Embedded system solutions
Emsiso

Emsiso d.o.o.
Zagrebška cesta 20
SI-2000 Maribor
Slovenia

tel. +386 2 46 129 07
fax. +386 2 46 129 08
www.emsiso.com

Member of
Seidel Electronics
www.seidel.at

SEIDEL GROUP

eDrive Basic CAN control

History

Version	Author	Description	Date
1.0	Gregor Kosic	Initial version	07.09.2015
1.1	Zdenko Mezgec	Extended description + examples	26.04.2018

DESCRIPTION

This document describes basic data flow on CAN bus for controlling the drive.

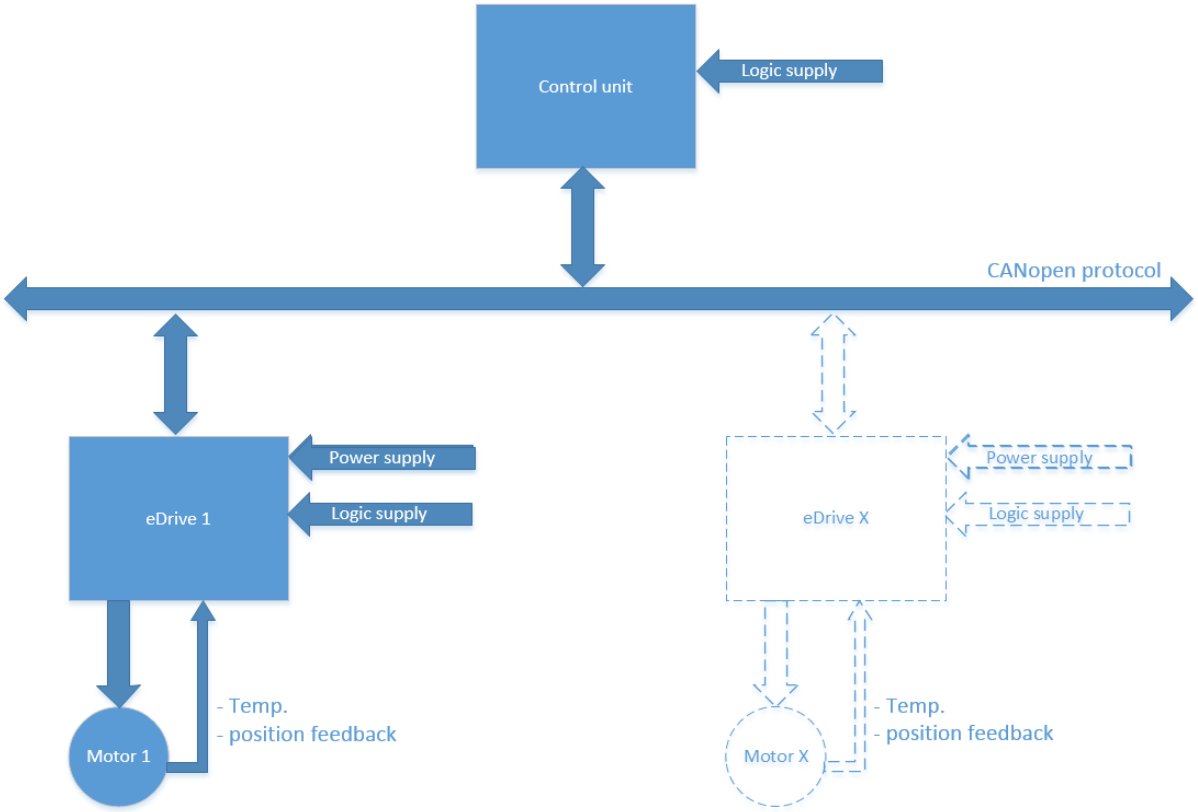
LEGAL REGULATIONS

Standards:

Nr:	Standard	Description	Issued
1	ISO 11898-1	Road vehicles – Controller area network (CAN) – Part 1: Data Link Layer	
2	ISO 11898-2	Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit	
3	ISO 11898-3	Road vehicles – Controller area network (CAN) – Part 3: Low-speed, fault-tolerant, medium-dependent interface	
4	CiA301 Version: 4.2.0	CANopen application layer and communication profile	February 2011
5	CiA402 Version: 3.0.0.	CANopen device profile for drives and motion control	December 2007
6	SDO example	http://www.canopensolutions.com/english/about_canopen/device_configuration_canopen.shtml	April 2018

DESCRIPTION OF REQUIREMENTS

1.1 Connections for supply and communication



1.2 Communication and protocols

The data link layer of CANopen, can only transmit short packages consisting of an 11-bit id, a remote transmission request (RTR) bit and 0 to 8 bytes of data. The CANopen standard divides the 11-bit CAN frame id into a 4-bit function code and 7-bit CANopen node ID. This limits the number of devices in a CANopen network to 127 (0 being reserved for broadcast).

In CANopen the 11-bit id of a CAN-frame is known as communication object identifier, or COB-ID.

Contents of an 11-bit CANopen frame:

	CAN-ID	RTR	Data length	Data
Length	11 bits	1 bit	4 bits	0-8 bytes

The data frame with an 11-bit identifier is also called "base frame format".

The default CAN-ID mapping sorts frames by attributing a function code (NMT, SYNC, EMCY, PDO, SDO...) to the first 4 bits, so that critical functions are given priority.

	Function code	Node ID
Length	4 bits	7 bits

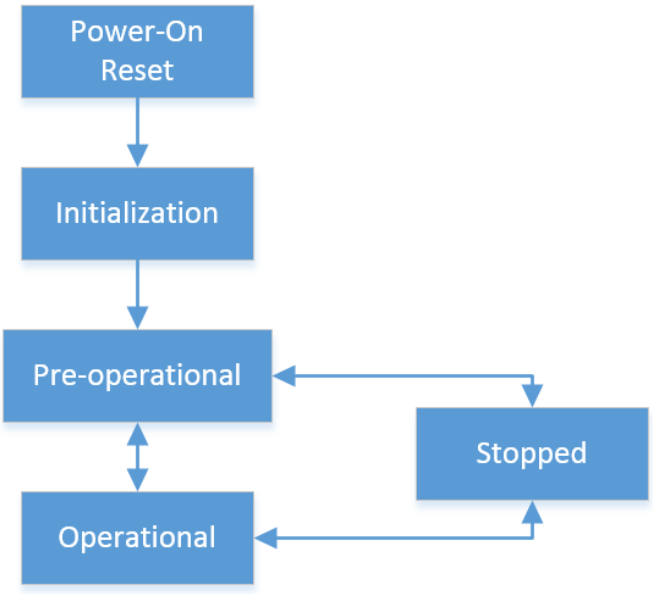
Different kinds of communication models are used in the messaging between CANopen nodes.

In a **master/slave** relationship, one CANopen node is designated as the master, which sends or requests data from the slaves. The NMT protocol is an example of a master/slave communication model.
Slave is emDrive motor controller

A **client/server** relationship is implemented in the SDO protocol, where the SDO client sends data (the object dictionary index and subindex) to an SDO server, which replies with one or more SDO packages containing the requested data (the contents of the object dictionary at the given index).
SDO server is emDrive motor controller

A **producer/consumer** model is used in the Heartbeat and Node Guarding protocols. In the *push-model* of producer/consumer, the producer sends data to the consumer without a specific request, whereas in the *pull model*, the consumer has to request the data from the producer.
Normally, emDrive is in consumer mode.

1.3 Network management (NMT) state of slave



Upon power-up eDrive (CANopen slave node) comes out of »power-on reset« and goes into initialization. It initializes the entire application, CAN/CANopen interfaces and communication. At the end of the initialization the node tries to transmit boot-up message. As soon as it is transmitted seuccessfully, the node switches to Pre-operational state.

Using the NMT Master message, an NMT Master can switch individual nodes or all nodes back and forth between the three major states: Pre-operation, Operational and stopped.

The NMT protocols are used to issue state machine change commands (e.g. to start and stop the devices), detect remote device bootups and error conditions.

The **Module control protocol** is used by the NMT master to change the state of the devices. The CAN-frame COB-ID of this protocol is always 0, meaning that it has a function code 0 and node ID 0, which means that every node in the network will process this message. The actual node ID, to which the command is meant to, is given in the data part of the message (at the second byte). This can also be 0, meaning that all the devices on the bus should go to the indicated state.

NMT request:

COB-ID (11bits)	*Data byte 0*	*Data byte 1*
0x000	*Requested state*	*Addressed node*

NMT Command (Requested state):

0x01 = start remote node
0x02 = stop remote node
0x80 = enter pre-operational
0x81 = reset node
0x82 = reset communication

1.4 SDO Messages

The SDO protocol is used for setting and reading values from the object dictionary of the remote device.

1.4.1 SDO download (setting)

The message for the Initiate SDO Download service (write data) is structured as follows:

Command byte 0x2B	OD main-index	OD sub-index	Data (max. 4 bytes)

The SDO server responds with protocol byte 0x60:

Command byte 0x60	OD main-index	OD sub-index	Empty (4 byte)

Example:
An SDO download to the OD entry [6071], with which the tarqet torque is to be set to value 300 (as an SIGNED16 value, i.e. 0x01 2C), therefore appears as follows:

2B	71 60	00	2C 01 00 00

The node (SDO server) then acknowledges successful completion with the message

60	71 60	00	00 00 00 00

CAN ID	Length	DATA
601	8	2B 71 60 00 2C 01 00 00
581	8	60 71 60 00 00 00 00 00

1.4.2 SDO Upload (reading)

The message for the Initiate SDO Upload service (reading data) is structured as follows:

Command byte (0x40)	OD Main Index	OD Sub Index	Empty (4 bytes)

The SDO server responds with:

Command byte 0x4B	OD Main Index	OD Sub Index	Data (4 bytes)

Example:
The SDO server must always respond to the typical request to read out target torque (identity object [6071]) – value 300)

40	18 10	01	00 00 00 00

Response

4B	18 10	01	19 03 00 00

CAN ID	Length	DATA
601	8	40 71 60 00 00 00 00 00
581	8	4B 71 60 00 2C 01 00 00

For segmented writes/read check further information/examples at:

http://www.canopensolutions.com/english/about_canopen/device_configuration_canopen.shtml

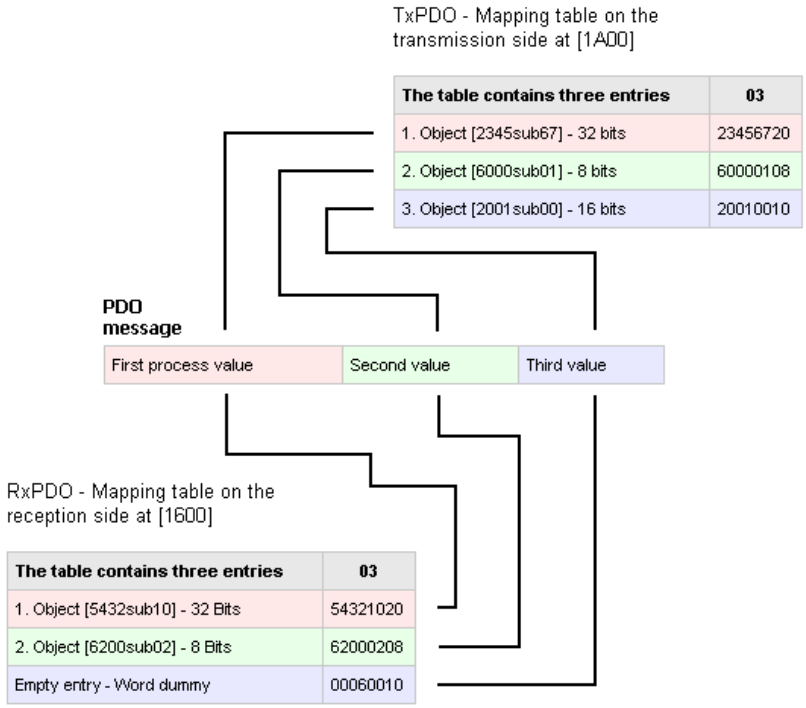
1.5 PDO

A CAN message that contains process data is called PDO (**"Process Data Object"**). Transmission of PDOs is only possible in the "Operational" state. PDOs have no fixed format. The content of a PDO can also not be readily interpreted. Both the transmitter and the receiver know how the content of a PDO is to be interpreted. The so-called **"PDO mapping"** describes which individual process variables in the data field of a PDO are transmitted, how they are arranged and which data type and length they have. Therefore the content and meaning of the data field of each defined PDO is described in the form of a PDO-mapping record inside the object dictionary both on the transmit and on the receive side.

The PDO producer composes the data field of a PDO in accordance with its TxPDO mapping. For this it takes the current data of the variables to be transmitted from its object dictionary and copies these into the data field of the PDO before the CAN message (PDO) is sent.
The same happens on the consumer side: on the basis of the RxPDO mapping record, the data bytes of the received PDOs are copied into local object dictionary entries and thus generally device-specific actions are triggered.

But now back to PDO mapping. The principle of the arrangement (mapping) of process variables is shown in the following (the variables are available in the form of object dictionary entries in the application profile). The mapping of the individual process variables in the data field of a PDO is described in the form of a table. This is also given as an object dictionary entry, namely for every transmit- and receive-

PDO in [16xx] or [1Axx]. These tables, and therefore the mapping of the process variables in the data field of a PDO can be configured via SDO write accesses.



In this example there are exactly two object links: from object (process variable) [2345sub67] of the PDO producer to object [5432sub10] of the PDO consumer and from object [6000sub01] of the producer to object [6200sub02] of the consumer. The third transmit object, [2001sub00] is not evaluated on the receiver side and is therefore covered up with a so-called dummy object.

Procedure to change desired TPDO is as followed

(example shown from emDrive Configurator tool using SDO writes for changing:

- TPDO number 1,
- mapping object 0x3641 subindex 3
- mapping object 0x3641 subindex 4
- sync transmission type
- Device node id is currently value 2

Notice: before mapping any object to PDO, user must check in provided eds file if object can be mapped to PDO.

1. Disabling PDO communication for TPDO 1
    a. COB-ID (0x1800-1) set to value 0x80000000
        i. Example: for TPDO 2 we would change index 0x1801

2. Disabling PDO mapping for TPDO 1
    a. Number of entries (0x1A00-0) set to value 0
        i. Example: for TPDO 2 we would change index 0x1A01

3. Mapping object 1 index 0x3641 subindex 3 which size is 16 bits
    a. PDO Mapping entry (0x1A00-1) set to value 0x36410310
        i. (0x INDEX(2B) SUBINDEX (1B) SIZE_in_bits (1B))

4. Mapping object 2 index 0x3641 subindex 4 which size is 16 bits
    a. PDO Mapping entry (0x1A00-2) set to value 0x36410410

5. Set number of mapped objects
    a. Number of entries (0x1A00-0) set to value 2 (number of objects)

6. Setup transmitt communication type
    a. Transmission type (0x1800-2) set to value 1 (sync)
    1 - sync (TPDO will be sent for every sync)
    254 - async with time specific in ms. TPDO will be sent automatiically by device. In case user chooses 254-async then there is additional parameter that needs to be set
        i. Event timer (0x1800-5) set to value 4000 (4 sec for example)

7. Setup COBID and enable PDO
    a. COB-ID bit meaning

10	9	8	7	6	5	4	3	2	1	0
PDO number			1	NODE_ID						

    b. COB-ID (0x1800-1) set to value 0x182 (0b00110000010)
        i. PDO number 1 and NODE_ID 2-device node id
        ii. Notice: By default device will independently change COBID whenever user changes node id of device

8. Save parameters
9. Reset device
10. Set device in operational mode and device is ready to send TPDO according to transmission type settings

1.5.1 Default PDO settings

Receive PDO 1 (data from control unit to eDrive)

byte							
0	1	2	3	4	5	6	7
Control Word		Target Velocity				Target Torque	

1.5.2 Receive PDO 2 (data from control unit to eDrive)

byte							
0	1	2	3	4	5	6	7
Target Position				Res.	Res.	Res.	Res.

1.5.3 Transmit PDO 1 (data from eDrive to control unit)

byte

0	1	2	3	4	5	6	7
Status Word		Position Actual Value				Torque Actual val	

1.5.4 Transmit PDO 2 (data from eDrive to control unit)

byte							
0	1	2	3	4	5	6	7
Controller temp	Motor temp	DC Link Voltage		Logic Power Supply Voltage		Current Demand	

1.5.5 Transmit PDO 3 (data from eDrive to control unit)

byte							
0	1	2	3	4	5	6	7
Motor current actual value		Electrical angle		Phase A Current		Phase B Current	

1.6 Master on power up sequence:

1. Wait for boot-up message from slave.
   Slave (eDrive) is by default configured to node id = 10.
   Slave message:

COB-ID (11bits)	*Data byte 1*
0x700+NodeID	*0x00*

   Note: other bytes not send

2. Check vendor ID (sdo read object 0x1018, 1 -> eDrive response 0x3C6)

   Sdo read command from master (control unit):

COB-ID (11bits)	*Command byte*	*Obj. Index (2 byte)*	*Obj. sub-index (byte)*	*Data (4bytes)*
0x600+NodeID	*0x40*	*0x1810*	*0x01*	*0*

   Sdo read response from slave (eDrive):

COB-ID (11bits)	*Command byte*	*Obj. Index (2 byte)*	*Obj. sub-index (byte)*	*Data (4bytes)*
0x580+NodeID	*0x43*	*0x1810*	*0x01*	*0x19030000*

3. Send PDO to enable PWM (setting control word to 6, 7, 15)

COB-ID (11bits)	*Data (default slave RPDO1)*
0x200+Node-ID	*06 00 xx xx xx xx xx xx*

COB-ID (11bits)	*Data (default slave RPDO1)*
0x200+Node-ID	*07 00 xx xx xx xx xx xx*

COB-ID (11bits)	*Data (default slave RPDO1)*
0x200+Node-ID	*0F 00 xx xx xx xx xx xx*

4. Periodically:
   a. send sync (each clave (eDrive will response with three TPDOs)

   master broadcast sync message:

COB-ID (11bits)	*Data byte 0*
0x080	*0x00*

   Other data bytes are not transmitted.

   Slave response:

COB-ID (11bits)	*Data (default slave TPDO1)*
0x180+Node-ID	*xx xx xx xx xx xx xx xx*

COB-ID (11bits)	*Data (default slave TPDO2)*
0x280+Node-ID	*xx xx xx xx xx xx xx xx*

COB-ID (11bits)	Data (default slave TPDO3)
0x380+Node-ID	xx xx xx xx xx xx xx xx

COB-ID (11bits)	Data (default slave TPDO4)
0x480+Node-ID	xx xx xx xx xx xx xx xx

b. master transmits PDO-for controlling

COB-ID (11bits)	Data (default slave RPDO1)
0x200+Node-ID	0F 00 xx xx xx xx xx xx

COB-ID (11bits)	Data (default slave RPDO2)
0x300+Node-ID	0F 00 xx xx xx xx xx xx

5. To disable PWM

COB-ID (11bits)	Data (default slave RPDO1)
0x200+Node-ID	06 00 xx xx xx xx 00 00

6. To disable drive

Broadcast NMT request go to 'Pre-operational'