USB-2-X

Host Communication Protocol

Version: 0.23
2009-JUN-20

TRINAMIC
MOTION CONTROL

TRINAMIC Motion Control GmbH & Co. KG
Sternstraße 67
D - 20357 Hamburg, Germany
http://www.TRINAMIC.com

Table of contents

Table of tables

1 Life support policy

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specific written consent of TRINAMIC Motion Control GmbH & Co. KG.

Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

2 Basic definitions

2.1 Communication

Communication between the USB host and the USB-2-X device always takes place over USB. The communication protocol is a master-slave protocol: The USB host is the master sending commands to the USB-2-X device. The USB-2-X device is the slave executing commands and sending back responses to the USB host if required by the command. Communication is based on data packets containing start character, packet ID, payload size, payload data, packet checksum and stop character. If start, stop and some other control characters are part of the payload data itself there must be foregoing a special delimiter control character to indicate that they are not to be treated as control characters. Packet reception can be acknowledged by the ACK control character or in case of an error by the NAK control character.

2.2 Data packets

STX

1 Byte:	Packet ID (must not be a control character)
2 Bytes:	Payload Size n (n < 256), 1st Byte = 0xF0 + High Nibble of n, 2nd Byte = 0xF0 + Low Nibble of n
n Bytes:	Payload Data
2 Bytes: 8-Bit Nibble	Checksum c, calculated by adding ID, size and payload bytes, 1st Byte = 0xF0 + High of c, 2nd Byte = 0xF0 + Low Nibble of c

ETX

Payload data must not contain STX, ETX, and DLE, ACK or NAK control characters. If such a character shall be transmitted as a payload data byte it has to be preceded by a DLE character. These additional DLE characters must not be included in the payload size.

The checksum is calculated by 8-bit addition starting from the packet ID byte up to and including the last payload data byte. Additional DLE control characters have to be included in the checksum calculation.

Each packet is responded by the receiver by either ACK indicating successful command execution or by NAK indicating an error condition. The composition of packets is the same for both transfer directions.

2.3 ASCII codes of control characters

Control Character	ASCII Code (decimal)
STX	2
ETX	3
ACK	6
DLE	16
NAK	21

Table 2.1: ASCII Codes of control characters

3 Definition of data packets

3.1 Overview

Table 3.1 shows an overview of all USB-2-X packet ID codes. Packet ID codes in the left column are commands sent from the USB host to the USB-2-X device. Some commands require a response from the USB-2-X device. ID codes of USB-2-X responses are shown in the right column.

Command Packet ID (Host→USB-2-X)	**Purpose**	**Response Packet ID** (USB-2-X→Host)
0x31	IIC-Write	---
0x32	IIC-Read	0x42
0x33	CAN-Write	---
0x34	CAN-Read	0x44
0x35	CAN-Filter	---
0x36	LIN-Write	---
0x37	LIN-Read	0x47
0x38	SPI-Write	0x48
0x39	SPI-Read	0x49
0x3A	SPI-Write-No-Response	---
0x3B	RS485-Write	---
0x3C	RS485-Read	0x4C
0x51	IIC-Bit rate	---
0x52	CAN-Bit rate	---
0x53	LIN-Bit rate	---
0x54	SPI-Init	---
0x55	RS485-Bitrate	--
0xFF	Firmware-Version	0xFo

Table 3.1: Overview of command and response codes

3.2 IIC write

USB host transmits:

- Packet ID: 0x31
- Payload Data: 1 Byte = IIC Address, followed by the IIC data bytes

USB-2-X responds:

- ACK or NAK

3.3 IIC read

USB host transmits:

- Packet ID: 0x32
- Payload Data: 1 Byte IIC Address, 1 Byte Number of bytes to be read

USB-2-X responds:

- ACK or NAK

Then USB-2-X transmits the read data in the following data packet:

- Packet ID: 0x42
- Payload Data: the read data, number of bytes as requested by host

3.4 IIC bit rate

USB host transmits:

- Packet ID: 0x51
- Payload Data: 1 Byte IIC Bit rate Code (see table 3.2):

IIC Bit rate Code	resulting IIC Bit rate [kBit/s]
0	11,71875
1	23,43750
2	35,15625
3	46,87500
4	58,59375
5	93,75000
6	140,62500
7	187,50000
8	234,37500
9	375,00000

Table 3.2: IIC Bit rate Code

USB-2-X responds:

- ACK or NAK

3.5 CAN write

As long as the CAN interface is switched off (e.g. after power-on) each *CAN write* command is responded with NAK. The CAN interface can be switched on using the *CAN bit* rate command.

USB host transmits:

- Packet ID: 0x33
- Payload Data:
  - 4 Byte CAN identifier, most significant byte first, format according to table 3.3 and table 3.4
  - 0...8 Bytes CAN message data. In case of a remote frame (RTR=1) the message data bytes won't be transmitted on the CAN bus but rather their number is treated as the data length code (DLC) of the remote frame. There must be as many data bytes as requested by the remote frame.

USB-2-X responds:

- ACK or NAK

31	30	29	28 0
IDE=1	RTR	don't care	Extended Identifier Bits [28:0]

Table 3.3: Extended CAN identifier format

31	30	29 11	10 0
IDE=0	RTR	don't care	Standard Identifier Bits [10:0]

Table 3.4: Standard CAN identifier format

3.6 CAN read

USB host transmits:

- Packet ID: 0x34
- Payload Data: no payload data

USB-2-X responds:

- ACK or NAK

Then USB-2-X transmits the read data in the following data packet:

- Packet ID: 0x44
- Payload Data (only if payload size differs from zero):
  - 1 Byte Error code: 0=no error
    1=software receive buffer overflow
    2=hardware receive buffer overflow
    3=software and hardware receive buffer overflow
  - 4 Bytes reception time, most significant byte first, resolution $170^2/_3$ µs, wrap-around after 203h 36m 47,75s
  - 4 Byte CAN identifier, most significant byte first, format according to table 3.3 and table 3.4
  - 0...8 Bytes CAN message data. In case of a remote frame (RTR=1) the data bytes were not received from the CAN bus, their values have to be treated as don't care. However, their number indicates, how many data bytes the requested message shall contain.

USB host responds:

- ACK or NAK

3.7 CAN filter

This command can be used to restrict the reception of CAN messages to dedicated identifiers or ranges of identifiers. By default all CAN messages are received and stored in the internal software receive buffer. This may lead to buffer overflow: messages received earlier will be overwritten by messages received later if they are not read out in time using the *CAN* read command.

USB host transmits:

- Packet ID: 0x35
- Payload Data:
  - 1st Byte: Filter Organization: 0=one 32-bit Filter
    1=two 16-bit Filter
    2=four 8-bit Filter
    3=no message will pass the Filter (bytes may be omitted)

- 2nd to 5th Byte: 4 Byte CAN Identifier, most significant byte first, Interpretation acc. to Filter Organization:
  one 32-bit Filter: Format acc. to table 3.3 and table 3.4
  two 16-bit Filter: Format acc. to table 3.5 and table 3.6
  four 8-bit Filter: Format acc. to table 3.7

- 6th to 9th Byte: 4 Byte Mask (Format like Identifier):
  Bit=1: Identifier-Bit is ignored by the filter
  Bit=0: Identifier-Bit must match in order that the message is stored

USB-2-X responds:

- ACK or NAK

15	14	13... 0
IDE=1	don't care	Extended Identifier Bits [28:15]

Table 3.5 16-bit Filter Format for Extended CAN Identifier

15	14	13... 11	10... 0
IDE=0	RTR	don't care	Standard Identifier Bits [10:0]

Table 3.6 16-bit Filter Format for Standard CAN Identifier

7... 0
Extended Identifier Bits [28:21] Standard Identifier Bits [10:3]

Table 3.7 8-bit Filter Format for Extended or Standard CAN Identifier

3.8 CAN bit rate

USB host transmits:

- Packet ID: 0x52
- Payload Data:
  - 1 Byte CAN Bit rate:

    See table 3.8. Power-on setting is 0: CAN controller disabled and CAN transceiver in standby mode. This is the suggested setting to save power if CAN is not used.

    Bit 7 (MSB)=1: Incoming CAN messages which have passed the filter will directly be forwarded to the USB host using packet ID 0x44 without prior *CAN read* command. The USB host must not acknowledge such data packets.

CAN Bit rate Code	resulting CAN Bit rate [kBit/s]
0	CAN disabled
1	10
2	20
3	50
4	100
5	125
6	250
7	500
8	800
9	1000

Table 3.8 CAN bit rate

USB-2-X responds:

- ACK or NAK

3.9 LIN write

USB host transmits:

- Packet ID: 0x36
- Payload Data:
  - 1 Byte LIN message ID
  - 0..8 Bytes LIN message data
  - 1 Byte Period:
    - Bit 7 (MSB) = 1: send this LIN message periodically to avoid the idle condition on the LIN bus. A previously defined periodic LIN message is replaced by this message, i.e., only one message can be defined as periodic message at any one time. Singular messages can be sent in parallel with the periodic message even if they have the same ID. Singular messages delay the transmission of the periodic message, i.e., the periodic message will only be sent after not having sent a singular message for the duration of one period.
      Bits [6...0] = Transmission period in integer multiples of 10 milliseconds
      Period = 0: stop transmission of periodic LIN message immediately
      After Reset initially no message is sent periodically.
    - Bit 7 (MSB) = 0: send this LIN message only once (singular message). A previously defined periodic LIN message is sent out periodically further on.
      Bits [6...0] are don't care

USB-2-X responds:

- ACK or NAK

3.10 LIN read

USB host transmits:

- Packet ID: 0x37
- Payload Data:
  - 1 Byte LIN message ID
  - 1 Byte Number of requested data bytes

USB-2-X responds:

- ACK or NAK

Then USB-2-X transmits the read data in the following data packet:

- Packet ID: 0x47
- Payload Data:
  - 1 Byte Error code: 0 = no error
    1 = read timeout
    2 = LIN checksum error
    3 = loss of data (receive overrun)
    4 = LIN header corrupted
  - 0..8 Bytes LIN message data

USB host responds:

- ACK or NAK

3.11 LIN bit rate

USB host transmits:

- Packet ID: 0x53
- Payload Data: 1 Byte Bit rate: 0 = 2400 bit/s, 1 = 9600 bit/s, 2 = 19200 bit/s

USB-2-X responds:

- ACK or NAK

3.12 SPI write

This command allows transferring data bytes between the USB host and the SPI slave over the SPI bus. USB-2-X always acts as the SPI master. The received number of bytes equals the sent number of bytes.

USB host transmits:

- Packet ID: 0x38
- Payload Data: data bytes to be sent

USB-2-X responds:

- ACK or NAK

Then USB-2-X transmits the read data in the following data packet:

- Packet ID: 0x48
- Payload Data: data bytes received, byte count equals number of bytes sent

USB host responds:

- ACK or NAK

3.13 SPI write no response

This command is the same as *SPI write* except that the received slave data is not returned to the USB host. If the received slave data is not of interest this command can be used instead of *SPI write* to save the time required by the USB host to acknowledge the received slave data.

Packet ID: 0x3A
Payload Data: data bytes to be sent

USB-2-X response: ACK or NAK

3.14 SPI timestamp

The SPI write timestamp package is an additional response that will be generated after each *SPI write* or *SPI write no response* command when the *write timestamp* flag is set in the *SPI init* command.

The USB-2-X transmits the timestamp in the following packet:

- Packet ID: 0x4A
- Payload data: 4 Bytes SPI write cycle start time, most significant byte first, resolution $170^2/_3$ µs, wrap-around after 203h 36m 47.75s

3.15 SPI read

This command is intended to read out the SPI slave receive buffer. The SPI slave receive buffer stores data which were retrieved from the SPI slave by USB-2-X in reaction to a falling edge on the SPI slave request line. *Enable Slave Request* must be set to one (see table 3.9) in order to enable USB-2-X to react on the SPI slave request line.

USB host transmits:

- Packet ID: 0x39
- Payload Data: no payload

USB-2-X responds:

- ACK or NAK

Then USB-2-X transmits the data received from the SPI slave in the following data packet:

- Packet ID: 0x49
- Payload Data: received SPI data (one datagram only)

USB host responds:

- ACK or NAK

If *Enable Slave Request* and *Push Slave Data* are both set to one (see table 3.9) this command is not required. In this case the data received from the SPI slave will be forwarded to the USB host automatically:

USB-2-X sends:

- Packet ID: 0x49
- Payload Data:
  - 4 Bytes reception time, most significant byte first, resolution $170^2/_3$ µs, wrap-around after 203h 36m 47.75s
  - received SPI data (up to 251 bytes incl. leading byte count, one datagram at a time)

Data packets of this type will only be sent from USB-2-X to the USB host when any command processing and responding has been finished, i.e., when USB-2-X waits for the reception of the next command start character STX.

In contrast to the other data packets it is neither required nor allowed to acknowledge this type of data packet by sending ACK or NAK. This implies that transmission of such a data packet cannot be repeated in case of failure to receive it error-free.

3.16 SPI init

This command adjusts SPI clock settings, further SPI related parameters and flushes the SPI slave receive buffer. The *SPI init* command must be executed before any other SPI command can be used.

Packet ID: 0x54
Payload Data:

	7	6	5	4	3	2	1	0
1st Byte	0	Base Clock[2:0]			Clock Polarity	Clock Phase	Clock Divisor [1:0]	
2nd Byte	0	Slave Select Delay [2:0]			Inter-Byte Delay [3:0]			
3rd Byte	SPI Enable	0	0	Half Duplex	Write Timestamp	Push Slave Data	EnableSlave Request	3V3

USB-2-X response: ACK/NAK

Parameter Name	Parameter Coding
Base Clock [2:0]	0: CPU clock frequency = 1.5 MHz 1: CPU clock frequency = 3.0 MHz 2: CPU clock frequency = 4.5 MHz (only allowed for 5V operation) 3: CPU clock frequency = 6.0 MHz (only allowed for 5V operation) 4: CPU clock frequency = 7.5 MHz (only allowed for 5V operation) **SPI clock frequency = CPU clock frequency / SPI clock divider**
Clock Polarity	0: SPI clock line idle level = low 1: SPI clock line idle level = high
Clock Phase	0: 1st clock edge latches bit 7, 2nd clock edge shifts out bit 6 onto the bus 1: 1st clock edge shifts out bit 7 onto the bus, 2nd clock edge latches bit 6
Clock Divisor [1:0]	0: SPI clock divider = 2 2: SPI clock divider = 32 1: SPI clock divider = 8 3: SPI clock divider = 128 **SPI clock frequency = CPU clock frequency / SPI clock divider**
Slave Select Delay [2:0]	Minimum delay after slave select going low until 1st byte / after last byte until slave select going high: **min_ss_delay [CPU clock cycles] = 6 + 3 * 2^Slave Select Delay** 0: min_ss_delay = 9 cycles 4: min_ss_delay = 54 cycles 1: min_ss_delay = 12 cycles 5: min_ss_delay = 102 cycles 2: min_ss_delay = 18 cycles 6: min_ss_delay = 198 cycles 3: min_ss_delay = 30 cycles 7: min_ss_delay = 390 cycles **Note**: Due to SPI hardware constraints the actual delay act_ss_delay varies: **min_ss_delay ≤ act_ss_delay ≤ min_ss_delay + SPI bit time** with **SPI bit time [CPU clock cycles] = SPI clock divider**
Inter-Byte Delay [3:0]	Delay between any two bytes of an SPI transfer: **byte_delay [CPU clock cycles] = 8 * 2^Inter-Byte Delay** for 1 ≤ Inter-Byte Delay ≤ 8 (0 means no delay) 0: byte_delay = 0 cycles (back-to-back byte transfer without delay) 1: byte_delay = 16 cycles (only for SPI clock divider 2 or 8) 5: byte_delay = 256 cycles 2: byte_delay = 32 cycles (only for SPI clock divider 2, 8 or 32) 6: byte_delay = 512 cycles 3: byte_delay = 64 cycles (only for SPI clock divider 2, 8 or 32) 7: byte_delay = 1024 cycles 4: byte_delay = 128 cycles 8: byte_delay = 2048 cycles **Note**: The delay after the first byte of a Slave Request transfer is up to 15 CPU clock cycles plus one SPI bit time longer than the delay between the following bytes. This is because the first byte of a Slave Request transfer contains the number of the following bytes and must be processed before the following bytes can be transferred.
SPI Enable	0: SPI hardware disabled 1: SPI hardware enabled
Half Duplex	1: Slave requests have priority over master writes (packet 0x3A) 0: write commands have priority
Write Timestamp	0: No timestamps for SPI writes will be generated. 1: After every SPI write, the exact start time of the write cycle will be send back as a type 0x4A package.
Push Slave Data	Only evaluated if Enable Slave Request = 1 0: Data received in reaction to SPI slave request must be read using *SPI read* command 1: Data received in reaction to SPI slave request are forwarded to the USB host automatically
Enable Slave Request	0: SPI slave request line is ignored, SPI slave cannot request a transfer 1: SPI slave can request USB-2-X to start a transfer by a falling edge on the SPI slave select line. The first byte received from the slave is interpreted as remaining byte count of this transfer.
3V3	0: logic high level of SPI bus lines is 5V 1: logic high level of SPI bus lines is 3.3V (only allowed for CPU clock frequency = 1.5 MHz or 3.0 MHz)

Table 3.9: SPI init parameters

- IIC cannot be used simultaneously with SPI. To re-enable IIC after SPI has been used, disable the SPI interface by executing an SPI init command with the *SPI Enable* bit (Bit 7 of data byte 3) set to zero. This will disable SPI (all SPI pins will be tri-stated then) and re-initialize IIC. After that, the IIC bit rate must also be set up again using the IIC bit rate command.
- The bit rates of all the other interfaces are calculated using a base clock value of 7.5MHz. So, when setting the base clock frequency to a value different than 7.5MHz, all bit rates of the other interfaces (IIC, LIN, RS485, and CAN) will be wrong. So, set the clock frequency back to 7.5MHz when disabling SPI to get all other bit rates right or use a base clock frequency of 7.5MHz when planning to use SPI simultaneously with LIN, RS485 or CAN.

3.17 RS485 bit rate

This command enables the RS485 interface and sets up its baud rate. It also disables the LIN interface, as RS485 and LIN cannot be used simultaneously (to re-enable LIN after RS485 has been used, the *LIN bit rate* command has to be used once again, as the *LIN bit rate* command also disables RS485).

USB host transmits:

- Packet ID: 0x55
- Payload Data: 1 Byte Bit rate: 0 = 2400 bit/s, 1 = 9600 bit/s, 2 = 19200 bit/s

USB-2-X responds:

- ACK or NAK

3.18 RS485 write

This command writes data to the RS485 interface. The response will not be sent until all data bytes have been written to the RS485 interface.

USB host transmits:

- Packet ID: 0x3B
- Payload Data: data bytes to be sent via the RS485 interface

USB-2-X responds:

- ACK or NAK

3.19 RS485 read

This command reads data from the receive buffer of the RS485 interface.

USB host transmits:

- Packet ID: 0x3C
- Payload data: no payload data

USB-2-X responds:

- ACK or NAK

Then, the USB-2-X device returns all data bytes that could be read from the receive buffer in the following packet:

- Packet ID: 0x4C
- Payload data: the data bytes that were read form the RS485 receive buffer

This packet will also be sent back when the receive buffer is empty (in this case with no payload data).

USB host responds:

- ACK or NAK

3.20 Firmware version

USB host transmits:

- Packet ID: 0xFF
- Payload Data: 1 Byte Reset-Flag (0=don't reset USB-2-X device, 1=reset USB-2-X device)

USB-2-X responds:

- ACK or NAK

Then USB-2-X transmits the read data in the following data packet:

- Packet ID: 0xF0
- Payload Data:
  - 1 Byte Reset-Flag (0=no reset executed since last 0xFF command, 1 otherwise)
  - up to 32 bytes of plain ASCII version information text

USB host responds:

- ACK or NAK

4 Firmware update and bootstrap loader

Firmware Updates can be executed using the bootstrap loader (BSL). The BSL is independent from the firmware, stored in a separate flash memory area and protected against erasure. The BSL is able to receive a firmware, to program it into the flash memory, to check it on integrity and to start it. The BSL is automatically started after Reset. Reset can be initiated by power-on (plugging the USB cable) or by the Firmware-Version command with Reset-Flag set to 1. The BSL expects Motorola S-record compliant data over the USB interface. Motorola S-records are made up of ASCII characters and are usually combined in files. One S-record corresponds to one line of text in the file. The BSL supports the following S-record syntax:

```
typedef struct {
  char[2]     type;             // e.g. "S1" (= 53h 31h)
  char[2]     count;            // e.g. "23" (= 32h 33h): 23h ASCII-Bytes follow
  char[4]     address;                  // e.g. "C400" (= 43h 34h 30h 30h): data storage address
  char[0-504] data;             // up to 252 data bytes
  char[2]     checksum;         // one's-complement of the modulo-256 sum over count, addr + data
} S_RECORD;
```

Each S-record is completed by line end character(s) not to be sent to the BSL. S-record types S0, S1 and S9 are supported (file format .s19). Each S-record is acknowledged with a status code by the BSL. Any positive status code (0...0x7F) corresponds to the version number of the BSL and indicates error-free reception and processing of the S-record. Any negative status code (0x80...0xFF) indicates an error according to table 3.10.

BSL status code (hexadecimal)	Meaning
0...0x7F	BSL version number; S-record received and processed error-free
0x80	Error: Firmware could not be started, is not available or corrupted
0x81	Error: wrong S-record type: only S0, S1 and S9 are supported
0x82	Error: wrong S-record address: firmware storage at this address is not allowed
0x83	Error: wrong checksum, S-record corrupted
0x84	Error: Erasing a flash memory page failed
0x85	Error: Programming a S1-record into the flash memory failed
0x86	Error: S-record contains illegal character(s)
0x87	Error: illegal S-record length

Table 4.1: Bootstrap loader status codes

S0 and S9-records are evaluated regarding their checksum and acknowledged with a status code.

The data field of S1-records contains the actual firmware bytes. Checksum and storage address of S1-records are evaluated and the related flash memory pages are erased if necessary. The received firmware bytes are programmed into the flash memory. Afterwards the record is acknowledged with a status code.

Starting the firmware is accomplished by an arbitrary firmware command (see 3 Definition of data packets). This is possible since each firmware command starts with the STX control character (see 2.2 Data packets). STX is recognized by the BSL as not being part of an S-record. STX causes the BSL to jump to the fix firmware entry point if a firmware is stored in the flash memory. The BSL is exit, and then the firmware receives the command bytes following after STX and processes the command accordingly. This way the firmware can be started also without prior update. After starting the firmware updates are possible only after a new Reset. In case there is no functioning firmware stored in the flash memory the STX control character is acknowledged with an error code and the BSL stays active.

5 Revision history

Rev. No.	Date	Author	Changes
0.01	2003-AUG-18	OK	first draft
0.02	2003-AUG-26	OK	ASCII codes of control characters added
0.03	2003-AUG-28	OK	LIN commands and version info added
0.04	2003-OKT-08	TG	LIN read: parameter *count of requested Bytes* added
0.05	2003-OKT-14	TG	LIN read: more error codes added
0.06	2003-OKT-20	TG	Firmware update added
0.07	2004-JAN-24	TG	CAN commands revised and expanded
0.08	2004-FEB-05	TG	LIN write expanded for periodic message repetition
0.09	2004-MAR-05	TG	CAN commands structure changed for faster execution; CAN Bit rate supports 750 instead of 800 kbps, CAN-Read error codes added/changed
0.10	2004-MAR-10	TG	CAN commands completely revised again
0.11	2004-APR-05	TG	SPI, Async-Data and commands overview added
0.12	2004-MAY-10	TG	Translation to English
0.13	2004-MAY-12	TG	SPI init changed
0.14	2004-MAY-19	TG	Minor corrections
0.16	2004-JUL-22	TG	CAN push mode added
0.18	2004-JUL-27	TG	JTAG commands and *SPI write no response* added
0.19	2004-AUG-12	TG	SPI init: changed Base Clock, Slave Select Delay and Inter-Byte Delay
0.20	2004-OCT-01	TG	new company address
0.21	2005-JAN-25	OK	RS485 commands added
0.22	2005-NOV-11	OK	New SPI features added
0.23	2009-JUN-20	SD	Minor changes