The eddyCAN interface on the TXx

May 31, 2016

Contents

1 Essentials

Communication with eddylab's Eddy Current driver is based on USB or the CAN-Bus. This document gives full insight into the eddyCAN-Protocol.

The eddyCAN-Protocol is closely related to the well known CANopen-Protocol. eddyCAN is a proprietary CAN-protocol based on CANopen - as it does not fulfill all (but many) mandatory specifications of the CANopen-Protocol. Nevertheless - all the naming conventions that are known from CANopen are used here as well. A fully compliant CANopen-interface will be available shortly.

2 CAN-Bus

2.1 Bitrates

The CAN-specification defines the physical medium and the transfer mechanisms on the bus. The eddyCAN-protocol uses the 11-bit identifier and supports bitrates up to 1 Mbit. Table 1 contains the available bitrates and their corresponding cable lengths.

Bitrate	Max. cable length
1 MBit/s	25m
500 kBit/s	50m
250 kBit/s	250m[1]
100 kBit/s	500m[1]
50 kBit/s	1000m[2]

Table 1: CAN bitrates and cable lengths.

2.2 Connector

The TX-Driver has a D-Sub 9 connector. Only two pins (2 and 7) of this connector are assigned to the CAN-Bus. The remaining pins are assigned to other functions - and are not n.c. → do not apply random voltages or loads! Table 2 shows the pin assignment.

Pin	Function	Comment
1	Output 1	switching output 0..5V
2	CAN-Low	
3	GND	optional for the CAN-Bus
4	Input 1	switching input 0..5V
5	Input 2	switching input 0..5V
6	GND	Gnd for the switching inputs and outputs
7	CAN-High	
8	Output 2	switching output 0..5V
9	n.c.	

Table 2: Pin assignment on the D-Sub 9

[1]For a bus length greater than about 200m the use of optocouplers is recommended. If opto-couplers are placed between CAN controller and transceiver this affects the maximum bus length depending on the opto-coupler's propagation delay.

[2]For a bus length greater than about 1km - bridge or repeater devices may be needed.

2.3 Termination

The twisted pair cable has to be terminated with 120-Ω on both ends. We recommend to use CAN-Connectors with integrated (switchable) resistors. These connectors facilitate the setup of fieldbus systems. The right connector in Figure 1 is the terminated (resistor is switched on) end of the bus. The left connector shows an intermediate connection on the bus (resistor is switched off).

Figure 1: CAN-Bus connectors with integrated termination.

3 eddyCAN protocol

As mentioned above - the eddyCAN-Protocol is a proprietary CAN-protocol based on the general CANopen communication profile. The components of the protocol are summarized as follows.

1. Communication objects and message types.
2. CAN identifier distribution
3. Device configuration
4. Network management

3.1 Communication objects and message types

This section gives an overview of the messages on the network and their functions. Basically there are four types of messages.

Network management (NMT): The purpose of these messages is to control the nodes on the network. NMT messages are used to initialize, start or stop a node. Further use is to monitor the presence of a node (supervision of the network). NMT messages are transmitted from a master to control the slaves on the network. The slaves do not confirm an NMT-message.

Process data object (PDO): These are used to transfer and receive realtime data. All of the 8 databytes in a CAN-frame can be used to exchange data (i.e. positions). A node that transmits data is called a producer (TPDO) and a node that receives data is called a consumer (RPDO). The TX-Driver is a PDO producer. The data within a PDO-message is the position of the sensor. The mechanisms that trigger a message transmission can be of different nature.

- The transmission can be triggered timer driven by a device-internal clock. This results in a constant sampling-rate and a predictable bus load. See **Update Time** and **Nr Of Nodes** in section 3.3.2.
- For low sampling-rates or snapshotted data the remotely triggered transmission will minimize the amount of needless bus load. The TX-Driver will transmit a message after the reception of a remote frame. The remote frame needs to carry the identifier of PDO3 (see table 3) and the required number of bytes to be received.
- For the purpose of remotely triggering synchronous TPDOs over the entire network a SYNC message can be sent. Every node on the network configured to transfer a PDO upon the reception of a SYNC message will do so. The identifier of a SYNC message is always 128/(80h) (see table 3). The datalength is always zero.

Databytes in PDOs: The TX-Driver exists as single- and dual-channel driver. Each with the option to transfer the position of an incremental encoder. This results in four possible interpretations of eight databytes in a CAN-frame. The following figures reflect the possible byteorders:

Byte0..3
Position 1

Figure 2: PDO structure (4 bytes) for a single-channel driver without encoder

Byte0..3	Byte4..7
Position 1	Position 2

Figure 3: PDO structure (8 bytes) for a dual-channel driver without encoder

Byte0..3	Byte4&5
Position 1	Encoder

Figure 4: PDO structure (6 bytes) for a single-channel driver & encoder

Byte0..2	Byte3..5	Byte6&7
Position 1	Position 2	Encoder

Figure 5: PDO structure (8 bytes) for a dual-channel driver & encoder

Datatypes in PDOs: The position-value in a PDO is always 4 bytes long except for a dual-channel driver & encoder (3 bytes in this case). The byte order is always LSB first (i.e. in a dual-channel message without encoder - byte 0 is the LSB of Position 1 and Byte 4 is the LSB of Position 2). The conversion to a fixed-point value in [mm] is as follows: interpret the 32-bit-value as integer and divide this value by 2^{27}.

The 3 byte position (dual-channel driver & encoder) is converted analogously: interpret the

24-bit-value as integer and divide this value by 2^{19}. The byte order is always LSB first (i.e. in dual-channel message with encoder - byte 0 is the LSB of position 1, byte 3 is the LSB of position 2 and byte 6 is the LSB of the encoder value).

The encoder is a 16-bit-integer value which reflects the native data-representation of incremental encoders.

Service data object (SDO): SDOs are used to configure a device. Every device has a set of parameters over a defined index range. Entries up to a length of four bytes can be accessed with SDO read and write commands. SDOs are not used to transfer realtime data. An SDO is always followed by a confirmation message.

Special messages: SYNC and Error For the purpose of synchronizing events on the network - a SYNC message can be sent from the master. An interesting event on a TX-Driver network is the synchronous sampling of all positions. The transmission of the simultaneously sampled data will be accomplished in sequential fashion (with PDOs).

Furthermore - errors on a node (TX-Driver) are indicated in error messages. Most errors on a network result from communication or protocol errors.

3.2 CAN identifier distribution

The eddyCAN protocol (same as in CANopen) is based on the 11-bit CAN-identifier. The 11-bit identifier is split up into a functional part and a node-id part. The node-id has to be unique for every device on the network. The functional part defines the type of message or service and is applicable for every device of the same type. This combined identifier (known as COB-ID in CANopen) will be unique for every CAN-message on the network - for the case that all node-ids are unique. A device's node-id has to be defined by the end-user (system integrator).

The node-id of eddylab's TX-Driver can be defined on the CAN-Bus or with eddylab 2.0.1 and newer versions.

The combination of functional part (4-bit) and node-id part (7-bit) in the 11-bit-identifier results in a defined connection set. The 7-bit node-id allows 127 slave-nodes (physical) on the network. Table 3 reflects the identifier distribution. The highest priority identifier is 0. The higher the identifier of a message the lower its priority.

Message	Identifier(s)	Comment
NMT	0	broadcast administration
SYNC	128(80h)	broadcast synchronisation
Error	128(80h) + node-id	error indication
PDO1(tx)	384(180h) + node-id	cyclic transmission (internal trigger)
PDO2(tx)	640(280h) + node-id	transmission after sync request
PDO3(tx)	896(380h) + node-id	transmission after remote request
SDO(tx)	1408(580h) + node-id	transmission of configuration data
SDO(rx)	1536(600h) + node-id	reception of configuration data
Heartbeat	1792(700h) + node-id	transmission of the heartbeat

Table 3: Identifier Distribution on the TX-Driver

Example: A device configured with a node-id of 2 will transmit the PDO1 with a CAN-identifier of 386(182h). If the same device receives a SYNC-request (128(80h)) - PDO2 with a

CAN-identifier of 642(282h) will be transmitted. The content of each message will always be the position(s) of the TX-Driver+Sensor(s). The major difference between PDO1 and PDO2 is the kind of triggering the messages.

3.3 Device Configuration

All the configurable entries (or objects) on the TX-Driver are organized within a strictly defined list. The entries in this list can be addressed with a 16-bit index and a 8-bit subindex. Read and write access on the index is accomplished with SDOs (see section 3.1). Table 4 reflects the available configuration entries. The index starts at 2000h - this corresponds with the manufacturer specific profile area in the CANopen object dictionary.

Index(hex)	Name	Type	Access	Subindex(hex)
2000	Node-ID	Unsigned8	RW	-
2001	Bitrate Index	Unsigned8	RW	-
2002	Nr Of Nodes	Unsigned8	RW	-
2003	Update Time	Unsigned16	RW	-
2004	Can Config Register	Unsigned16	RW	-
2005	Basic Config Register	Unsigned16	RW	-
2006	Filter	Unsigned8	RW	1;2
2007	Comparator	Unsigned32	RW	1;2
2008	Counts Per Revolution	Unsigned16	RW	-
2009	Encoder Resolution	Unsigned16	RW	-
200A	Reset Encoder	-	W	-

Table 4: Configuration Index on the TX-Driver

3.3.1 Read and Write Access

The configuration entries can be accessed with SDOs. The TX-Driver can be interpreted as a provider of its configuration data. Data exchange (read or write) is always a confirmed service and is never initiated by the TX-Driver. The TX-Driver on the network responds (confirmation) with data or accepts data. That is: for the case that data shall be written to the device - the request contains data. The actual procedure (read or write) is defined within the first byte of the CAN-frame (Command byte). An SDO is always 8 bytes long and has a defined structure as shown in figure 3.3.1.

Byte0	Byte1..2	Byte3	Byte4..7
Command	Index	Sub-Index	Data

Figure 6: Basic structure of a SDO

Byte 0: is the command byte. Table 5 lists the command byte and the corresponding function.

Bytes 1..2: are the index bytes. These address the 16-bit index. The byte order is LSB first. This means that the index 2001h will be transferred as byte 1: 01h and byte 2: 20h.

Byte 3: is the subindex. If no subindices are available on an index - the subindex is always 0. If no subindex is available on an index and a subindex entry is requested - the result will always be the entry at 0 (no subindex).

Bytes 4..7: are the data bytes. Data with a maximum length of 4 bytes can be transferred within one SDO-frame. The byte order is LSB first for data sets greater than one byte. Each entry has a specific length. This length is given in the index descriptions (section 3.3.2) and in table 4.

Direction as seen from TX	Command	Function
→[3]	22h	write data to device (4 bytes max)
←	60h	confirmation after writing data to device
→	40h	request data from device
←	43h	confirmation with 4 bytes data
←	4Bh	confirmation with 2 bytes data
←	4Fh	confirmation with 1 byte data
←	80h	error

Table 5: Command bytes and functions

3.3.2 Index descriptions

Node-ID (2000h/1byte): is the unique 7-bit-identifier on the network (value range 1..127). A restart is required after modifying this value to become valid. This parameter can also be modified with eddylab.

The Bitrate Index (2001h/1byte): defines the bitrate on the network. All nodes on the network need to be configured with the same bitrate. Table 6 lists the bitrate index and the corresponding bitrate. A restart is required after modifying this value to become valid. This parameter can also be modified with eddylab.

Bitrate Index (dec)	Bitrate
0	50kBit/s
1	100kBit/s
2	250kBit/s
3	500kBit/s
4	1MBit/s

Table 6: Bitrate Index

A Nr Of Nodes (2002h/1byte): can be defined to facilitate the efficient usage of the CAN-Bus (for PDO1 - device internal clocked transmission of position data). This function is only useful for networks which are strongly dominated by TX-Drivers (i.e. this functions assumes a very low busload that is caused by other 'Not-TX-Drivers'.) The purpose is to generate a 50%-busload - by defining the number of participating TX-Drivers (network-wide) on each TX-Driver. The TX-Drivers auto-generate a transmission-rate that fulfills a 50%-busload condition. The value range is 1..127. This function becomes active by setting the following parameter (**Update Time 2003h**) to a value of 65535.

The Update Time (2003h/2bytes): defines the time in [ms] that passes between subsequent updates on the CAN-Bus (PDO1). The value range is 1..65534 ms. A value of 0 inhibits the timer driven transmission of position data. This setting is needed if only the remotely driven (PDO3) or sync driven (PDO2) transmission of position data is desired. A value of 65535 activates the 50%-busload function.

[3]The → corresponds with the incoming message.

The Can Config Register (2004h/2bytes): configures three CAN-parameters on the TX-Driver. If Autostart Operational is set - the device will enter the Operational state after Boot-Up. Sync Encoder additionally transmits the position of a connected encoder. If Heartbeat is set - the TX-Driver will periodically transmit a Heartbeat-message. Table 7 lists the register value and the corresponding functionality.

Value	Autostart Operational	Sync Encoder	Heartbeat
0			
1	×		
2		×	
3	×	×	
4			×
5	×		×
6		×	×
7	×	×	×

Table 7: Can Register Functions

The Basic Config Register (2005h/2bytes): configures the encoder-interface. If Rotary Encoder is set - the device assumes a rotary encoder. If not set - the TX-Driver assumes a linear encoder. If desired the counting direction can be reversed with Reverse Encoder Direction. Table 8 lists the register value and the corresponding functionality.

Value	reserved	Rotary Encoder	Reverse Encoder Direction
0			
2		×	
4			×
6		×	×

Table 8: User Register Functions

The Filter Index (2006h/1byte): defines the edge frequency for every channel. The TX-Driver can be configured with five edge frequencies. A subindex of 1 addresses channel 1 and a subindex of 2 addresses channel 2. Table 9 lists the filter index and the corresponding edge frequency.

Filter Index (dec)	edge frequency
0	no Filter is active
1	10 Hz
2	100 Hz
3	1 kHz
4	10 kHz

Table 9: Selectable Filters on the TX-Driver

A comparator value (2007h/4bytes): can be defined for every channel. The value range is 0..1. A comparator value of 0.5 means that the output will be switched at 50% of the sensor's measuring range. If the sensor's position is above 50% the output will be high - otherwise low. For single channel devices both comparator values are assigned to one channel. This permits the definition of a low and a high threshold for one channel. The subindex

addresses the respective channel. The datatype is a fixed-point-value between 0..1. The value is calculated as follows. Multiply the comparator value by 2^{30}. This 32-bit-integer value has to be transferred to the TX-Driver.

Example: If the desired comparator value is 0.5 - the integer-value is $0.5 \times 2^{30} = 536870912$.

Counts per Revolution (2008h/2bytes): defines the number of increments of a rotary encoder (4x encoding). The value range is 1..65535.

The Encoder Resolution (2009h/2bytes): is the resolution of a linear encoder (4x encoding) in $[nm]$. A typical value is 100 nm.

Reset (200Ah): the encoder (linear) if an over- or underrun occurred. No data is transferred.

3.3.3 Aborted Transfers

If an index cannot be accessed or does not exist the confirmation message from the TX-Driver contains a error code. The command byte of a message containing an error code is always 80h (see Table 3). The error code of an aborted transfer is 4 bytes long. The supported codes are:

Code (hex)	meaning
05040001h	Command unknown
06020000h	Index does not exist
06090011h	Subindex does not exist
06090030h	Range exceeded
06010001h	Attempt to read a write only object
06010002h	Attempt to write a read only object
08000000h	General error

Table 10: Codes for aborted transfers

3.4 Network management (NMT)

Every node (slave) can be in three different states after Boot-Up. These are Pre-Operational, Operational and Stopped. The actual state is controlled from the master. Each state offers a specific functionality.

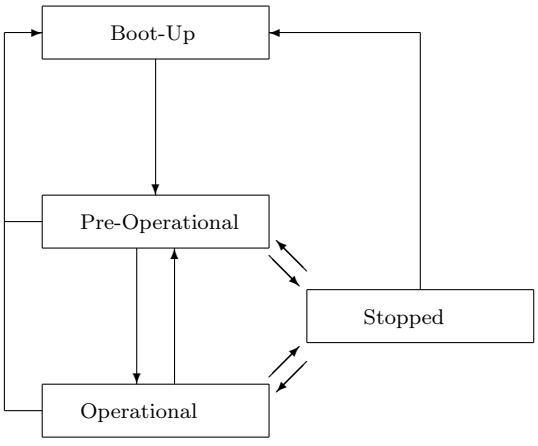

Figure 7: NMT-States

The Pre-Operational state is used for device configuration. The device can be configured with SDOs. The Pre-Operational state is the default state after Boot-Up. This default state can be modified to Operational in the CAN-Config-Register (2004h).

In Operational a node transfers PDOs. These can be triggered internally by a timer, by remote frames or by a sync request. The kind of trigger is configured with the parameter **Update Time (2003h)**. If this value is zero - the TX-Driver will only react on sync and remote messages. If this value is 65535 - the 50%-busload function is active. All values between 1..65534 define the time in [ms] that passes between subsequent messages. Configuration with SDOs is not possible.

A Stopped device does not provide SDOs or PDOs. This state only reacts on NMT-messages to other states.

The state of a node is controlled with NMT-messages. The basic layout of a NMT-message is as follows: The identifier of the message is always 0. The message is 2 bytes long. The first

id	Byte0	Byte1
0	Command	Node-id

Figure 8: Structure of a NMT message

byte (byte 0) is the command to the desired state. The second byte (byte 1) is the id of the desired node. If the second byte is 0 - all nodes are addressed. This enables the simultaneous control of all nodes with one command. The commands are listed in table 11.

Command	Command-byte (hex/dec)
Start node	01h/01
Stop node	02h/02
Set Pre-Operational	80h/128
Reset node	81h/129

Table 11: NMT-Commands

3.4.1 Heartbeat

The TX-Driver can be configured to produce a periodic heartbeat-message. This message can be used to monitor the presence of a node. The period of this message is 2 seconds. The message is one byte long and contains the state of the heartbeat producing node. The

id	Byte0
1792+node-id	State

Figure 9: Structure of the Heartbeat message

states are listed in table 12. The TX-Driver can be configured to produce a heartbeat in the CAN-Config-Register (2004h). The heartbeat is available in the states Operational, Pre-Operational and Stopped. A heartbeat message is never confirmed by the master. Regardless of the configuration in the CAN-Config-Register - one heartbeat-message is transmitted after Boot-Up. The purpose of this Boot-Up message is to indicate a newly connected device on the network.

State	byte 0 (hex/dec)
Boot-Up	0h/0
Stopped	04h/04
Operational	5h/5
Pre-Operational	7Fh/127

Table 12: States in the heartbeat-message

3.5 Error Indication

The TX-Driver transmits error messages if those occur. The identifier of an error message is always 80h+node-id (i.e. a device configured with a node-id of 2 will transmit an error message with an CAN-identifier of 130(82h)). The error code is a 16-bit value (2 bytes). The supported codes are:

Code (hex)	meaning
1000h	Generic Error
8120h	CAN Passive
8200h	Protocol Error

Table 13: Error codes