[NAME TODO]

Paul Nykiel

August 13, 2017

Contents

1 Introduction

Fast and extensible communication over various serial-interfaces. Primarily designed for radio-controlled models, mesh capable.

2 Example Application

Long Range: Multiple Transmitters, Drone as Repeater. TCP for Transmitters.

3 Protocol

The protocol is designed for serial interfaces, the data is binary encoded to optimize for slow connections.

3.1 Timing

A new package should be transmitted by the sender at a rate of 50Hz (every 20ms) which is the rate a normal Servo or ESC gets signals at. The minimum time between each package should be 10ms to allow every device to parse the data.

3.2 Mesh behavior

Flag, Routing Timeout, UID

3.3 Package structure

The protocol is packaged based, every package consists of four major parts:

- Header
- Configuration
- Data
- Footer

3.3.1 Header

The header consists of three bytes:

- Start byte
- Unique-ID
- Transmitter-ID

Start byte The start byte is one byte which is always *0xC9*.

Unique-ID The unique-ID is generated by the initial sender. It is used to identify a message, especially in a mesh-application.

Algorithm for generation of the UID The easiest way of generation is having a counter shared by all devices, every time a device receives or transmits a package the counter gets incremented. If the upper limit of a byte (255) is reached the counter overflows, this implementation guarantees maximum time between reoccurring UIDs.

Transmitter-ID A persistent ID which should be unique to the transmitter. Optimally it should be easily configurable by the user, for example via a configuration file, a settings menu or DIP-Switches on the module.

3.3.2 Configuration

The configuration consists of at least one byte.

First configuration byte (general) The first byte consists of the following bits:

- Bit 0-2: Resolution
- Bit 3-5: Channel count
- Bit 6: Error Bit
- Bit 7: Following

Resolution Three bytes containing the resolution of each channel transmitted, possible values are:

Value	Resolution	Bit per Channel
0b000	32	5
0b001	64	6
0b010	128	7
0b011	256	8
0b100	512	9
0b101	1024	10
0b110	2048	11
0b111	4096	12

Channel count Three bytes containing the amount of channels used, possible values are:

Value	Channels
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	256

Error If a device registers an error (checksum, timeout during package transmission) this flag is set for the next 4 packages send by this device.

Following This byte is a flag whether the following byte is still a configuration byte and not a data byte.

Second configuration byte (mesh) The second byte consists of the following bits: Bit 0: Mesh-Message Bit 1-4: Routing Length Bit 5-6: unused Bit 7: Following

Mesh-Message A flag whether the message is intended as a Mesh-Message, this flag defaults to false if the following flag in the first configuration byte is not set.

Routing Length A number counting the amounts of nodes this message has been passed by. The initial sender sets the value and every node decrements the value. If the value reaches zero, the package doesn't get forwarded.

Following This byte is a flag whether the following byte is still a configuration byte and not a data byte. At the moment this byte needs to be false.

3.3.3 Data

The size (in bytes) of the data section is determined by the resolution and the channel count:

*ceil((resolution * Channel count)/8)*

The data is just queued together so the first r (with r being the channel resolution) bits are the first channel, bits r+1 to r*2 are the second until the last channel. If there are unused bits left at the end they are set to 0.
0 equals don't care

3.3.4 Footer

The footer consists of two bytes Checksum End byte

Checksum For the checksum think of the transmitted data as list starting with the unique-ID and ending with the last data byte. The checksum is once calculated by the sender and once by the recipient to verify the data is still correct.

Algorithm Now take every but the last byte and do a XOR Operation with the next byte. Every operation returns a new byte, so your list which was initially n bytes long now is only n-1 bytes long. Repeat this Process until the list is only one byte long, this byte is your checksum.

End byte The end byte is one byte which is always *0x93*.