Brief Introduction to CRSF V3 on FC side

We are going to add some features in the CRSF V3 protocol. There are two of them related to the flight controller: higher baud rate for the CRSF UART port & dynamically packed RC frames. The ideas are explained and example code for the changes are given with respect to betaflight implementation.

1. Faster Baud Rate Support

Once a normal communication channel is established using CRSF V2 (the current protocol), the XF RX will send a new CRSF frame with a new baud rate speed proposal.
If the FC agrees to use that proposed baud rate, the FC should send the "accept" reply. then both the RX & FC should switch to the new baud rate ASAP.
If the FC finds that the proposed baud rate is not supported, simply send the "reject" reply. If the XF RX gets the "reject" reply, it will try to propose another slower baud rate until either all the baud rate requests have been rejected or "accept" reply is received.

For the XF RX we are going to support 2 more baud rate settings in CRSF V3: 2Mbps and 1Mpbs. The first proposal will be 2Mbps. If it’s rejected then we will send the second proposal with 1Mbps. Currently we decided to support these 2 settings. CRSF V3 will only be running if either one of the baud rates is accepted.

Examples

A. 2Mbps baud rate proposal sent from the RX to FC

C8 0C 32 C8 EC 0A 70 01 00 1E 84 80 22 72
C8: sync byte
0C: frame length
32: command type
C8: recipient address (address of FC)
EC: sender address (address of RX)
0A: general command type
70: baud rate proposal
01: port id
00 1e 84 80: the proposed baud rate to be used in hex format (0x001e8480=2000000Bps)
22: command frame CRC byte with CRC8 & polynomial 0xBA for all command payloads (from recipient address to the 4 bytes baud rate)
72: CRSF frame CRC byte with CRC8 & polynomial 0xD5

B. Accept reply sent from the FC to TX

C8 09 32 EC C8 0A 71 01 01 5E A0
C8: sync byte
09: frame length
32: command type
EC: recipient address (address of RX)
C8: sender address (address of FC)
0A: general command type
71: baud rate proposal response
01: port id

01: reply (01 indicates “accepted” or 00 for “rejected”)
5E: command frame CRC byte with CRC8 & polynomial 0xBA for all command payloads (from recipient address to the 4 bytes baud rate)
A0: CRSF frame CRC byte with CRC8 & polynomial 0xD5

C. extend CRC8 APIs to manipulate the command CRC in the command frame 0x32

Original CRC8 APIs implementation can be found on https://github.com/betaflight/betaflight/blob/master/src/main/common/crc.c#L62-L94.

We made some changes on extending the APIs to handle different polynomials.

```
uint8_t crc8_calc(uint8_t crc, unsigned char a, uint8_t poly)
{
   crc ^= a;
   for (int ii = 0; ii < 8; ++ii) {
       if (crc & 0x80) {
           crc = (crc << 1) ^ poly;
       } else {
           crc = crc << 1;
       }
   }
   return crc;
}

uint8_t crc8_update(uint8_t crc, const void *data, uint32_t length,
uint8_t poly)
{
   const uint8_t *p = (const uint8_t *)data;
   const uint8_t *pend = p + length;

   for (; p != pend; p++) {
       crc = crc8_calc(crc, *p, poly);
   }
   return crc;
}

void crc8_sbuf_append(sbuf_t *dst, uint8_t *start, uint8_t poly)
{
   uint8_t crc = 0;
   const uint8_t * const end = dst->ptr;
   for (const uint8_t *ptr = start; ptr < end; ++ptr) {
       crc = crc8_calc(crc, *ptr, poly);
   }
   sbufWriteU8(dst, crc);
}
```

Of course there would be changes in the crc.h https://github.com/betaflight/betaflight/blob/master/src/main/common/crc.h#L30-L32.

```
uint8_t crc8_calc(uint8_t crc, unsigned char a, uint8_t poly);
uint8_t crc8_update(uint8_t crc, const void *data, uint32_t length,
uint8_t poly);
void crc8_sbuf_append(struct sbuf_s *dst, uint8_t *start, uint8_t
poly);
#define crc8_dvb_s2(crc, a)                          crc8_calc(crc, a,
0xD5)
#define crc8_dvb_s2_update(crc, data, length)        crc8_update(crc,
data, length, 0xD5)
#define crc8_dvb_s2_sbuf_append(dst, start)
crc8_sbuf_append(dst, start, 0xD5)
#define crc8_poly_0xba(crc, a)                       crc8_calc(crc, a,
0xBA)
#define crc8_poly_0xba_sbuf_append(dst, start)
crc8_sbuf_append(dst, start, 0xBA)
```

2. Faster Dynamically Packed RC Frame 0x17

This dynamically packed RC frame will be used if CRSF V3 is enabled. The number of RC channels will vary from 1 to 32 on this frame. The structure is shown below:

```
typedef struct PACKED {
    uint8_t starting_channel:5;    // which channel number is the first one in the frame
    uint16_t no0:11;
    uint16_t no1:11;
    uint16_t no2:11;
    uint16_t no3:11;
    ...
    uint16_t no30:11;
    uint16_t no31:11;
} rc_channels_packed_dynamic_s
```

Below shows the conversion formula:

TICKS_TO_US(x)	((x - 992) * 5 / 8 + 1500)
US_TO_TICKS(x)	((x - 1500) * 8 / 5 + 992)

The length of the payload of RC data can be calculated using this formula: (5 + 11 * N) / 8. So the length of payload with just 1 channel = (5 + 11) / 8 = 2 bytes while it's (5 + 11 * 32) / 8 = 45 bytes for 32 channels.

The channel data **no0** holds the first channel data counting from the **starting_channel**. For example if we going to pack 4 channels with **start_channel** = 5, that means we will have this mapping:

US_TO_TICKS(channel 5) --> **no0**
US_TO_TICKS(channel 6) --> **no1**
US_TO_TICKS(channel 7) --> **no2**
US_TO_TICKS(channel 8) --> **no3**

Example code for scaling the received channel data

The actual RC data can be converted from the received channel data in this way:
Channel 5 = **TICKS_TO_US**(**no0**). This unpack conversion is actually used in CRSF V2 as well. In existing betaflight code (can be found on https://github.com/betaflight/betaflight/blob/master/src/main/rx/crsf.c#L348-L360) they use a similar approach by direct scaling the data. We now present the suggested way to do it.

```
STATIC_UNIT_TESTED uint16_t crsfReadRawRC(const rxRuntimeState_t
*rxRuntimeState, uint8_t chan)
{
    UNUSED(rxRuntimeState);
    return (crsfChannelData[chan] - 992) * 5 / 8 + 1500;
}
```

Example code for unpacking the channel data

This part is extracted from the modification made on the betaflight source code (can be found on https://github.com/betaflight/betaflight/blob/master/src/main/rx/crsf.c#L315-L346):

```
 #define CRSFV3_MAX_CHANNEL      32
#define CRSF_DYNAMIC_RC_CHANNELS_PACKED_RESOLUTION  11
#define CRSF_DYNAMIC_RC_CHANNELS_PACKED_MASK  0x07FF
#define CRSF_DYNAMIC_RC_CHANNELS_PACKED_STARTING_CHANNEL_RESOLUTION 5
#define CRSF_DYNAMIC_RC_CHANNELS_PACKED_STARTING_CHANNEL_MASK  0x1F

STATIC_UNIT_TESTED uint8_t crsfFrameStatus(rxRuntimeState_t
*rxRuntimeState){
   UNUSED(rxRuntimeState);
#if defined(USE_CRSF_LINK_STATISTICS)
   crsfCheckRssi(micros());
#endif
   if (crsfFrameDone) {
       crsfFrameDone = false;
       // unpack the RC channels
       if( crsfChannelDataFrame.frame.type ==
CRSF_FRAMETYPE_RC_CHANNELS_PACKED ) {
           // use ordinary RC frame structure (0x16)
           const crsfPayloadRcChannelsPacked_t* const rcChannels =
(crsfPayloadRcChannelsPacked_t*)&crsfChannelDataFrame.frame.payload;
           crsfChannelData[0] = rcChannels->chan0;
           crsfChannelData[1] = rcChannels->chan1;
           crsfChannelData[2] = rcChannels->chan2;
           crsfChannelData[3] = rcChannels->chan3;
           crsfChannelData[4] = rcChannels->chan4;
           crsfChannelData[5] = rcChannels->chan5;
           crsfChannelData[6] = rcChannels->chan6;
           crsfChannelData[7] = rcChannels->chan7;
           crsfChannelData[8] = rcChannels->chan8;
           crsfChannelData[9] = rcChannels->chan9;
           crsfChannelData[10] = rcChannels->chan10;
           crsfChannelData[11] = rcChannels->chan11;
           crsfChannelData[12] = rcChannels->chan12;
           crsfChannelData[13] = rcChannels->chan13;
           crsfChannelData[14] = rcChannels->chan14;
           crsfChannelData[15] = rcChannels->chan15;
       }
       else {
           // use dynamic RC frame structure (0x17)
           uint8_t n;
           uint8_t readByte;
```

```c
            uint8_t readByteIndex = 0;
            uint8_t bitsMerged = 0;
            uint32_t readValue = 0;
            uint8_t startChannel = 0xFF;
            const uint8_t *payload = crsfChannelDataFrame.frame.payload;
            uint8_t numOfChannels =
((crsfChannelDataFrame.frame.frameLength - CRSF_FRAME_LENGTH_TYPE_CRC)
* 8 - CRSF_DYNAMIC_RC_CHANNELS_PACKED_STARTING_CHANNEL_RESOLUTION) /
CRSF_DYNAMIC_RC_CHANNELS_PACKED_RESOLUTION;
            for (n = 0; n < numOfChannels; n++) {
                while(bitsMerged <
CRSF_DYNAMIC_RC_CHANNELS_PACKED_RESOLUTION) {
                    readByte = payload[readByteIndex++];
                    if(startChannel == 0xFF) {
                        // get the startChannel
                        startChannel = readByte &
CRSF_DYNAMIC_RC_CHANNELS_PACKED_STARTING_CHANNEL_MASK;
                        readByte >>=
CRSF_DYNAMIC_RC_CHANNELS_PACKED_STARTING_CHANNEL_RESOLUTION;
                        readValue |= ((uint32_t) readByte) <<
bitsMerged;
                        bitsMerged += 8 -
CRSF_DYNAMIC_RC_CHANNELS_PACKED_STARTING_CHANNEL_RESOLUTION;
                    }
                    else {
                        readValue |= ((uint32_t) readByte) <<
bitsMerged;
                        bitsMerged += 8;
                    }
                }
                crsfChannelData[startChannel + n] = readValue &
CRSF_DYNAMIC_RC_CHANNELS_PACKED_MASK;
                readValue >>=
CRSF_DYNAMIC_RC_CHANNELS_PACKED_RESOLUTION;
                bitsMerged -=
CRSF_DYNAMIC_RC_CHANNELS_PACKED_RESOLUTION;
            }
        }
        return RX_FRAME_COMPLETE;
    }
    return RX_FRAME_PENDING;
}
```

3. New Telemetry Frames

A. 0x1C Link Statistics RX

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

B. 0x1D Link Statistics TX

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