# RC Control

## Choice of the RC – Controller and the receiver

The XCopter is designed to be controlled by a conventional RC Transmitter.  
A more specific explanation why the Graupner MX-16 HoTT was chosen, is listed below.

### The requirements for the RC- Controller and the receiver are:

* Support of sum signal (PPM)
* Provide 4 channels or more, 8 channels are optimal
* The costs have to be less than 350€
* Easy configuration of the RC- Controller

### A selection of leading companies producing RC- Controllers:

1. Graupner
2. Futaba
3. Spektrum
4. Modelcraft

### Reasons for Graupner:

* Graupner is an innovative and leading company in RC- modelling
* Graupner ensures a high quality standart
* Graupner provides lots of datasheets for each product
* Graupner has a big RC- community

### Major properties of the RC- Controller:

* 8 channel
* HoTT technology (Support for Telemetry data, sum signal, transmit up to 16 channels)
* Bidirectional communication between transmitter and receiver
* Free configurable switches
* Signal range 4 km
* Very fast rebinding

## RC Transmitter Settings

This section is about how the Graupner HOTT Receiver has to be set up.  
Every option listed below is a translated version of the option entry, since the Language of the Transmitter is german. The german option names are singed as [Name].  
The XCopter Profile Settings on the MX-16 Transmitter are:

### Basic model settings [Grundeinst]:

|  |  |
| --- | --- |
| Control mode [Steueranord] | 2 |
| throttle stick behaviour [Motor an K1] | "kein" (default) |
| changing Flight Phase with Channel8 delayed[K8 Verzögert] | "nein" (default) |
| Tail unit type [Leitwerk] | "normal" (default) |
| Tail unit servo type[Querr./Wölb] | "1QR" (default) |
| Channel Output [Empf. Ausg] | S 2 -> [Ausgang] 1 S 3 -> [Ausgang] 2 S 1 -> [Ausgang] 3 |

### RC Receiver Settings [Telemetrie]

|  |  |
| --- | --- |
| Receiver output type [CH OUT TYPE] | "SUMD HD 08" |

More information is available in the original Graupner MX-16 user manual (Ref http://www.graupner.de/fileadmin/downloadcenter/anleitungen/20070222092922\_4701\_GB.pdf)  
Graupner HoTT can deliver a digital SUM-Signal (SUMD) on most common HOTT Receivers.   
The Signal can be found on the following Outputs:

GR-12L -> Output #6  
GR-16 -> Output #8  
GR-24 -> Output #8  
GR-32 -> Output #8

## Graupner HoTT-SUMD Protocol

The “GR-16” receiver supports two different sum signals, “SUMO”- and “SUMD”- Signal.  
The SUMO- Signal is an analogue sum signal and is equal to a pulse position modulation (PPM) whereas the SUMD- Signal is a digital UART protocol. Similar to a Spektrum Satellite receiver.

### Definition

HoTT SUMD is implemented with as a serial connection. The data stream is generated by HoTT receivers. The transmitter generates data frames at a data rate of 100Hz (10ms). Each data frame consists of a header followed by a data section representing the channel data and is concluded by a CRC checksum.  
Each data frame is sent as a consistent data burst leaving minimal gaps less than 50µs between transmitted data bytes. The serial connection has to be setup whit the following:

* 115200 Bit/s baud rate
* 8 Databits
* no Paritybit
* 1 Stopbit

### Structure of a HoTT- SUMD frame

A single SUMD data frame consists of three consecutive sections:  
SUMD\_Header, SUMD\_Data, SUMD\_CRC.

The SUMD\_Data section contains the channel data in sequential order. The number of channels to be transmitted can be up to 32. Each channel data is represented by an unsigned 16 bit word.

### SUMD\_Header section description

|  |  |  |
| --- | --- | --- |
| ***Byte*** | ***Byte\_Name*** | ***Byte\_Value*** |
| Byte 0 | Vendor\_ID | 0xA8 |
| Byte 1 | Status | 0x01 or 0x81 |
| Byte 2 | Number of channels |  |

### SUMD Data section description

Byte n\*2+1 High Byte of channel n

Byte n\*2+2 Low Byte of channel n

### SUMD\_CRC section description

Byte (N\_Channels+1 )\*2+1 High Byte of CRC

Byte (N\_Channels+1 )\*2+2 Low Byte of CRC derived

### Channel data interpretation

Each channel data is represented by an unsigned 16 Bit Word. The data range is derived from the pulse length for standard servos.

|  |  |  |
| --- | --- | --- |
| Stick Position | Channel Data | Remark |
| Extended low position (-150%) | 0x1c20 | Equivalent to 900µs length |
| Low position (-100%) | 0x2260 | Equivalent to 1100µs length |
| Neutral position (0%) | 0x2ee0 | Equivalent to 1500µs length |
| High position (100%) | 0x3b60 | Equivalent to 1900µs length |
| Extended high position (150%) | 0x41a0 | Equivalent to 2100µs length |

### Implementation of the SUMD Parsing

SUMD is a serial format and can be read directly from the receiver that’s connected via UART.  
Luckily Altera is offering an RS232 UART IP Core, which can be added to our SoPC using Qsys.  
It only requires two additional GPIO Pins, for receiving or transmitting serial data.   
Reading and controlling the UART will be part of the UART driver.  
The UART has to be initiated with the following settings, to receive a SUMD-Frame:

- 115200 Baud

- No Parity

- 1 Stop Bit

Every received Byte has to be interpreted according to the definitions of the SUMD signal format, which is described in the previous section.  
Following Steps are executed by the RC interpreted Controller:

### Saving raw SUMD-Frame Bytes from the UART

The SUMD-Controller has to wait for a new SUMD-Frame. A frame starts if the value of a received Byte equals the VendorID. After that, the following Bytes will be saved in an Array.  
The size of the Array will be equal to the frame this can be calculated with:  
SUMD-Frame length = SUMD Header length + Number of Channels \* 2 + CRC length)

### Interpreting the received SUMD-Frame

According to the SUMD format description, every Byte has its own specific purpose.  
The actual received RC-commands are sliced into a High Byte and a Low Byte,   
thus it is necessary to append both Bytes to a 16 Bit Integer. Every Channel value can be stored in an Array, which can be accessible in a c struct including all additional Data of the SUMD-Frame.

# UART Driver

This driver will offer functions to initiate and read the RS232 UART IP Core.  
It is also possible to check if a new Byte was read. Which is convenient for framing a serial protocol. The driver is divided in a source file "Drivers/Driver\_UART.c " and a   
header file "Drivers/Driver\_UART.h".

### SUMD-Frame-high

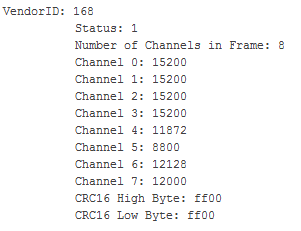


Figure 8 SUMD-Frame-high

### SUMD-Frame-low

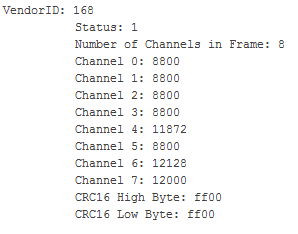


Figure 9 SUMD-Frame-low