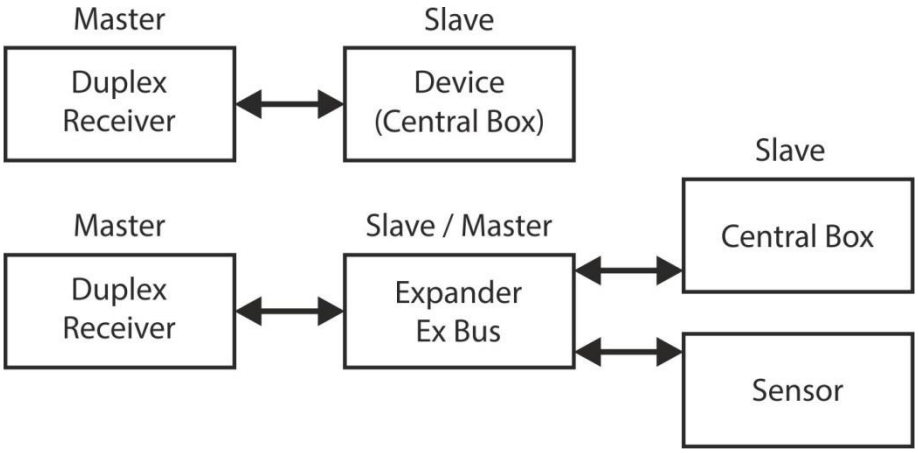
EX Bus communication protocol

EX Bus is a standard of serial data transmission, designed primarily for transmission of operational information between a receiver and other devices connected to it. EX Bus replaces standard of channel values transmission in the form of PPM, and also provides security features, faster response, and bidirectional communication (transmission of EX telemetry). EX Bus also enables transmission of information for remote configuration of connected devices via the DC/DS transmitters. This function is available only for the products of JETI model and its description is not a part of this document.

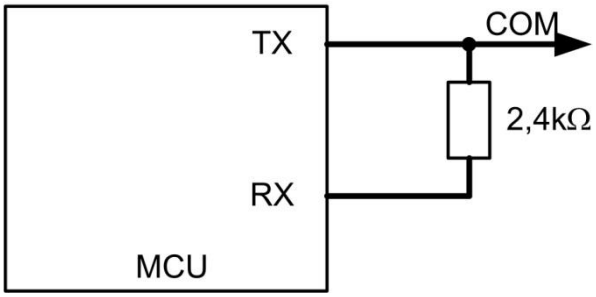
Topology

The bus has a topology called "point to point". The receiver is marked as a "Master" in the network, because it always initiates the communication. A device connected to the receiver is marked as a "Slave". In case of request for connecting several devices to a receiver, it is mandatory to use EX Bus Expander that processes all inputs and transforms them into one single output.



Physical layer

The communication is realized by serial asynchronous interface (USART) in a half-duplex mode.
Communication speed: **125 kBaud (LowSpeed) or 250 kBaud (HighSpeed)**
Number of data bits: **8**
Number of stop bits: **1**
Parity: none
Communication lines RX and TX are physically connected through a resistor (we recommend the value of 2.4 kilo ohms).

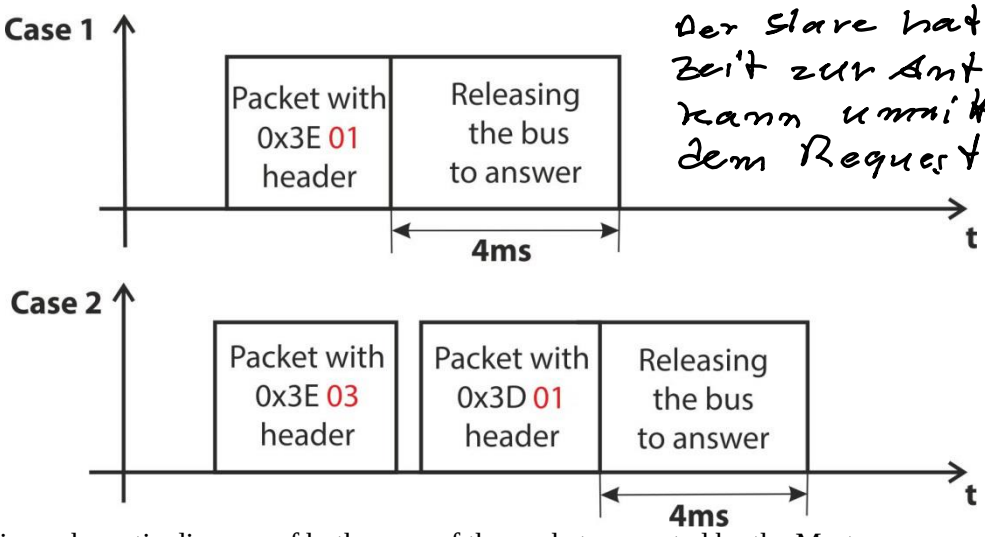


Logical levels:
Maximum level of logical zero "0": 1,5V
Minimum level of logical one "1": 3,0V

Accessing to the shared data bus

The only element of the network that initiates all communications is a "Master". The "Slave" always just responds to the questions of the "Master". In the specification of the EX Bus, there exist two types of packets that are generated by the "Master":

- A packet with the header, whose second byte is 0x01. It subsequently gives further scope for Slave to respond (releases the line for at least 4ms)
- A packet with the header, whose second byte is 0x03. It subsequently does not give any scope for Slave to respond.



Der Slave hat max. 4ms Zeit zur Antwort. Diese kann unmittelbar nach dem Request erfolgen

The figure above is a schematic diagram of both cases of the packet generated by the Master.
In the first case the Master generates a packet with the header, whose second byte is 0x01, which is always followed by releasing of a bus reserved for the "Slave".
In the second case the Master first generates the packet with the header, whose second byte is 0x03, which is not followed by releasing of a bus and then the Master generates the packet, whose second byte is 0x01.

The bus is released in the following way: The TX line is reconfigured as an input with the use of an internal pull-up resistor. After the timeout has passed, the TX line returns into output state.

The Master transfers information about channel values in data packets and also requests the telemetry and expects the response from the Slave.

Recognizing the communication speed

The Master can communicate at either LS (125kb) or HS (250kb) speed. The Slave by verifying the correctness of the packet (CRC) and should answer at this speed. The Slave should recognize the line speed, i.e. receives the data and, according to the success rate of the reception, keeps or switches the speed. The Master **does not switch** the speed dynamically during the operation.

The format of packets generated by the Master

Master sends to the Slave packets containing the information about channel values, requests for EX telemetry, and requests for the JETIBOX menu. In the communication, there might appear also other packet types, but they are not described in this document. These undocumented proprietary packets will follow the general format of the data, see below.
In case the Master does not receive the channel values from the receiver, it does not send any packet to the Slave. E.g. in case of a signal loss between a transmitter and a receiver, the Master (receiver) does not send any packets with the channel values to the Slave. Each packet contains an identifier. The Slave must insert the identifier value to the packet when sending the requested data back to the Master.

General format of packets generated by the Master

Byte No.	Length[Byte]	Data	Note
1	1	HEAD	Header; 0x3E or 0x3D
2	1	HEAD	Header; 0x01 or 0x03
3	1	LEN	Packet length including the header and CRC
4	1	Packet_ID	Packet ID
5	1	DATA_ID1	The data identifier 1
6	1	SUB_LEN1	Length of data block 1
7	SUB_LEN1	BLK1	Block data 1
7+SUB_LEN1	1	DATA_ID2	The data identifier 2
8+SUB_LEN1	1	SUB_LEN2	Length of data block 2
9+SUB_LEN1	SUB_LEN2	BLK2	Block data 2
….	….	….	….
	2	CRC16	CRC 16 CCITT; data sequence LSB, MSB

A packet containing the channel data

Byte No.	Length[Byte]	Data	Note
1	1	0x3E	Header
2	1	0x01 or 0x03	Header
3	1	LEN	Packet length including the header and CRC
4	1	Packet_ID	Packet ID
5	1	0x31	The data identifier – channels values
6	1	SUB_LEN	Length of data block
7	SUB_LEN	Data array	Channel values, data type for 1 channel is uint16_t, data sequence LSB,MSB, 1b=1/8us
7+SUB_LEN	2	CRC16	CRC 16 CCITT; data sequence LSB, MSB

The packet with the telemetry request

Byte No.	Length[Byte]	Data	Note
1	1	0x3D	Header
2	1	0x01	Header
3	1	LEN	Packet length including the header and CRC
4	1	Packet_ID	Packet ID
5	1	0x3A	The data identifier – request for telemetry
6	1	0	Length of data block
7	2	CRC16	CRC 16 CCITT; data sequence LSB, MSB

The packet with the request for the JETIBOX menu

Byte No.	Length[Byte]	Data	Note
1	1	0x3D	Header
2	1	0x01	Header
3	1	LEN	Packet length including the header and CRC
4	1	Packet_ID	Packet ID
5	1	0x3B	The data identifier – request for JETIBOX menu
6	1	0x01	Length of data block
7	1	BUTTON 0bLDUR0000	bit L – 0 if Left button is pressed, otherwise 1, bit D – 0 if Down button is pressed, otherwise 1, bit U – 0 if Up button is pressed, otherwise 1, bit R – 0 if the Right button is pressed, otherwise 1,
8	2	CRC16	CRC 16 CCITT; data sequence LSB, MSB

The format of packets generated by the Slave

The Slave generates only responses to the Master's requests. The Master requires the EX telemetry data or the JETIBOX menu. The description of EX telemetry can be found it in „JETI_Telem_protokol" document that is to be found on web pages www.jetimodel.com.

Packet with the EX telemetry

Byte No.	Length[Byte]	Data	Note
1	1	0x3B	Header
2	1	0x01	Header
3	1	LEN	Packet length including the header and CRC
4	1	Packet_ID	Packet ID
5	1	0x3A	The data identifier - EX telemetry
6	1	SUB_LEN	Length of data block, 0 – in case the EX telemetry is not sent
7	SUB_LEN	EX telemetry	EX telemetry starting with 0xNF and ending CRC8
7+SUB_LEN	2	CRC16	CRC 16 CCITT; data sequence LSB, MSB

Packet with the JETIBOX menu

Byte No.	Length[Byte]	Data	Note
1	1	0x3B	Header
2	1	0x01	Header
3	1	LEN	Packet length including the header and CRC
4	1	Packet_ID	Packet ID
5	1	0x3B	The data identifier – JETIBOX menu
6	1	SUB_LEN	Length of data block - 0x20
7	SUB_LEN	EX telemetry	32 characters for the JETIBOX screen
7+SUB_LEN	2	CRC16	CRC 16 CCITT; data sequence LSB, MSB

Checksum

The checksum is a 16-bit, type CCITT. The checksum starts at the first byte of the message (0x3B for Slave packet). Sample code to calculate the CRC16-CCITT in the C language:

```
uint16_t crc_ccitt_update( uint16_t crc, uint8_t data )
{
uint16_t ret_val;
  data ^= (uint8_t)(crc) & (uint8_t)(0xFF);
  data ^= data << 4;
  ret_val = ((((uint16_t)data << 8) | ((crc & 0xFF00) >> 8))
            ^ (uint8_t)(data >> 4)
            ^ ((uint16_t)data << 3));
  return ret_val;
}
uint16_t get_crc16z(uint8_t *p, uint16_t len)
{
    uint16_t crc16_data=0;

    while(len--) { crc16_data=crc16_update(crc16_data, p[0]); p++; }
    return(crc16_data);
}
```

Packet example - Master

```
0x3E 0x03 0x28 0x06 0x31 0x20 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82
0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F 0x82 0x1F
0x82 0x1F 0x82 0x1F 0x4F 0xE2

0x3E 0x03 - Packet header that forbids answering
0x28 - Message length (40)
0x06 - Packet ID
0x31 - The data identifier - channel values
0x20 - Length of data blocks (32) - 16 channels x 2B
0x1F82 - Value of the 1st channel (8066)/8000000 = 1,00825ms
…
0xE24F - CRC16-CCITT
```

0x1F82 = 8066

```
0x3D 0x01 0x08 0x06 0x3A 0x00 0x98 0x81

0x3D 0x01 - Packet header with a permission to answer
0x08 - Message length (8)
0x06 - Packet ID
0x3A - Data identifier - request for telemetry
0x00 - Zero length -> Request
…
0x8198 - CRC16-CCITT
```

```
0x3D 0x01 0x09 0x88 0x3B 0x01 0xF0 0xA3 0x24

0x3D 0x01 - Packet header with a permission to answer
0x09 - Message length (9)
0x88 - Packet ID
0x3B - Data identifier - request for JETIBOX menu
0x01 - Length of the data
0xF0 - Information of the buttons
0x24A3 - CRC16-CCITT
```

Packet example - Slave

```
0x3B 0x01 0x20 0x08 0x3A 0x18 0x9F 0x56 0x00 0xA4 0x51 0x55 0xEE 0x11 0x30 0x20 0x21
0x00 0x40 0x34 0xA3 0x28 0x00 0x41 0x00 0x00 0x51 0x18 0x00 0x09 0x91 0xD6
```

0x3B 0x01 – Packet header
0x20 – Message length (32)
0x08 – Packet ID
0x3A – Data identifier – **EX telemetry**
0x18 – Length of data blocks (24)
0x9F… – EX telemetry
0xD691 – CRC16-CCITT

```
0x3B 0x01 0x28 0x88 0x3B 0x20 0x43 0x65 0x6E 0x74 0x72 0x61 0x6C 0x20 0x42 0x6F 0x78
0x20 0x31 0x30 0x30 0x3E 0x20 0x20 0x20 0x34 0x2E 0x38 0x56 0x20 0x20 0x31 0x30 0x34
0x30 0x6D 0x41 0x68 0xEB 0xDE
```

0x3B 0x01 – Packet header
0x28 – Message length (40)
0x88 – Packet ID
0x3B – Data identifier – **JETIBOX menu**
0x20 – Length of data blocks (32)
0x43… – EX telemetry
0xDEEB – CRC16-CCITT