CRSF Protocol

06. Dec. 2021 Rev 10

Table of content

Features

- Low latency high update rate for RC signals between RC - XF and XF - FC
- Bidirectional communication
- Share telemetry from flying platform to the RC
- Edit configuration for direct connected devices and remotely connected devices (RC can configure FC or OSD over crossfire)
- Share receiver serial number to TX so it can be matched to model memory.

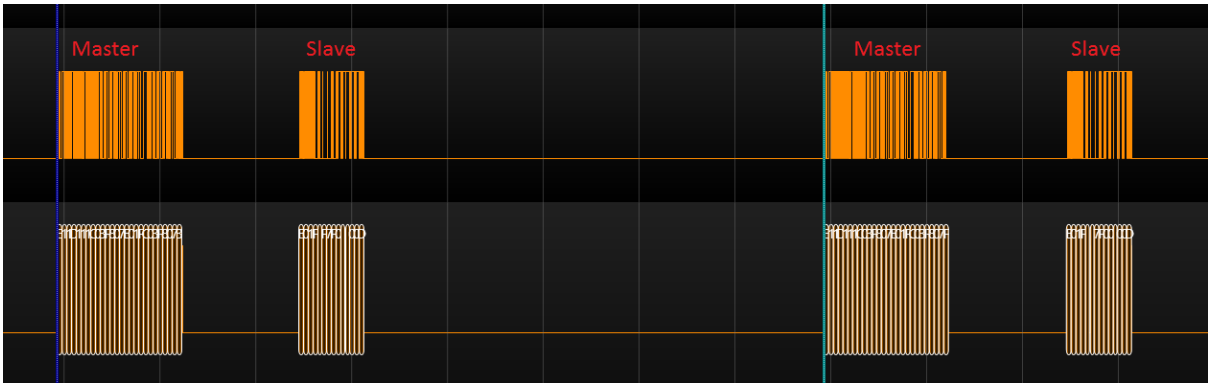
Future features:

- System should be able to handle multiple of the same sensors using sensor ID’s

Hardware

Single wire half duplex UART

This configuration is usually used between RC and Crossfire TX. XF TX support both inverted and non-inverted UART. The RC acts as master in this case and the XF TX can only send telemetry if it's synchronized to the RC frames sent by the RC. The RC frame update rate should not be sent more often than every 4ms. The UART runs at 416666baud 8N1 (inverted or non-inverted) at 3.3V level.



Dual wire / full duplex UART

This configuration is usually used on the flying platform side. Two devices are connected by regular UART connection. Only non-inverted (regular) UART is supported in this configuration. The UART runs at 416666baud 8N1 at 3.0 to 3.3V level.

Frame

Structure

The basic structure for each frame is the same. There is a range of Types with an extended header which will have the first few bytes of payload standardized. This is required to route frame across multiple devices for point to point communication. Each CRSF frame is **not longer than 64 bytes** (including the Sync and CRC bytes).

Broadcast Frames:
<Device address or Sync Byte> <Frame length> <Type><Payload> <CRC>

Extended header frames:
<Device address or Sync Byte> <Frame length> <Type><Destination Address> <Origin Address> <Payload> <CRC>

Device address or Sync Byte:	(uint8_t) Device address for I2C or Sync byte serial connections.In case of I2C (BST) this is mostly “Broadcast address” or defined by Router.
Frame length:	Amount of bytes including Type, Destination and Origin address for Extended header frames frames,
Payload and	CRC (uint8_t). Valid range is between 2 and 62. If this uint8 value is out of valid range,the frame must be discarded. Basically, It’s entire frame size - 2.
Type:	Frame type (uint8_t)
CRC:	8 Bit CRC of the frame. See CRC (uint8_t)
Sync Byte:	0xC8
Endianness	Big endian (MSB)

Device addresses

0x00	Broadcast address
0x0E	Cloud (a.k.a. MQTT broker)
0x10	USB Device
0x12	Bluetooth Module/WIFI
0x14	Video Receiver
0x20-0x7F	Dynamic address space for NAT
0x80	OSD / TBS CORE PNP PRO
0x8A	Reserved
0xB0	Crossfire TX integration test terminal
0xB2	Crossfire RX integration test terminal

0xC0	Current Sensor / PNP PRO digital current sensor
0xC2	GPS / PNP PRO GPS
0xC4	TBS Blackbox
0xC8	Flight controller
0xCA	Reserved
0xCC	Race tag
0xCE	VTX
0xEA	Remote Control
0xEC	R/C Receiver / Crossfire Rx
0xEE	R/C Transmitter Module / Crossfire Tx
0xF0	PPG Controller
0xF2	PPG Mainboard

CRC

CRC includes Type and Payload of each frame.

Code example:

/* CRC8 implementation with polynom = $x^7+ x^6+ x^4+ x^2+ x^0$ (0xD5) */
unsigned char crc8tab[256] = {
0x00, 0xD5, 0x7F, 0xAA, 0xFE, 0x2B, 0x81, 0x54, 0x29, 0xFC, 0x56, 0x83, 0xD7, 0x02, 0xA8, 0x7D,
0x52, 0x87, 0x2D, 0xF8, 0xAC, 0x79, 0xD3, 0x06, 0x7B, 0xAE, 0x04, 0xD1, 0x85, 0x50, 0xFA, 0x2F,
0xA4, 0x71, 0xDB, 0x0E, 0x5A, 0x8F, 0x25, 0xF0, 0x8D, 0x58, 0xF2, 0x27, 0x73, 0xA6, 0x0C, 0xD9,
0xF6, 0x23, 0x89, 0x5C, 0x08, 0xDD, 0x77, 0xA2, 0xDF, 0x0A, 0xA0, 0x75, 0x21, 0xF4, 0x5E, 0x8B,
0x9D, 0x48, 0xE2, 0x37, 0x63, 0xB6, 0x1C, 0xC9, 0xB4, 0x61, 0xCB, 0x1E, 0x4A, 0x9F, 0x35, 0xE0,
0xCF, 0x1A, 0xB0, 0x65, 0x31, 0xE4, 0x4E, 0x9B, 0xE6, 0x33, 0x99, 0x4C, 0x18, 0xCD, 0x67, 0xB2,
0x39, 0xEC, 0x46, 0x93, 0xC7, 0x12, 0xB8, 0x6D, 0x10, 0xC5, 0x6F, 0xBA, 0xEE, 0x3B, 0x91, 0x44,
0x6B, 0xBE, 0x14, 0xC1, 0x95, 0x40, 0xEA, 0x3F, 0x42, 0x97, 0x3D, 0xE8, 0xBC, 0x69, 0xC3, 0x16,
0xEF, 0x3A, 0x90, 0x45, 0x11, 0xC4, 0x6E, 0xBB, 0xC6, 0x13, 0xB9, 0x6C, 0x38, 0xED, 0x47, 0x92,
0xBD, 0x68, 0xC2, 0x17, 0x43, 0x96, 0x3C, 0xE9, 0x94, 0x41, 0xEB, 0x3E, 0x6A, 0xBF, 0x15, 0xC0,
0x4B, 0x9E, 0x34, 0xE1, 0xB5, 0x60, 0xCA, 0x1F, 0x62, 0xB7, 0x1D, 0xC8, 0x9C, 0x49, 0xE3, 0x36,
0x19, 0xCC, 0x66, 0xB3, 0xE7, 0x32, 0x98, 0x4D, 0x30, 0xE5, 0x4F, 0x9A, 0xCE, 0x1B, 0xB1, 0x64,
0x72, 0xA7, 0x0D, 0xD8, 0x8C, 0x59, 0xF3, 0x26, 0x5B, 0x8E, 0x24, 0xF1, 0xA5, 0x70, 0xDA, 0x0F,
0x20, 0xF5, 0x5F, 0x8A, 0xDE, 0x0B, 0xA1, 0x74, 0x09, 0xDC, 0x76, 0xA3, 0xF7, 0x22, 0x88, 0x5D,
0xD6, 0x03, 0xA9, 0x7C, 0x28, 0xFD, 0x57, 0x82, 0xFF, 0x2A, 0x80, 0x55, 0x01, 0xD4, 0x7E, 0xAB,
0x84, 0x51, 0xFB, 0x2E, 0x7A, 0xAF, 0x05, 0xD0, 0xAD, 0x78, 0xD2, 0x07, 0x53, 0x86, 0x2C, 0xF9};

```
uint8_t crc8(const uint8_t * ptr, uint8_t len)
{
  uint8_t crc = 0;
  for (uint8_t i=0; i<len; i++) {
    crc = crc8tab[crc ^ *ptr++];
  }
  return crc;
}
```

Routing

If a device has more than one CRSF port it's required to forward all received frames to the other ports. CRSF works as a star network. It's forbidden to use any loop connection as it would keep forwarding the same message endlessly. This can be detected if one device receives it's own Parameter ping devices request or Heartbeat message it should file an error and tell the user.

Frames within Broadcast frames range will be forwarded to any other CRSF port.

Received frames within Extended header frames range should be parsed. The origin address should be stored in a lookup table. Each other port's lookup table should then be searched for matching the destination address. In case of a found match the frame should only be forwarded to the matching port. If there is no match the frame will be forwarded to any other CRSF port except the one the frame was received.

Frame Types

The following list shows the content of each frame type. The most common frames have an own type. Commands and others share one type.

If a value is unknown, variable max value will be sent instead.

Broadcast frames

Type range: 0x00 to 0x27. There are also some other broadcast frames (e.g. Mavlink!)

0x02 GPS

Payload:

```
typedef struct {
    int32_t latitude;               // degree / 10`000`000
    int32_t longitude;              // degree / 10`000`000
    uint16_t groundspeed;           // km/h / 100
    uint16_t heading;               // degree / 100
    uint16_t altitude;              // meter - 1000m offset
    uint8_t satellites;             // # of sats in view
} libCrsf_gps_s
```

0x03 GPS Time Frame

This frame need to be synchronized with the ublox time pulse. The maximum offset of time is +/-10ms.

Payload:

```
typedef struct {
    int16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
    uint16_t millisecond;
} libCrsf_gps_time_s
```

0x06 GPS Extended Frame

Payload:

```
typedef struct {
    uint8_t fix_type;               // Current GPS fix quality
    int16_t n_speed;                // Northward (north = positive) Speed [cm/sec]
    int16_t e_speed;                // Eastward (east  = positive) Speed [cm/sec]
    int16_t v_speed;                // Vertical    (up  = positive) Speed [cm/sec]
    int16_t h_speed_acc;            // Horizontal Speed accuracy cm/sec
    int16_t track_acc;      // Heading accuracy in degrees scaled with 1e-1 (degrees times 10)
    int16_t alt_ellipsoid;          // Meters Height above GPS Ellipsoid (not MSL)
    int16_t h_acc;                  // horizontal accuracy in cm
    int16_t v_acc;                  // vertical accuracy in cm
    uint8_t reserved;
    uint8_t hDOP;                   // Horizontal dilution of precision,Dimensionless in units of.1.
    uint8_t vDOP;                   // vertical dilution of precision, Dimensionless in units of .1.
} libCrsf_gps_extended_s;
```

0x07 Vario sensor

Payload:

```
typedef struct {
    int16_t v_speed;                // Vertical speed ( m/s * 100 )
} libCrsf_vario_s
```

0x08 Battery sensor

Payload:

```
typedef struct {
    int16_t voltage;            // Voltage ( mV * 100 )
    int16_t current;            // Current ( mA * 100 )
    int24_t capacity_used;      // uint24_t Capacity used ( mAh )
    uint8_t remaining;          // uint8_t Battery remaining ( percent )
} libCrsf_battery_s
```

0x09 Barometric Altitude sensor

Payload:

```
typedef struct {
    uint16_t altitude_packed;    // Altitude above start (calibration) point.
                                 // value depends on MSB:
                                 // MSB = 0: altitude is in decimeters - 10000dm offset.
                                 // MSB = 1: altitude is in meters. Without any offset.
} libCrsf_altitude_baro

//unpack function example:
int32_t get_altitude_dm(uint16_t packed){
    return (packed & 0x8000) ? (packed & 0x7fff)*10 : (packed - 10000);
}
// Due to OpenTX counts any 0xFFFF value as incorrect, the maximum sending value is
limited to 0xFFFE (32766 meters)
//pack function example:
uint16_t get_altitude_packed (int32_t altitude_dm){
enum {
    ALT_MIN_DM = 10000, //minimum altitude in dm
    ALT_THRESHOLD_DM = 0x8000 - ALT_MIN_DM, //altitude of precision changing in dm
    ALT_MAX_DM = 0x7ffe * 10 - 5, //maximum altitude in dm
}
    if(altitude_dm < -ALT_MIN_DM) //less than minimum altitude
        return 0; //minimum
    if(altitude_dm > ALT_MAX_DM) //more than maximum
        return 0xfffe; //maximum
    if(altitude_dm < ALT_THRESHOLD_DM) //dm-resolution range
        return altitude_dm + ALT_MIN;
    return ((altitude_dm+5) / 10) | 0x8000; //meter-resolution range
}
```

0x0B Heartbeat

Payload:

```
typedef struct {
    int16_t origin_add;     // Origin Device address
} libCrsf_heartbeat_s
```

0x0F Discontinued

0x10 Video transmitter telemetry

Payload:

```
typedef struct {
    uint8_t origin_address;
    uint8_t power;              // VTX power in dBm
    uint16_t frequency;         // VTX frequency in MHz
    uint8_t pit_mode:1;             // 0=Off, 1=On
    uint8_t pitmode_control:2;      // 0=Off, 1=On, 2=Switch, 3=Failsafe
    uint8_t pitmode_switch:4;       // 0=Ch5, 1=Ch5 Inv, … , 15=Ch12 Inv
} libCrsf_vtx_telemetry_s
```

0x14 Link statistics

Uplink is the connection from the ground to the UAV and downlink the opposite direction.

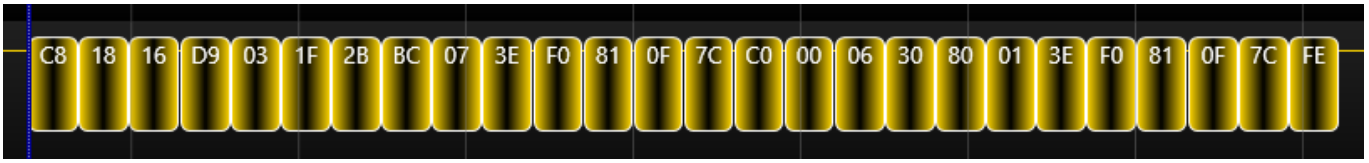
Payload:

```
typedef struct {
    uint8_t up_rssi_ant1;           // Uplink RSSI Ant. 1 ( dBm * -1 )
    uint8_t up_rssi_ant2;           // Uplink RSSI Ant. 2 ( dBm * -1 )
    uint8_t up_link_quality;        // Uplink Package success rate / Link quality ( % )
    int8_t up_snr;                  // Uplink SNR ( dB )
    uint8_t active_antenna;         // number of antenna
    uint8_t rf_profile;             // enum 4fps = 0 , 50fps, 150hz
    uint8_t up_rf_power;            // enum 0mW = 0, 10mW, 25 mW, 100 mW, 500 mW, 1000
                                       mW, 2000mW, 250mW, 50mW
    uint8_t down_rssi;              // Downlink RSSI ( dBm * -1 )
    uint8_t down_link_quality;      // Downlink Package success rate / Link quality ( % )
    int8_t down_snr;                // Downlink SNR ( dB )
} libCrsf_link_statistics_s
```

0x16 RC channels packedPayload:

- 11bits Channel 1
- 11bits Channel 2
- …
- 11bits Channel 16

16 channels packed into 22 bytes.
Center (1500us) = 992
TICKS_TO_US(x) ((x - 992) * 5 / 8 + 1500)
US_TO_TICKS(x) ((x - 1500) * 8 / 5 + 992)



0x17 Subset RC channels packed

Payload:

```
typedef struct PACKED {
    uint8_t starting_channel:5;          // which channel number is the first one in the frame
    uint8_t res_configuration:2;         // configuration for the RC data resolution (10 - 13 bits)
    uint8_t digital_switch_flag:1;       // configuration bit for digital channel
    uint16_t channel[]:res;              // variable amount of channels (with variable resolution
based on the res_configuration) based on the frame size
    uint16_t digital_switch_channel[]:10;    // digital switch channel
} libCrsf_subset_rc_channels_packed_s
```

Calculation example to convert rc values to channel values
Unpacking on TX side: ((x - 3750) * 8 / 25 + 993)
Unpacking on FC side: (x * S + 988) {S = 1.0 for 10-bit, S = 0.5 for 11-bit, S = 0.25 for 12-bit, S = 0.125 for 13-bit}

0x18 RC channels packed 11bits - unused

(same as 0x16, but same conversion style as 0x17)

0x19 - 0x1B Reserved Crossfire

0x1C Link statistics rx

Payload:

```
typedef struct {
    uint8_t rssi_db;            // RSSI ( dBm * -1 )
    uint8_t rssi_percent;       // RSSI in percent
    uint8_t link_quality;       // Package success rate / Link quality ( % )
    int8_t snr;                 // SNR ( dB )
    uint8_t rf_power_db;        // rf power in dBm
} libCrsf_link_statistics_rx_s
```

0x1D Link statistics tx

Payload:

```
typedef struct {
    uint8_t rssi_db;            // RSSI ( dBm * -1 )
    uint8_t rssi_percent;       // RSSI in percent
    uint8_t link_quality;       // Package success rate / Link quality ( % )
    int8_t snr;                 // SNR ( dB )
    uint8_t rf_power_db;        // rf power in dBm
    uint8_t fps;                // rf frames per second (fps / 10)
} libCrsf_link_statistics_tx_s
```

0x1E Attitude

Payload:

```
typedef struct {
    int16_t pitch;          // Pitch angle ( rad / 10000 )
    int16_t roll;           // Roll angle ( rad / 10000 )
    int16_t yaw;            // Yaw angle ( rad / 10000 )
} libCrsf_attitude_s
//WARNING: Angle values must be in -180° +180° range!
```

0x1F MAVLink FC

LIBCRSF_BF_MAVLINK_FC_ADD_DATA

Payload:

```
typedef struct {
    int16_t airspeed;
    uint8_t base_mode;        // vehicle mode flags, defined in MAV_MODE_FLAG enum
    uint32_t custom_mode;  // autopilot-specific flags
    uint8_t autopilot_type;    // FC type; defined in MAV_AUTOPILOT enum
    uint8_t firmware_type;    // vehicle type; defined in MAV_TYPE enum
}
```

0x21 Flight mode text based

Payload:

- char[] Flight mode (Null-terminated string)

0x27 Production

Reserved for production use, payload format defined in each product itself.

0x78 - 0x79 KISS FC

Reserved range

0x7A - 0x7F Betaflight MSP

-

0x7A MSP_Request

Description:

- CRSF frame which wraps MSP request (‘$M<’ or ‘$X<’)
- Supported by Betaflight devices
- Supported devices will respond with 0x7B

0x7B MSP_Response

Description

- CRSF frame which wraps MSP response (‘$M>’,’$X>’,‘$M!’,’$X!’)
- Supported by Betaflight devices
- Supported device will send this frame in response of MSP_Request (0x7A)

MSP frame over CRSF Payload packing:

- MSP frame is stripped from header ($ + M/X + </>/!) and CRC
- Resulted MSP-body might be divided in chunks if it doesn't fit in one CRSF-frame.
- A ‘Status’ byte is put before MSP-body in each CRSF-frame.
- Status byte consists of three parts:
  bits 0-3 represent the sequence number of the CRSF frame,
  bit 4 checks if current MSP chunk is the beginning (or only) of a new frame (1 if true),
  bits 5-6 represent the version number of MSP protocol (1 or 2 currently),
  bit 7 represents an error (for response only).
- Chunk size of the MSP-body is calculated from size of CRSF frame. But size of the MSP-body must be parsed from the MSP-body itself (with respect to MSP version and Jumbo-frame).
- The last/only CRSF-frame might be longer than needed. In such a case, the extra bytes must be ignored.
- Maximum chunk size is defined by maximum length of CRSF frame 64 bytes, therefore, maximum MSP-chunk length is 57 bytes. Minimum chunk length might by anythig, but the first chunk must consist of size and function ID (i.e., 5 bytes for MSPv2).
- CRC of the MSP frame is not sent due to it’s already protected by CRC of CRSF. If CRC is needed, it should be calculated at the receiving part.
- MSP-response must be sent to the origin of the MSP-request. It means that <destination> and <origin> bytes of CRSF-header in response must be the same as in request but swapped.

0x80 Ardupilot

Reserved range

0xAA CRSF MAVLink envelope

Description:

- CRSF MAVLink envelope is designed to transfer MAVLink protocol over CRSF routers, it supports both MAVLink2 and MAVLink1 frames. Since MAVLink frames are generally much longer than CRSF frames (281 bytes for MAVLink2 vs 64 bytes for CRSF), MAVLink frames will be broken up into chunks and transferred by encapsulating it with CRSF frames.
- Note that encoding / decoding correct chunk count while writing / reading MAVLink envelopes should be handled by the user to ensure data integrity.

Data Struct:

```
#define LIBCRSF_MAV_ENV_DATA_MAX_LEN    58  // max payload size
#define LIBCRSF_MAV_ENV_CHUNK_MAX       4     // max chunks for MAVLink

typedef struct {
    uint8_t mav_env_total_chunk : 4;  // total number of chunks (max 4)
    uint8_t mav_env_curr_chunk : 4;   // current chunk count (max 4)
    uint8_t mav_env_data_size;          // size of data
    uint8_t mav_env_data[LIBCRSF_MAV_ENV_DATA_MAX_LEN]; // data array
} libCrsf_mav_env_data_s;
```

Frame Structure:

<Sync><Len><Type(0xAA)><totalChunk(bit3 - 7):currChunk(bit0-3)><dataSize>
<dataStart>...<dataEnd><CRC>

0xAC CRSF_MAV_SYS_STATUS_SENSOR

Description:

- CRSF frame for packing info of MAVLink enabled flight controller’s sensor status
- To decode data packed within the frame, Please refer to https://mavlink.io/en/messages/common.html#MAV_SYS_STATUS_SENSOR

Data Struct:

```
typedef struct {
    uint32_t sensor_present;
    uint32_t sensor_enabled;
    uint32_t sensor_health;
} libCrsf_mav_sys_status_sensor_s;
```