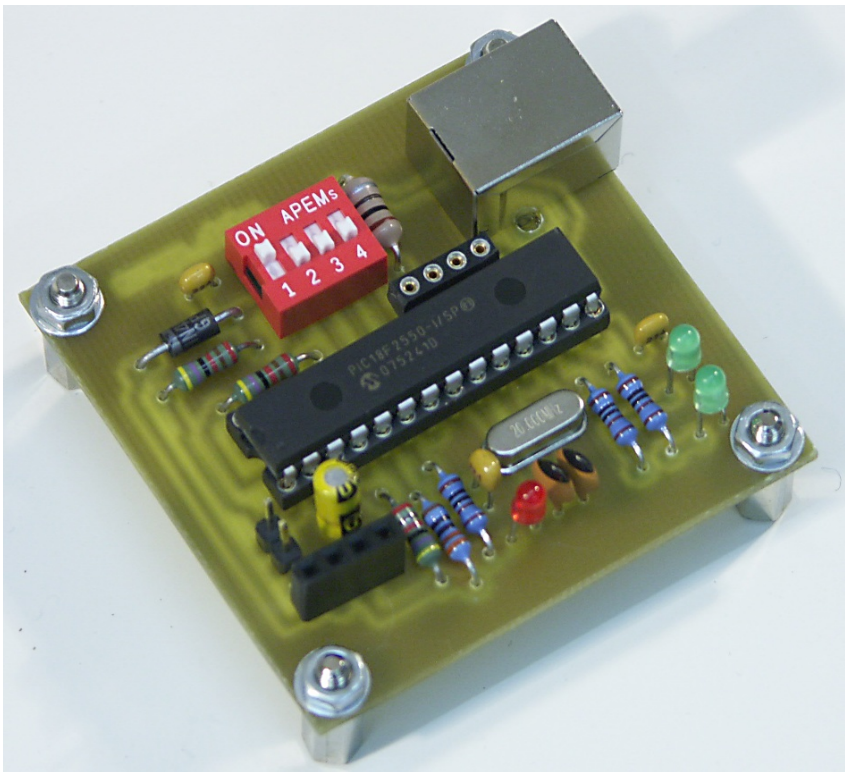
The Software-Interface of the adsbPIC-Decoder V2

Firmware 11



Author: sprut (www.sprut.de)
Last change: 12.11.2012

-

1 Table of contents

-

2 TERMS OF USE:

3 Introduction

This handbook is a very early draft probably full of typing errors and other mistakes. Normally I would not publish it in this bad condition, but I like to give the user something into the hands for the first step with my decoder and PC-software.
The handbook will be updated and “debugged” continuously to improve its quality.

This handbook describes the software interface of the adsbPIC-decoder. It is published to help other programmers to develop software that supports the adsbPIC-decoder.

4 The USB-Interface

The adsbPIC is a bus powered USB-2.0 device.

The decoder-firmware is using a serial-port-emulation and needs no additional drivers on most computer systems.
The decoder-bootloader needs microchip-MCD-driver (Windows) or libusb-driver (Linux).

4.1 USB-interface of the firmware

Interface	USB2.0 full speed
Power supply	bus powered
Power consumption	< 100 mA
VID	0x04D8
PID	0x000A

-

4.2 USB-interface of the bootloader

Interface	USB2.0 full speed
Power supply	bus powered
Power consumption	< 100 mA
VID	0x04D8
PID	0xFF0B
Number of USB-configurations	1
Number of interfaces	1
Number of endpoints	2
<u>Endpoint1:</u>	
Datadirection	OUT
Mode	BULK
bufersize	64 Byte
<u>Endpoint2:</u>	
Datadirection	IN
Mode	BULK
bufersize	64 Byte
<u>USB-Timeouts:</u>	
Write to device	100 ms
Read from device	1000 ms

5 Bootloader and Firmware

The boot-block of the PICs program memory (Addresses 0x0000 ... 0x07FF) contains the bootloader. The remaining flash memory can be used by the firmware

As bootloader for the adsbPIC the **bootloader-adsb** (a modified bootloader-0) is used. Bootloader and firmware are two independent programs. Both use the USB-interface in different ways. For the operating system of the PC both seem to be different USB-devices. However, only one of both can be active at a time.

The bootloader starts after reset or power-on in each of the both following cases:
1. Pin 1 of the control-PIC is connected to low-level (jumper is set)
2. The content of EEPROM-cell with address 0xFE is equal to 0xFF.

In all other cases the firmware starts.

The software adsbScope can activate the bootloader. To upload new firmware the software USBoot can be used.

-

6 Bootloader

6.1 Basics

From time to time a new version of the firmware will be published to fix errors and to integrate new features. The bootloader is a simple to use tool to load the new firmware into the PIC via the USB-interface.

The bootloader is a small piece of software, which has to be programmed into a special area of the control PIC of the adsbPIC. To program it into the control PIC a PIC programmer is needed. This can be a Brenner5 (with Windows-Software P18) or a Brenner8 (Firmware 0.5 or later; software US-Burn V1.2 or later) or any other modern PIC-programmer.

The bootloader is available in a separate hex-file. It is contained in the adsbPIC-decoder-ZIP-file. This bootloader is based on the Microchip-Bootloader for the „PICDEM USB FS DEMOBOARD“.

The communication uses both endpoints and will be initiated from the PC. It will always work in these 4 steps:

1. the PC writes data into the out-Endpoint.
2. the Bootloader works off the command.
3. the Bootloader writes data into the in-Endpoint
4. the PC reads the data from the in-Endpoint

The following table shows the structure of data blocks that are exchanged between PC and bootloader. The size of a data block is limited to 64 bytes.

The real length of the datablock depends on the command. If a command don’t needs any additional data at all, then even a length of only 1 byte is possible.

If the PC sends a data block that is to long for these command then the unnecessary bytes at the end of the datablock will be ignored. If at the end of a (to short) datablock some bytes are missing, then the bootloader will use data bytes that are randomly contained in the USB-buffer memory.

General data structure:

	Address	**Meaning**
Command	0x00	What to do
Length of data	0x01	Number of data bytes
Address low	0x02	Bits 0..7 of address
Address high	0x03	Bits 8..15 of address
Address upper	0x04	Bits 16..23 of address
data byte 1	0x05	
data byte 1	0x06	
...	...	
data byte n	0xFF	

-

The following commands are supported by the bootloader.

Name of the command	**Code in Datablock**
READ_VERSION	0x00
READ_FLASH	0x01
WRITE_FLASH	0x02
ERASE_FLASH	0x03
READ_EEDATA	0x04
WRITE_EEDATA	0x05
READ_CONFIG	0x06
WRITE_CONFIG	0x07
RESET	0xFF

Any other command will be ignored by the bootloader.

6.2 Commands for the Bootloader

6.2.1 READ_VERSION

I use this command to identify the bootloader and to read out its bootloader-version.
One can read out two bytes of information from the device.
Byte 3 has to have the device-identifier-value 0x01, or the device is no bootloader.
Byte 2 contains the bootloader-version.

PC -> Bootloader

Address	**Meaning**	**Content**
0x00	Command	0x00

The response contains the device-identifier: 0x01.

Bootloader -> PC

Address	**Meaning**	**Content**
0x00	Command	0x00
0x01	Not used	-
0x02	Version	-
0x03	Device	**0x01**

6.2.2 READ_FLASH

With this command the content of the program memory can be read out. Up to 59 consecutive bytes can be read in one step. The address points to the first byte to be read.

PC -> Bootloader

Address	**Meaning**	**Content**
0x00	Command	0x01
0x01	Data length	1 .. 59
0x02	Address low	
0x03	Address high	

-

0x04	Address upper	

The bootloader sends back the whole command-datablock with the memory-data appended to the end.

PC -> Bootloader

Address	Meaning	Content
0x00	Command	0x01
0x01	Data length	1 .. 59
0x02	Address low	
0x03	Address high	
0x04	Address upper	
0x05	**Data byte 1**	
...	...	...
0xXX	**Data byte n**	

6.2.3 WRITE_FLASH

With this command new program code can be written into the program memory of the control PIC. But before data is written into a memory area, it has to be erased by the ERASE_FLASH command. (If not, then the memory will finally contain the old code OR-combined with the new code.)

Exactly 16 consecutive bytes will be written in one step! The address points on the first address inside the program counter that will be used. The lowest 4 bits of the address have to be ‘0000’, thus the write process starts at the begin of a 16-byte-block of memory.
If the data length is different from 16, then the bootloader will do nothing!

PC -> Bootloader

Address	Meaning	Content
0x00	Command	0x02
0x01	Data length	**0x10**
0x02	Address low	0xX0
0x03	Address high	
0x04	Address upper	
0x05	Data byte 1	
...	...	...
0x14	Data byte 16	

The response contains the command only.

PC -> Bootloader

Address	Meaning	Content
0x00	Command	0x02

6.2.4 ERASE_FLASH

With this command program memory can be erased. The memory will be erased in blocks of 64 bytes. Several consecutive blocks can be erased in one step.
The address (byte 2...4) identifies the first block to be erased (the lower 6 bits are ignored / set to 000000).

PC -> Bootloader

Address	Meaning	Content
0x00	Command	0x03
0x01	Number of blocks	
0x02	Address low	
0x03	Address high	
0x04	Address upper	

The response contains the command only.

PC -> Bootloader

Address	Meaning	Content
0x00	Command	0x03

6.2.5 RESET

After the decoder received this command, it will disconnect from the USB-bus and execute a reset.

PC -> Bootloader

Address	Meaning	Content
0x00	Command	0xFF

In this case the bootloader sends NO response back to the PC.

-

6.3 Example for the use of the Bootloader

The following Delphi-code shows how to use the bootloader. It is using a procedure **Sende_Empfange(NrS, NrE)**. This procedure sends **NrS** bytes of the array **send_buf** to the USB-device and receives then **NrE** bytes from the USB-device and writes this data into the array **receive_buf**.
Both arrays are 64 bytes long.

The bootloader will be identified with READ_VERSION.
Then the program memory area from 0x800 up to 0x7FFF will be erased and programmed with new code from the array Hexin.Flash.

Finally the new programmed program memory will be read again and compared with Hexin.Flash to identify errors.

If no error was detected, then the EEPROM memory cell 0xFE will be erased and the decoder reseted.

```
// is there a bootloader ?
send_buf[0]:=READ_VERSION;    //0
Sende_Empfange(1, 4);
if (receive_buf[0] <>  READ_VERSION) or (receive_buf[3]<>1) then exit;

//program memory
Adresse    := $800;
Endadresse :=$7FFF;
while Adresse<Endadresse do begin
  //erase 64 byte
  send_buf[0]:= ERASE_FLASH;
  send_buf[1]:= 1;                             // 1 x 64 byte
  send_buf[2]:=  Adresse and $0000FF;          // low
  send_buf[3]:= (Adresse and $00FF00) shr  8;  // high
  send_buf[4]:= (Adresse and $FF0000) shr 16;  // upper
  Sende_Empfange(5,1);
  //write 4 x 16 byte
  for k:=0 to 3 do begin
    send_buf[0]:= WRITE_FLASH;
    send_buf[1]:= 16;                            // length 16 Byte
    send_buf[2]:=  Adresse and $0000FF;          // low
    send_buf[3]:= (Adresse and $00FF00) shr  8;  // high
    send_buf[4]:= (Adresse and $FF0000) shr 16;  // upper
    for L:=0 to 15 do  send_buf[5+L]:=Hexin.Flash[Adresse+L] and $FF;
    Sende_Empfange(21,1);
    Adresse:=Adresse+16
  end;
end;

//check
Fehler:=0;
```

-

```
Adresse    := $800;
Endadresse :=$7FFF;
while Adresse<Endadresse do begin
  send_buf[0]:= READ_FLASH;
  send_buf[1]:= 16;                            // length
  send_buf[2]:=  Adresse and $0000FF;          // low
  send_buf[3]:= (Adresse and $00FF00) shr  8;   // high
  send_buf[4]:= (Adresse and $FF0000) shr 16;   // upper
  Sende_Empfange(5,send_buf[1]+5);

  for k:=0 to receive_buf[1]-1 do begin
    if (receive_buf[k+5] and $FF) <> (Hexin.Flash[Adresse+k] and $FF)
    then begin
      inc(Fehler);
    end;
  end;
  Adresse:=Adresse+16
end;

if Fehler=0 then begin
  // write 0 into EEPROM-cells 0xFE and 0xFF
  send_buf[0]:= WRITE_EEDATA;
  send_buf[1]:= 2;            // length 1 Byte
  send_buf[2]:= $FE;          // low
  send_buf[3]:= 0;            // high
  send_buf[4]:= 0;            // upper
  send_buf[5]:= 0;
  send_buf[6]:= 0;
  Sende_Empfange(7,1);

  // reset device
  send_buf[0]:= RESET;
  Sende_Empfange(1,0);

end else Memo.lines.add('Flash-Error');
```

6.4 Example Code for Linux

The bootloader for the adsbPIC-decoder is (nearly) identical to the bootloader of the Brenner8-PIC-programmer. To understand bootloader-operation it may be helpful to study the usburn-software. Download this archive:
http://www.sprut.de/electronic/soft/usburn/linux/usburn_0_4.tar

It contains the C-project of the usburn-Linux-software. The bootloader-related code is contained in the file firmware.c. The main-file is usburn.c, The USB-related code is at the begin of the file in programmer_usb.c.

The code requires the installation of the libusb-package.

-

7 Firmware

The firmware of the adsbPIC received commands from the PC via USB, works off the commands and sends a response back to the PC
The firmware resides inside the control PIC of the decoder from address 0x800.

The communication uses both endpoints and will be initiated from the PC. It will always work in these 4 steps:

1. the PC writes an ASCII-string into the adsbPIC.
2. the firmware works off the command.
3. the firmware writes an ASCII-string into the USB-in-Endpoint
4. the PC reads the ASCII-string from the adsbPIC

The only command without a response-string from the decoder is the RESET-command.

Each command-string contains at least one bye of data. The first data-byte of a command string is the command. The following table contains all commands for the adsbPIC-firmware:

Command	**Code**	**Meaning**
READ_VERSION	0x00	Read device identifier and firmware version
SET_MODE	0x43	Control adsb-dataflow
SET_OFFSET	0x39	Set comparator-voltage offset, control AGC
READ_OFFSET	0x38	Read comparator voltage offset
SYS_PWM2	0x58	Set PWM (for test only)
SYS_ADC	0x51	Selects input for the ADC
RD_ADC	0x37	Measure voltage with ADC
SYS_EEPROM	0x5A	Read/write the EEPROM
SYS_I2C	0x54	write to the I2C-interface
RESET	0xFF	Reset the adsbPIC-decoder

7.1 Format of the command strings

PC and firmware communicate by ASCII-strings. This slows down the communication speed (compared to the speed of raw-data) but simplifies the use of the decoder. All other adsb-decoders use strings too, thus my decoder has to be compatible anyway.

Every command-string starts with the prefix '#'.
Then follow all data bytes. The bytes are converted into 2-digit hexadecimal strings. The bytes are interspaced by'-'.
The end of the string is a byte with the value 0x0D (CR).

To send the bytes 1, 5, 10, 255 to the decoder, it is formatted into this string:

-

‘#01-05-0A-FF’ followed by 0x0D

The responses from the decoder are strings too. The prefix is ‘#’ and at the end of each string are the 3 bytes 0x0A, 0x0D, 0x00.
The response string contains always 16 data-bytes.

7.2 General Commands

7.2.1 READ_VERSION

I use this command to identify the adsbPIC and to read out its firmware version. One can read out two bytes of information from the device.
Byte 3 has to have the value 4, or the device is no adsbPIC. Byte 2 contains the firmware version.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0x00

‘#00’

adsbPIC -> PC

Address	Meaning	Content
0x00	Command	0x00
0x01	Not used	-
0x02	Firmware version	?
0x03	Identifier	0x04

‘#00-00-06-04’

7.3 Commands for the Comparator

The comparator of the decoder converts analog video (from the receiver) into digital video (pulses). It needs a reference voltage. This voltage is generated inside the decoder, and it should be about 100 mV above the mean analog signal level.

The firmware has an automatic gain control (AGC). It measures the mean analog signal level every 1.3 seconds and adjusts the reference voltage level. Thus the offset is kept constant.

The user can monitor the voltage levels, read out and change the offset value.

At reset/power-on the decoder reads the last used offset from the internal EEPROM and activates the AGC.

7.3.1 SET_OFFSET

This command sets the offset voltage between mean analog signal level and reference voltage level. The default is 100mV, but some decoders may work better with other offsets. The offset is an 8 Bit value and is the value in Millivolt. The new value is stored inside the decoder permanently.

With byte 2 of the command the AGC can be switched on (1) or off (0). For normal operation the AGC has to be switched on.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0x39
0x01	Not used	-
0x02	AGC mode	0-off 1-on
0x03	AGC offset	10..250

'#39-00-01-64'

The response from the decoder contains no useful information.

adsbPIC -> PC

Address	Meaning	Content
0x00	Command	0x39
…	…	…

'#39-…'

The decoder stores the AGC-offset-voltage (in Millivolt) in EEPROM at address 0x00. At EEPROM-address 0x03 the AGC-mode is stored. (But for special reasons the value 0 in this EEPROM-cell means AGC-on and the value 1 means ADC-off.)

At reset or power-up the decoder reads the both values from the EEPROM and controls AGC based on the stored values.

7.3.2 READ_OFFSET

With this command the user can read out the momentary used offset voltage value and AGC-mode.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0x38

'#38'

The byte 2 of the response contains the AGC-mode (1=on; 0=off) while byte 3 contains the offset voltage in Millivolt.

adsbPIC -> PC

Address	Meaning	Content
0x00	Command	0x38
0x01	Not used	-
0x02	AGC mode	0-off 1-on
0x03	AGC offset	10..250

'#38-00-01-64'

-

7.3.3 SYS_ADC

This command selects the ADC input. Possible inputs are the reference voltage level and the mean analog signal level.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0x51
0x01	ADC-input	0-reference 1-analog sig.

'#51-01'

The response from the decoder contains no useful information.

7.3.4 RD_ADC

The ADC will measure the voltage level at the selected ADC input (see SYS_ADC) and repots it to the PC.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0x37

'#37'

The adsbPIC sends the 10-bit result (raw data). ADRESL contains the lower 8 bits while ADRESH contains the upper 2 bits of the ADC result.

adsbPIC -> PC

Address	Meaning	Content
0x00	Command	0x37
0x01	ADRESL	lower 8 bit
0x02	ADRESH	upper 2 bits

#37-00-00-A3'

The raw 10-bit result has to be converted into a usable value. The resolution of the ADC is 4.88mV (5V/1024). To convert the raw-ADC-data into a voltage in Millivolt, it has to be multiplied by 4.88.

It is highly recommended to disable the ADSB-decoding for the time of the voltage measurement:

'#43-00'	set mode to 0 (stop decoding)
'#51-01'	selects ADC input (analog signal)
'#37'	measure voltage
'#43-02'	set old mode (e.g. 2)

7.3.5 SYS_I2C

This command can be used to send commands via I2C-interface. Some users may use a satellite-TV-tuner as receiver. These tuners require control commands to be tuned to 1090 MHz.

-

The I2C-interface of the tuner has to be connected to the both pins RB5 (Data) and RB6 (Clock) of the decoders microcontroller. If the I2C-interface is used, then the manual switches switch-A (CRC) and switch-B (1MBit) can not be used.

The I2C-interface supports 2 different modes. The first mode transmits a fixed code sequence to the tuner: “C2-30-8C-8E-00”. This should be the correct code for the BSJE3-159A tuner. This mode is selected, if the command-byte is followed by 0x00.

PC -> adsbPIC

Address	**Meaning**	**Content**
0x00	Command	0x54
0x01	Mode	0-send standard data

‘#54-00’

The second mode can transmit any code sequence to the tuner. The maximum length of the sequence is 14 bytes. This mode is selected, if the command-byte is followed by a number in the range of 0x01 … 0x0E. This number is equal to the number of bytes following.

PC -> adsbPIC

Address	**Meaning**	**Content**
0x00	Command	0x54
0x01	Mode	x-number of databytes
0x02	1st databyte	0..255
0x03	2nd databyte	0..255
…		

‘#54-05-C2-30-8C-8E-00’

If the transfer was successful, then the first byte of the response is 0x54. If the decoder was not able to claim the bus, then the first byte of the response is 0x00. If a bus error was detected, then the first byte of the response is 1 or 3. If the tuner failed to send ACK-signals, then this byte is set to 2 or 3.

adsbPIC -> PC

Byte0	**Meaning**
0x54	ok
0x00	bus was busy
0x01	bus error during transmission
0x02	no ACK from slave (tuner)
0x03	bus error and no ACK

7.3.6 SYS_PWM2 (for test only)

Not needed

-

7.4 Commands for the control-PIC

7.4.1 SYS_EEPROM

With this command the user can write information into the decoders EEPROM or read out information from this EEPROM. A practical application is the activation of the bootloader.
To activate the bootloader the EEPROM cell 0xFE has to be set to 0xFF and then a reset command has to be executed.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0x5A
0x01	Mode	0x02
0x02	EE-address	0xFE
0x03	Data byte	0xFF

'#5A-02-FE-FF'

7.4.2 RESET

After the decoder received this command, it will disconnect from the USB-bus and execute a reset.
No response will be send back to the PC.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0xFF

The adsbPIC will not send any response to the PC.

7.5 Commands for the ADSB-decoder

7.5.1 SET_MODE:

This command controls the ADSB-decoder. The decoding can be switched on and off. In addition the CRC-check and the DF17/18/19-filter can be activated.
The use of frame-numbers and timestamps can be activated too and an alternative binary data format can be selected.

PC -> adsbPIC

Address	Meaning	Content
0x00	Command	0x43
0x01	mode	...

'#43-02'

the structure of the mode-byte:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3..0

-

binary format	heart beat	frame numbers	MLAT / time-tag	Mode 0..4

The **lower 4** bits of the databyte 0x01 contain the mode of the decoder. There are 4 basic modes:

Mode	CRC	filter	description
0	No	No	The decoder will not send any ADSB-data to the PC
2	No	No	The decoder will send all received ADSB-frames to the PC
3	No	Yes	The decoder will send only DF17, DF18 and DF19-frames to the PC
4	Yes	Yes	The decoder will send only DF17, DF18 and DF19-frames with correct CRC-checksum to the PC

The decoder will by default use the RAW-data format (the same that is used by other decoders) and send the ADSB-frames as text-strings to the PC:
***12345678901234;**
***1234567890123456789012345678;**

If the **bit 4** of the mode-byte is set to 1, then the decoder will add time stamps to all frames. These time stamps are required for MLAT. The time stamp is a 48-bit counter that increments with 12-MHz clock. It is used as prefix to the raw-data at each frame. If the time counter is active, then instead of the leading '*' a '@' is used as prefix of the string.
@**123456789ABC**12345678901234;
@**123456789ABC**1234567890123456789012345678;

If the **bit 5** of the mode-byte is set to 1, then the decoder will add 32-bit frame numbers to all frames. This can be used during test to identify frame losses in the PC-software. To keep compatibility the frame numbers are added behind each frame as an independent text-string. The prefix for the frame-number-string is '#'.
*12345678901234;**#00000123;**
*1234567890123456789012345678;**#00000124;**

If the **bit 6** of the mode-byte is set to 1, then the decoder will send heartbeats. These are additional DF23 frames with information about the decoder state.

If the **bit 7** of the mode-byte is set to 1, then the decoder will not use the normal RAW-data-format at all. Instead it will use a so called binary format. It is identical to the binary output format of the Beast (receiver decoder Mode-S-Beast). The binary format results in less processing load for the decoder and less load for the USB-port. In binary format every frame is made of 16 (short frame) or 23 (long frame) byte.

Byte	Value	Content
0	0x1A	start symbol
1	0x32 (short frame) or 0x33 (long frame)	marks long or short frame
2 .. 7	6 byte long time tag	for MLAT
8	0x00	not used by adsbPIC
9..15	7 byte RAW data	RAW data
16..22	7 byte RAW data	additional RAW data for long frame

-

In binary format the time TAG is always sent while frame numbers are never sent. The values of the Bits 4 and 5 are ignored.

The command-response from the decoder contains no useful information.

7.6 RS232-Interface

Since firmware 6 the decoder has an additional RS232-interface. Since firmware 7 the speed and polarity of this interface can be changed. No special commands are necessary to change the RS232-setup. All this can be realized by the manipulation of two EEPROM cells. The old SYS-EEPROM-command can be used for this.

At reset or power-up the decoder reads the EEPROM cells 0x01 and 0x02. The value at 0x01 is used to set the RS232-speed while the value at 0x02 controls the polarity of the RS232-signals.

<u>Values at the EEPROM cell 0x01:</u>

Value=0	115.200 kbit (default)
Value=1	921.600 kbit (known as 1 Mbit)
Value=2	19.2 kbit

Any other value at the cell 0x01 will set the speed to 115.200 kbit.

<u>Values at the EEPROM cell 0x02:</u>

Value=0	polarity for external RS232-driver-chips (default)
Value=1	polarity for use without external driver

Any other value at the cell 0x02 will set the polarity for the use with a driver-chip.

To change the speed or polarity of the RS232-interface one has to write the correct values into the EEPROM-cells and to execute a reset command for the decoder.

8 Performance

If users compare decoders, then they normally discuss frame rates. As higher the number of received frames per minute (fpm), as better the performance. This is not a bad point of view, but in areas with low traffic, the frame rate is limited by the number of aircraft too. Because of this, I prefer the frames per aircraft and minute (fpma).

Every aircraft transmits in every second two DF17-frames and 10 other frames. This adds up to 700 frames per aircraft and minute (fpma). To track an aircraft continuously, 100 … 200 fpma are enough.

The decoder is not able to detect and process all received frames. The detection probability depends on receiver output signal quality, the number of frames per minute at the decoder input and the RAW-data mode (ASCII or binary).

In an ideal test environment (with an ADS-B-frame-generator) I was able to measure the following results:

If the “receiver” delivers 6000 fpm, then the decoder will detect about 85% of these frames. If binary RAW-data format is used, then 95% is possible. At higher fpm-values, the detection probability goes down. If ASCII-format is used for RAW-data, then not more then 75.000 fpm can be reported by the decoder. If binary format is used then the limit is at 100.000 fpm.

Connected to a real receiver, the frame rate will normally not exceed 6.000 fpm. The signal quality of the receiver output is not ideal. Frames from different aircraft have different power levels, and noise is confusing the decoder.

FPGA-decoders can reach better performance. The BEAST detects about 6 times more frames (compared to my decoder). This results in the double number of detected aircraft.

-

9 EEPROM of the control-PIC

Some values are stored inside the EEPROM of the decoder.

The value at address 0xFE controls the bootloader. If its value if equal tot 0xFF, then the bootloader will be activated after reset or power-on.

from	**to**	**No of bytes**	**Data type**	**Value**	**Default**
0x00	0x00	1	byte	Offset voltage in mV	100
0x01	0x01	1	byte	RS232-speed	0
0x02	0x02	1	byte	RS232-polarity	0
0x03	0x03	1	byte	Offset-voltage-regulation	0
…					
…					
0xFE	0xFE	1	Byte	Bootloader marker	0