digisolutions

DV-RPTR

USB / serial interface specification

for open source firmware version 1.10 to 1.69b

developer manual

22 March 2013

Table of contents

Scope

This document describes how the communication between a DV-RPTR board (with open source firmware installed) or similar and a PC or embedded computer works.

Physical connection

USB connection

The DV-RPTR uses the PID of Atmel **`(0x03EB)`** and the VID of the CDC example **`(0x2307)`**. It implements a 'communication device class' device and need no special handling like other USB serial devices. All 'physical' parameters (baudrate, stop-bits, flow-control) are ignored (don't care).

Serial connection

As 2nd possible interface a DV-RPTR has a real serial 2-wire connection (TxD, RxD; 5V tolerant CMOS level pins). This interface can be used, if no AMBE addon board is present.

The physical parameter are fixed to 115200baud 8N1, no flowcontrol.

Data- and packet Formats

All data send to or receive from a DV-RPTR are framed in packets. This packets are used to identify the beginning (time independent) and length of the payload data. Additional a 16bit checksum secures the rest on physical serial connections.

The packet structure is calles PCP2 (packet communication protocol 2). The previous version (PCP) is used on the modified Siemens C5 radios

PCP1

On Siemens C5, PCP is used for communication with the main µP via TTL-serial link (5V; 57600baud 8N1). The protocol is described in SC5BOS development manual.

The main features:

- split up into 10 ports (endpoints in C5)
- up to 256 byte payload
- CRC CCITT 16bits (ignore leading zeroes)
- short frames (3 byte length) with XOR check-byte

PCP2

The PCP2 has some improvements:

- optional split up into 8 endpoints (command-byte is now part of payload)
- up to 65535 byte payload possible
- only one kind of frame and one frame ID (byte **`'0xD0')`**
- CRC CCITT 16bits (ignore leading zeroes) same like PCP, check is optional for USB

PCP2 frame structure

Pos	Meaning
0	START-ID **`0xD0`**
1	Length lower byte (Length of payload only - offset 03 to first crc-byte)

2	Length upper byte (Length is in little endian - "PC ready")	
3	Payload first byte, should be used as CMD/Message/EP declaration	
4	Payload 2nd byte	
	...	
L1	CRC upper byte	(L1 is position Length +3)
L2	CRC lower byte	(L2 is position Length +4)

DV-RPTR communication

General

All messages are packed into a PCP2 frame and start with a command-byte, as defined below. The command-byte (**CMD**) is the first thing you must check of every message (after receiving and checking a PCP2 frame). So keep the logic in this way:

1. listen to frame-begin identifier
2. receive minimum 5 bytes
3. do a plausible check of frame-length (more than 2048 bytes are not plausible and can't handled by the DV-RPTR; less than 1 byte payload are not possible)
4. receive the complete frame
5. check CRC, if this was necessary (serial connection)
6. now we had a valid PCP2 frame received, now you can start to identify the message by reading the **CMD**.

Structure of command byte

Bits	Meaning
7	Response-Bit, its set to '1' in a reply of a "GET" cmd
6..4	Endpoint definition ("0" DFU bootloader, "1" D-Star Hotspot/Repeater, ...)
3..0	Message ID / Command / Cmd-Group: if 16 different cmds enough, use it as a ID, if not: preselector of a message-group, next byte is the ID (DFU-Bootloader and Hotspot uses a few cmds only)

Timing, gap and limit considerations

A PC program must be ensure that a PCP2 frame is transferred as a continuous block. The maximum allowed gap on a block transfer is 5ms.

The maximum burst length of one or more packets should never exceed 2048 bytes total. Enabling the CRC function results in ignoring all defective packets with a wrong CRC.

Hotspot/Repeater Messages

Commands / Messages used

ID	Message name	Type	Description
0x10	RPTR_STATUS	R/C	Get status information or set mode -> answer
0x11	RPTR_GET_VERSION	REQ	Get version information -> answer
0x12	RPTR_GET_SERIAL	REQ	Get Serial Number -> answer (32bit number)

`0x13`	RPTR_GET_CONFIG	**REQ**	Get configuration blocks -> answer
`0x14`	RPTR_SET_CONFIG	**CMD**	Update configuration block -> answer (ACK/NAK)
`0x15`	RPTR_RXPREAMBLE	**MSG**	Preamble detected message
`0x16`	RPTR_START	**MSG**	A reception starts (START-PATTERN in bit-stream)
`0x17`	RPTR_HEADER	**MSG**	A header was received (header is decoded in payload)
`0x18`	RPTR_RXSYNC	**MSG**	A reception starts on a received frame sync pattern
`0x19`	RPTR_DATA	**MSG**	Voice data
`0x1A`	RPTR_EOT	**MSG**	A STOP-PATTERN, end-of-transmission received
`0x1B`	RPTR_RXLOST	**MSG**	Sync lost (no valid reception any more)
`0x1C`	reserved		for future use
`0x1D`	reserved		for future use
`0x1F`	RPTR_SFC	**CMD**	Do a special function call

Legend:

CMD	Command to DV-RPTR, DV-RPTR answers with a ACK/NAK + optional additional data
REQ	Request data from DV-RPTR, DV-RPTRv answers with the data wanted
R / C	Request or Command, depends on additional data
MSG	Message to or from DV-RPTR to transport D-Star data

Requests

Status (RPTR_STATUS)

Command-byte:	`0x10`
Parameters:	**none**

Use this request, to get information about the current status of the DV-RPTR modem.

Request message to DV-RPTR: `0x10`

As complete PCP2 packet: `0xD0 0x01 0x00 0x10 crch crcl`

Answer from DV-RPTR:

Every answer contains the command-byte with OR-ed bit 7 as replied message ID. The first of the received data is in this case **`0x90`** (0x10 OR 0x80).

Pos	**Name**	**Bits**	**Description**
0	Message ID	8	contains always 0x90
1	state flags	16	bitset – see below
3	TX state	8	enumeration of transceiver state
4	receive-buffer-size	8	number of voicedata buffer for HF reception (default: 21)
5	transmit-buffer-size	8	number of voicedata buffer for transmission (default 12*21)
6	unsend counter	8	number of not transmitted voicedata

7	actual TX buffer index	**8**	current position of the while transmitting *(append in FW 1.11)*

Flags (Bitset):
Bit0 : Receiver enabled
Bit1 : Transmitter enabled
Bit2 : PC Watchdog enabled
Bit3 : Checksum-Calculation enabled (check packets received from PC)
Bit4 : I/O 21 Status (Jumper-Configuration)
Bit5 : I/O 23 Status (Jumper-Configuration)
Bit6 : rsvd
Bit7 : (1) physical layer not configured (C0 config see below).
Bit8 : Receiving
Bit9 : Transmitting (PTT enabled)
Bit10: PC Watchdog occured! (RX/TX disabled)
Bit11: PCP2 Checksum checked on reception

...

TX-State(8) : Enum (Disabled, TXdelay, Sync, Start, Header, Voicedata, EOT)
Receive-Buffer(8) : Number of Voice-Frame-History available (current: 21)
Transmit-Buffer (8): Number of Voice-Frame-Buffers available (current: 252)
Unsend-Frames (8) : Number of unsend Voice-Frames, if 0 the RPTR ends transmission

Version (RPTR_GET_VERSION)

Command-byte:	`0x11`
Parameters:	`none`

This command returns the 16bit BCD coded firmware version followed by a short device identification (ASCII string).

Hints

The reply can differ in length, because the length of the identification isn't fix. The device identification don't contain a length byte (Pascal short string) nor a Null termination byte (C style).

Replay-Example

```
91 01 15  "DV-RPTR R. 2013-01-01"
```

How we should the version `0x1501` interpreted and displayed?

Highest nibble:	Main Version **1**
Third nibble:	Subversion **5** (increased if the firmware became new features)
Second nibble:	Subsubversion **0** (increased if parts of the source are rewritten or optimized)
Lowest nibble:	Bugfixlevel **a** (increased with one or more bugfixes applied at the same time)

Display it like *"V1.50a"*. A zero in the last nibble means "no bugfix level" (no letter).

Serial number (RPTR_GET_SERIAL)

Command-byte:	`0x12`
Parameters:	`none`

Returns a unique serial number (32bit). This number is programmed with the bootloader into the proteced bootloader area. Use this value to identify DV-RPTER if you have more than one connected.

Hints

Not all devices contains a valid serial number... If you using the firmware w/o a bootloader (JTAG debugging) you read a Zero with this command. Handle Zero and **`0xFFFFFFFF`** not as valid numbers.

Information from a serial number

A serial number is BCD coded and contains the year (first byte), the calendar week (2nd byte) folloed by a 'D' nibble and a 3 digit BCD counter (up to 999 devices per week).

Configuration (RPTR_GET_CONFIG)

Command-byte:	`0x13`
Parameters:	none or config block ID

Reads out all configuration blocks used from DV-RPTR (hides blocks of unequipped add-ons).

Read a specified configuration block

using GET command:
Send it w/o a parameter, all implemented config blocks a transfered to the pc.
Send it with 1 byte parameter (config-block-id), only one block (or NAK if not available) transferred back.

Commands

Mode (RPTR_SET_STATUS)

Command-byte:	`0x10`
Parameters:	**`1 control byte`**

Cmd = 0x10, like GET
Param# = 1 Byte

Parameter 1 Byte (Bitset):
Bit0: Enable/Disable Receiver
Bit1: Enable/Disable Transmitter
Bit2: Enable/Disable PC Watchdog
Bit3: Enable/Disable Checksum-Calculation

-> Command returns ACK if success or NAK.

Configuration (RPTR_SET_CONFIG)

Command-byte:	`0x14`
Parameters:	list of complete config blocks

like GET_CONFIG, but to put the configuration into DV_RPTR.
You can combine all blocks together or just send one or some blocks.
-> this command returns a ACK, if configuration was accepted, otherwise NAK.

Special Function Call

Transmitting D-Star data

A valid D-Star stream contains several 1-time transmitted fields and the repeated 20ms voice-data block. For details see JARL protocol specification. The structure on HF looks like this:

Preamble (>= 32bit)	**Start-Pattern (15 bit)**	**Header with FEC (660bit)**	**AMBE-Data (72bit)**	**Slow-Data (24bit)**

AMBE-Data (72bit)	**Slow-Data (24bit)**	**...**	**AMBE-Data (72bit)**	**Stop-Pattern (32+15bit)**

To start a new D-Star transmission you can use the **RPTR_START** or the **RPTR_HEADER** message.

Usage and benefit of the optional **RPTR_START** message:

In the situation of a repeater or a multi-HF (more than one DV-RPTR) configuration you can detect the beginning reception 138ms before you got the decoded header-data. Now it makes sense to start the transmitter to reduce the system-delay and mark the TX channel as “busy”. The DV-RPTR keeps up to 420ms after the TX-Delay transmitting the preamble. Within this time you should send the **RPTR_HEADER** message.

Once **RPTR_HEADER** is transmitted to the DV-RPTR the “indexed buffer” logic must be used to transfer **RPTR_DATA** messages.

If the stream ends, you send a **RPTR_EOT** message to the DV_RPTR.

Stream counting

Every DV stream you send to the modem contains a ID (one byte number). This streamID isn't transferred over HF. It helps to separate the streams on the DV-RPTR modem. The DV-RPTR is capable to transmit only one stream at the same time. A new stream must contain a new streamID different from the previous one. Beginning a stream with **RPTR_START** / **RPTR_HEADER** you select a new ID (for example up-counting).

After this message all RPTR_DATA messages with another ID will be ignored: If some process (playback...) keep transmitting data it doesn't matter. Only the valid RPTR_DATA will be processed.

Considerations:

What happen, if the DV-RPTR already transmits and gets a new (new stream ID) **RPTR_START** message?

Answer: Nothing: This message will be ignored.

What happen, if the DV-RPTR already transmits and gets a new **RPTR_HEADER** message?

Answer: The DV-RPTR stops the current stream after finishing the current voice-data (within a span of 0-20ms), transmits the Stop-Pattern, a preamble followed by a new Start-Pattern and the new header. This behaviour is like a break.

How the break can prevented?

For every “normal” transmissions you should check the TX-State enumeration (**RPTR_STATUS** request). Example: A emergency or break transmission should result in an immediately start of this stream. A local stream you can handle in this way too, but on normal UDP streams you should consider the TX-State and begin with the new on if TX-State is “last frame”, “eot” or “idle”.

Indexed buffer logic

1. Send START (optional).
2. Send HEADER.

The transmitter turns on, if off. If TX on a "BREAK" (EOT->Sync) was generated.

3. Send VOICEDATA
   USE the same ID you used on HEADER-msg and KEEP counting (PKTcount). If some packets lost, the gap is filled with silence. You can retransmit lost packets.
   Count up to Transmit-Buffer-Size-1, then restart with #0.

4a. Stop sending VOICEDATA
   results in stopping transmission after the last VOICE-packet is transmitted.
   A STOP PATTERN is append before PTT goes off.

4b. Send EOT-Cmd
   This geneates a STOP PATTERN just after the current VOICE-packet. After this the PTT goes off.

Receiving D-Star data

```
/* handle_hfdata() processes bit-flags from rptr_func
 * A typical reception look like this:
 *
 * D0 | 03 00 | 15 | 00 00 crc crc          >Preamble detected (not implemented jet)
 * D0 | 03 00 | 16 | 01 00 crc crc          >Start-Frame-Pattern detected
 * D0 | 2C 00 | 17 | 01 00 {header data} crc crc
 *                   ^ Biterrors in header (not implemented jet)
 * D0 | 0F 00 | 19 | 01 00 {VoiceData} 55 2D 16 crc crc > Voicedata with FrameSync
 *                   ^ pktcount 0 to 20 (defined by Receive-Buffer-Size-1)
 * D0 | 0F 00 | 19 | 01 01 {VoiceData} {SlowData} crc crc > Slowdata not descrambled
 * ...
 * D0 | 03 00 | 1A | 01 00 crc crc          > End of Transmission
 *                 ^ transmission counter 1..255,0
 *            ^ cmd / type of paket ( = part of pkt-data )
 *     ^ Length of Data (packetlength-5)
 * ^ FramestartID
 */
Receiving
---------
```

1. A PREAMBLE-Message was send (not implemented jet).
2a. A START-Message was send.
2b. A RXSYNC-Message was send (missing START-PATTERN/Header)
3. A HEADER was send (137.5ms after START), no message in the case of 2b.
4. VOICE-DATA packets was send in 20ms gaps
5a. A EOT was send (STOP-FRAME-PATTERN received)
5b. A RXLOST was send (no sync-frame-pattern detected)

DV-RPTR configuration

Structure of a configuration block

To have the option for expanding the configuration in later firmware versions, I defined a very simple block format to separate config-values in logical units. Every configuration block starts with an uniqe ID byte (from 0xC0 to 0xCF) followed by the length of the configuration (1 byte):

`<config-block-id> <blocksize> <config-block>`

All data value greater 1 byte are expected in little endian (PC) format and not in big-endian (the AVR32 controller uses big endian internally).

List of configuration blocks

ID	Length	Configuration
`0xC0`	4	Basic DV-RPTR configuration (physical radio parameters)
`0xC1`	12	Internal or attached transceiver setup
`0xC2`	40	Dongle or stand alone repeater: header configuration and data
`0xC3`	20	Dongle or stand alone repeater: message text
`0xC4`	8	AMBE-Addon TLV320AIC codec basic configuration
`0xC5`	12	AMBE-Addon TLV320AIC codec AGC and DRC register configuration
`0xC6`	56	AMBE-Addon TLV320AIC codec custom filter block coefficients

Configuration details

Basic configuration (0xC0)

The Basic configuration contain the basic radio parameters.

Stucture (with config-block header)

0	**`0xC0`**	Configuration ID
1	**`0x04`**	Length of configuration data
2	**`Flags`**	Bitset with the following meaning:

7	Halfduplex mode (1 = disables receiver while transmitting)
6	Dongle mode (1 = disabled PTT output pin while transmitting)
5	reserved
4	reserved
3	Automatic RX-Inversion detect (1 = enabled)
2	TX-Channel select: 0 = FSK pin (default), 1 = AFSK pin (channel B)
1	TX-Inversion
0	RX-Inversion

3	**`Modulation`**	Voltage Peak-Peak (value 255 ^= 3.00V_{pp})
4..5	**`TX-Delay`**	Delay between transmitter enable and start of transmitting preamble in milliseconds

Structure in source code

```
// configuration routines - physical config (MAIN config C0)
typedef struct PACKED_DATA {
  unsigned char  flags;
  unsigned char  mod_level;
  unsigned short txdelay;     // attention: little-endian here!
} t_config_0;
```

Transceiver setup (0xC1)

Till today only a RDA1846 based concept board exists and con be used as small transceiver. The only necessary values needed are the transmit- and receive-frequency. The presence of a connected transceiver can be evaluated using the SFC **`SFC_TRX_CAPABILITIES (0xC1)`**.

Stucture (with config-block header)

0	**`0xC1`**	Configuration ID
1	**`0x0C`**	Length of configuration data
3..6	**`RX-Freq`**	Receive frequency given in Herz [Hz]
7..10	**`TX-Freq`**	Transmit frequency [Hz]
11	**`Flags`**	Bitset with the following meaning:

7	
6	
5	
4	
3	
2	
1	
0	

12..14	reserved	For future use

Structure in source code

```
// configuration routines - physical config
typedef struct PACKED_DATA {
  unsigned int  rx_freq;      // in Hz little endian here
  unsigned int  tx_freq;      // in Hz little endian here
  unsigned char flags;  // bandwidth...
  unsigned char rsvrd[3];
} t_config_1;
```

Dongle header configuration (0xC2)

This configuration is only used at a stand alone configuration. In standard modem operation no header data needed to configure (a D-Star header is a part of the transmission data).

0	**`0xC2`**	Configuration ID
1	**`0x28`**	Length of D-Star header data
2	**`RPTR flags`**	Dongle / Repeater control flags

7..4	reserved
3	AMBE addon: microphone PTT switch enabled
2	AMBE addon PTT can break
1	AMBE addon: listen to the internet (voice data from PC is audible)
0	AMBE addon: listen to the radio (received voice data is audible)

3	**`0x00`**	Header Flag 1 (not used / internal setup)
4	**`0x00`**	Header Flag 2 (not used / internal setup)
5	**`0x00`**	Header Flag 3 (not used / internal setup)
6..13	**`RPT2Call`**	Header Repeater 2 value (8 chars)
14..21	**`RPT1Call`**	Header Repeater 1 value (8 chars)

22..29	**YourCall**	Header YourCall value (8 chars)
30..37	**MyCall**	Header own Call value (8 chars)
38..41	**MyCall 2**	Header own Call suffix (4 chars only)

Dongle message text (0xC3)

0	**0xC3**	Configuration ID
1	**0x14**	Length of message text (is fixed to 20 chars)
3..21	**message**	Message text

TLV320AIC basic configuration

TLV320AIC AGC and DRC configuration

TLV320AIC filter coefficients config