FrSky SPort Telemetry with minimal additional hardware by [Dakkaron](https://github.com/Dakkaron/MinimalSPORTVoltmeter) - <https://github.com/Dakkaron/MinimalSPORTVoltmeter>
- FrSkySportTelemetry extension for INav by [RedTroll](https://github.com/variostudio/FrSkySportTelemetry) - <https://github.com/variostudio/FrSkySportTelemetry>
- Arduino S.Port to MAVLink Converter by [Closus](https://github.com/davwys/arduino-sport-to-mavlink) - <https://github.com/davwys/arduino-sport-to-mavlink>
- DIY FrSky telemetry multi-sensor by [Axel Brinkeby](http://axelsdiy.brinkeby.se/?p=2009) - <http://axelsdiy.brinkeby.se/?p=2009>

FrSky S.Port Telemetry library changelog

Version 20210509

[NEW] Added Gas Suite sensor

[NEW] Simplified send function in sensors

[FIX] Removed unnecessary semicolon from GPS_DATE_TIME_DATA_PERIOD define

Version 20210110

[FIX] Fixed unnecessary switching of TX/RX mode when serial with EVTINV is used

Version 20210108

[FIX] Fixed detecting empty frame in decoder

[NEW] Added support for non-inverted two wire serial (when used e.g. with external S.Port inverter or devices with regular serial S.Port telemetry output such as Jumper R8 receiver or Pixhawk)

[NEW] Added support for Teensy 4.1

[FIX] Refactored the FrSkySportSingleWireSerial code to make it more readable

[FIX] Minor editorial corrections

Version 20200503

[NEW] Added FrSkySportPollingDynamic polling class that polls active sensors more often and FrSkySportPollingSimple that polls sensors one by one in loop

[NEW] Updated examples to use the newly added polling classes

[NEW] Telemetry send method now returns the last sent APP ID

[NEW] Added support for ATmega2560 based boards (Arduino Mega)

[FIX] Fixed some typos

[FIX] Fixed cell voltage detection for in FrSkySportSensorXjt

[FIX] Replaced deprecated boolean type with bool in FrSkySportDecoder

[FIX] Added missing SOFT_SERIAL_PIN_Dx and SENSOR_NO_DATA_ID literals to keywords.txt

[FIX] Changed decoder state #define CRC to CRC_BYTE for consistency

[FIX] Added #define for empty frame header for better readability

Version 20200112

[NEW] Added ESC sensor and updated examples

[NEW] Added library.properties file

[NEW] Added alternative Dxx pin naming for ESP8266 as printed on some of the boards

[FIX] Limited allowed ESP8266 pins to those actually supported by the SoftwareSerial library and not used for strapping or HW serial pins

Version 20191110

[FIX] Fixed incorrect detection of stuffed bytes in FrSkySportDecoder that caused some data not decoded properly

Version 20191103

[NEW] Added support for Teensy 4.0 boards

[FIX] Slightly rearranged the serial port configuration and using defines now rather than magic numbers for clarity

[FIX] Simplified Teensy boards detection

Version 20190824

[NEW] Added support for ESP8266 boards

Version 20180402

[NEW] Added sending the RxBatt data when polling is enabled

[NEW] Added method for setting the RSSI and RxBatt to the FrSkySportTelemetry class (effective only when polling is enabled)

[NEW] Updated the FrSkySportTelemetryExample

[FIX] Minor editorial corrections

Version 20171114

[FIX] Removed unnecessary debug message from FrSkySportDecoder.cpp

Version 20160919

[NEW] Added support for Teensy LC, 3.5 and 3.6 boards and updated examples

[NEW] Added note about MLVSS sensor in the FLVSS sensor class

Version 20160818

[NEW] Added simplified polling to the FrSkySportTelemetry and FrSkySportDecoder class

[NEW] Added POLLING_ENABLED define to examples to simplify switching the polling on/off

[NEW] Removed separate connection diagrams for decoder, as the connection is now the same for both emulator and decoder classes

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 the S.Port mirror when it receives old FrSky Telemetry Hub data)

[NEW] Added CRC checking in decoder class

[NEW] Added return result (decoded appld) 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)

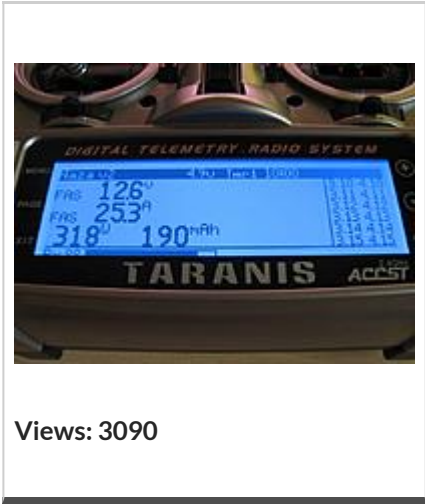
[NEW] added connection diagrams

Version 20140914

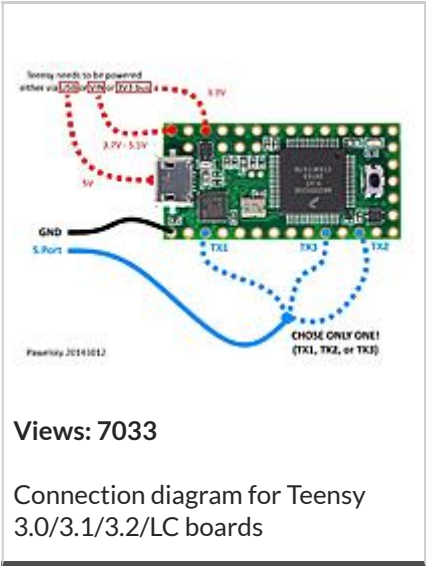
initial version of the library

Images

[View all Images in thread](#)

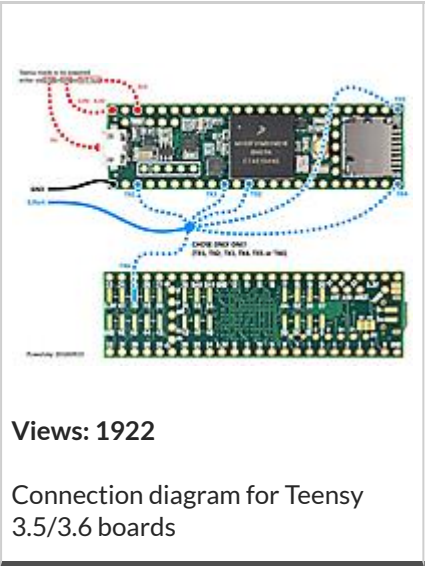


Views: 3090



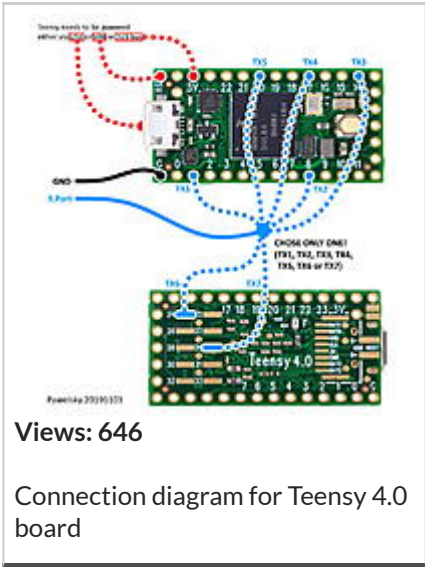
Views: 7033

Connection diagram for Teensy 3.0/3.1/3.2/LC boards



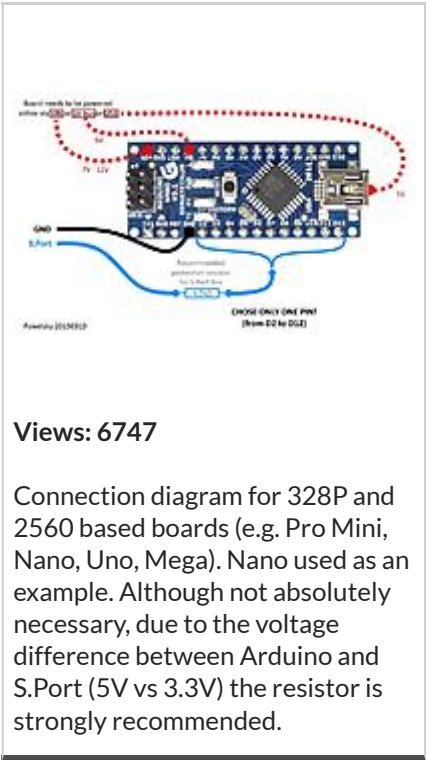
Views: 1922

Connection diagram for Teensy 3.5/3.6 boards



Views: 646

Connection diagram for Teensy 4.0 board



Views: 6747

Connection diagram for 328P and 2560 based boards (e.g. Pro Mini, Nano, Uno, Mega). Nano used as an example. Although not absolutely necessary, due to the voltage difference between Arduino and S.Port (5V vs 3.3V) the resistor is strongly recommended.

Files



View all Files in thread

FrSkySportTelemetry_20210.. v202100509 (1.44 MB) 694 views ZIP

Last edited by pawelsky; Feb 06, 2023 at 04:16 AM.

Sign up now to remove ads between posts

Receiver Battery V/A/mAh Telemetry, w/Wireless Switch



Sep 14, 2014

#2