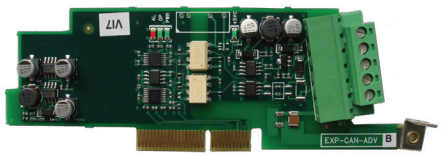
1S5F23, Manual EXP-CAN/DN-ADV
-IT/EN (rev 0.8 - 5-12-2022)

Instruction manual

EXP-CAN/DN-ADV
CANopen / DeviceNet interface expansion card



Contents

Caution

Use only the supplied screws!

Reinforced insulation

PELV (Protective Extra Low Voltage) EN 61800-5-1.

Introduction

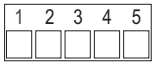
This manual describes the EXP-CAN/DN-ADV option card aimed at connecting the ADV200 series Drives to CANopen or DeviceNet networks. It is possible to use only one field bus expansion card per Drive.
This manual is addressed to desing engineers and technicians responsible for the maintenance and the commissioning of CANopen and DeviceNet systems.
A CANopen and DeviceNet basic knowledge is therefore required; for further information see the following manuals:

- CANopen CAL-Base COMMUNICATION PROFILE for Industrial Systems; CiA Draft Standard 301 Version 4.2 Date 13 February 2002 by CAN in Automation e. V.
- DeviceNet Specifications. Volume 1 - DeviceNet Communication Model and Protocol (Issued by ODVA).
- DeviceNet Specifications. Volume 2 - DeviceNet Device Profiles and Object Library (Issued by ODVA).

Mounting

Refer to ADV200 Quick Start up manual, chapter "**Installation of optional cards": the card must be inserted on slot 3**.

Connections

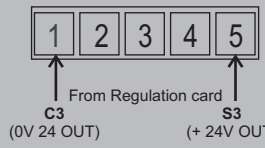
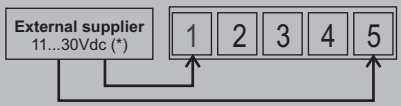


Wire sizes: 0.2 ... 2.5 mm^2 (AWG 24 ... 12)

For the Bus connection use a shielded loop, type as stated by the CANopen or DeviceNet specification.

The Bus connection is provided via a shielded loop (**type as stated by the CANopen or DeviceNet specification**) to be placed far from the power cables, with a minimum distance of 20 cm. The cable shielding must be continuous and grounded at a single point.

In addition, the unipotential connection of ADV200 CAN bus nodes must be ensured with card terminal 1 (V- / CAN_GND).

PIN	DeviceNet	CANopen	Function	Max
			BUS terminal : allows to connect the card to the CANopen or DeviceNet network	
			INTERNAL SUPPLY (Connection not insulated)  **EXTERNAL SUPPLY (Connection optocoupled)** 	
			(*) The supplier size have to be according to the used bus specification (CANopen or DeviceNet). Card absorption is 30 mA@24V	
1	**V-**	**CAN_GND**	Ground / 0V / V-	**0V**
2	**CAN_L**	**CAN_L**	CAN_L busline (dominant low)	**-**
3	**DRAIN**	**CAN_SHLD**	CAN shield	**-**
4	**CAN_H**	**CAN_H**	CAN_H busline (dominant high)	**-**
5	**V+**	**CAN_V+**	CAN external positive supply (dedicated for supply of transceiver and optocouplers)	**11 ... 30V**

Important!

Note on terminating resistor :

The first and last node of the CAN line must have a 120 ohm resistance between pins 2 and 4.

Caution

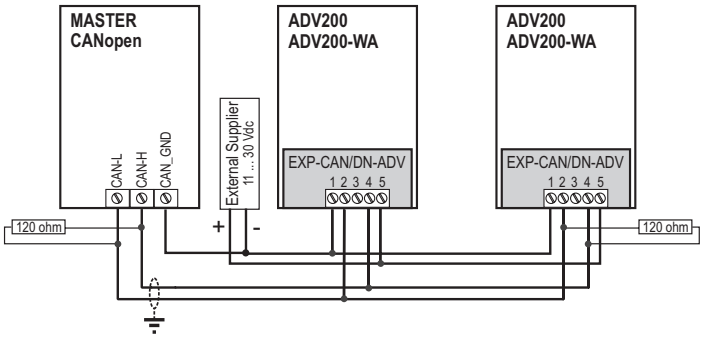
Terminals 1-5 must be powered:

- if using an external power supply the mains is galvanically isolated,

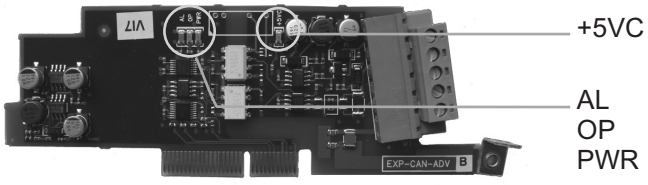
- if derived from the regulation card (terminals C3/S3, +24Vout/0V24 out) the mains is NOT galvanically isolated. The maximum available current value must not be exceeded (total max = 150 mA @ 24V), including any other expansion cards.

- on CANopen networks in general, the CAN_GND connection must apply to all participating nodes unless the CANopen network is completely galvanically isolated. If a node (master or slave) does not have the CAN_GND connection, or if the connection is not used, the user must ensure maximum rejection of line noise for all network participants.

The connection among the single cards is performed with a shielded cable as shown in the following figure.



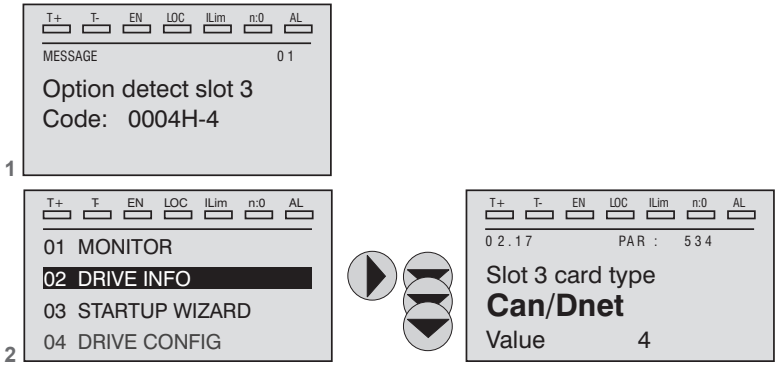
Leds



LEDs		CANopen	DeviceNet
AL	**(red)**	ON: the CANopen interface is in alarm condition	DeviceNet connection status, see next table
OP	**(green)**	ON: the CANopen interface is in Operational condition	
PWR	**(green)**	The led is ON when the expansion card is powered and active	
+5VC	**(green)**	The led is ON when the optoinsulated CAN node is correctly powered	

OP	AL	Meaning
ON	ON	Card power-up
BLINK	BLINK	Self test and Duplicate MAC-ID check is running
BLINK	OFF	Master configuration and/or I/O Polling wait not active
ON	OFF	I/O Polling active, operative status
OFF	BLINK	Minor fault (DUP MAC-ID fail, bus-off, bus-loss)
OFF	ON	Major fault (configuration error, internal error)
OFF	OFF	DeviceNet not configurated

Optional card recognization



1 - At power-on, the drive recognizes the presence of optional card in the expansion slot 3, this message is shown on the display.

2- On 02 DRIVE INFO menu, select the PAR 534 **Slot 3 card** type to read the recognized card type.

Value	Description	Card type
0	None	-
4	Can/Dnet	EXP-CAN/DN-ADV
255	Unknown	-

1.0 CANopen interface

CANopen is a communication profile for CAL-based industrial systems. The reference document is the CANopen CAL-Base COMMUNICATION PROFILE for Industrial Systems; CiA Draft Standard 301 Version 4.2 Date 13 February 2002 by CAN in Automation e. V.
The drive also implements the DS402 profile according to the CANopen Device Profile Drives and Motion Control v4.02 specification.
The CAN protocol (ISO 11898) is CAN2.0A with an 11-bit identifier.
The integrated CANopen interface is developed as a “Minimum Capabilty Device”.
The data exchange is cyclic; the Master unit reads the Slave input data and writes the Slave output data.

1.1 CANopen functions

This chapter describes the controlled functions of the CANopen communication profile.

Main features:

1) The “Minimum Boot-up” is managed; the “Extended Boot-up (CAL)” is not managed.
2) The SYNC function is implemented.
3) The PDO asynchronous assignment and RTR are managed.
4) The Node Guarding and HearthBeat protocols are managed.
5) The emergency message is managed (“EMERGENCY”).
6) The Dynamic ID distribution function (DBT slave) is not managed.
7) A “Pre-Defined Master/Slave connection” is implemented to simplify the Master tasks during the initialization phase. “Inhibit-Times” (in units of 100 uS) can be modified up to a value of 1 min.
8) The high-resolution synchronization is not supported.
9) “TIME STAMP” is not managed.
10) On the access of the structured parameters, the OFFhex option subindex (access to the whole object) is not managed.
11) In order to obtain a higher efficiency level, only the “Expedited” data transfer (max. 4 Bytes) of the SDO services is managed.

1.1.1 Pre-defined Master/Slave Connection

The “Pre-defined Master/Slave connection” allows a peer-to-peer communication between one Master and 127 Slaves; the Broadcast address is zero.

1.1.2 NMT Services (Network Management)

The NMT “mandatory” services are:

- Enter_Pre-Operational_State CS = 128
- Reset_Node CS = 129
- Reset_Communication CS = 130

Being that the “Minimum Boot-up” is used, also the following NMT services are managed:

- Start_Remote_Mode CS = 1
- Stop_Remote_Mode CS = 2

The COB-ID * of an initialization NMT service is always at 0; CS is the Command Specifier defining the NMT service.

1.1.3 Initialization

The ADV drive supports the Node Guarding and HeartBeat mechanism. The Node Guarding configuration can be performed through the master via the standard Object Dictionary elements (1006h, 100Ch, 100Dh) and the 1016h, 1017h objects for HeartBeat.
The drive checks the master functioning conditions through the Life Guarding. If the check fails, the drive enables the “Buss Loss” alarm. The Life Guarding threshold can be calculated as follows:

Value	Condition
60ms SYNC_PERIOD (*)	Default. No parameterization of the Node Guarding.
LIFE_TIME_FACTOR	Use of the synchronous mode. If not stated by the master, the Life_Time_Factor default value is equal to 3.
NODE_GUARDING_PERIOD (*)	set by the master
LIFE_TIME_FACTOR	If not otherwise stated, the value is equal to 3

1.1.4 Communication objects

This chapter describes the communication objects of the CANopen protocol; they are managed by the interface card.
The managed communication objects are:

1) 1 SDO reception Server.
2) 1 SDO transmission Server.
3) 4 reception PDOs.
4) 4 transmission PDOs.
5) 1 Emergency Object.
6) 1 Node Guarding - Life Guarding.
7) 1 SYNC object.

The following table lists the used communication objects in ascending order downward, and the Message Identifier; the “Resulting COB-ID” is obtained by adding the Node-ID (card address) to the number.

OBJECT	**MESSAGE ID**
NODE GUARDING & HB	1792 700h+NodeId
1st SDO rx	1536 600h+NodeId
1st SDO tx	1408 580h+NodeId
1st PDO tx	384 180h+NodeId
1st PDO rx	512 200h+NodeId
2nd PDO tx	640 280h+NodeId
2nd PDO rx	768 300h+NodeId
3st PDO tx	384 380h+NodeId
3st PDO rx	512 400h+NodeId
4th PDO tx	640 480h+NodeId
4th PDO rx	768 500h+NodeId
EMERGENCY	220 80h+NodeId
SYNC	128 80h
NMT	Network Management

Table 1.4.1: Communication Objects

1.1.5 Object Dictionary Elements

The Object Dictionary is accessible from a master CANopen.
The following table shows the communication objects used and accessibility with master CANopen.

Index (hex)	Name
1000	Device Type
1001	Error Register
1002	Manufacturer status register
1005	COB-ID SYNC Message
1006	Communication cycle period
1008	Manufacterer Device Name
1010	Store parameter
1009	Manufacturer Hardware Version
100A	Manufacturer Software Version
100C	Guard Time
100D	Life Time Factor
1014	COB-ID Emergency
1016	HeartBeat time consumer
1017	HeartBeat time producer
1018	Identity object
1400	1st Receive PDO
1401	2nd Receive PDO
1402	3rd Receive PDO
1403	4th Receive PDO
1600	Receive PDO1 mapping parameter
1601	Receive PDO2 mapping parameter
1602	Receive PDO3 mapping parameter
1603	Receive PDO4 mapping parameter
1A00	Transmit PDO1 mapping parameter
1A01	Transmit PDO2 mapping parameter
1A02	Transmit PDO3 mapping parameter
1A03	Transmit PDO4 mapping parameter
1800	1st Transmit PDO
1801	2nd Transmit PDO
1802	3rd Transmit PDO
1803	4th Transmit PDO

Table1.5.1: Objects used by the CANopen communication profile

The objects shown in bold in the table allow writing of the parameters assigned with the exchange of data in the PDO.
The allocation criterion is variable, and depends on the size (in bytes) of the parameter exchanged.

1.1.6 RX PDO Entries

The structure of the PDO Communication Parameter (index 1400h, 1401h) is:
1) Subindex 0 (Number of supported entries) = 2

2) The structure of Subindex 1 (COB-ID used by the PDO) is:
- Bit 31 (valid/invalid PDO) can be set via SDO.
- Bit 30 (RTR Remote Transmission Request) = 0 because this function is not supported.
- Bit 29 = 0 because the 11-bit ID is used (CAN 2.0A).
- Bits 11-28 are not used.
- Bit 0-10 COB-ID (see table 1.4.1).

3) Cyclic-synchronous Subindex 2 (Transmission Type), or synchronous according to the master performed setting (1 if SYNC has been foreseen, 254...255 if asynchronous). If not stated, the synchronous mode is active.

1.1.7 TX PDO Entries

The structure of the PDO Communication Parameter (index 1800h, 1801h) is:

1) Subindex 0 (Number of supported entries) = 3
2) The structure of Subindex 1 (COB-ID used by the PDO) is:
- Bit 31 (valid/invalid PDO) can be set via SDO.
- Bit 30 (RTR Remote Transmission Request) = 0 because this function is not supported.
- Bit 29 = 0 because the 11-bit ID is used (CAN 2.0A).
- Bits 11-28 are not used.
- Bit 0-10 COB-ID (see table 1.4.1).

3) Cyclic-synchronous Subindex 2 (Transmission Type), or synchronous according to the master performed setting (1 if SYNC has been foreseen, 254...255 if asynchronous). If not stated, the synchronous mode is active.
4) Inhibit timee

1.1.8 SDO Entries

Only the “Expedited” data transfer mode (max. 4 Bytes) is used.

1) Subindex 0 (Number of supported entries) = 3 because the device is a Server of the SDO service.
2) The structure of the Subindex 1 and 2 (COB-ID used by the SDO) is:
- Bit 31 (valid/invalid SDO); it is equal to 1 because just the Default SDOs are used.
- Bit 30 reserved = 0.
- Bit 29 = 0 because the 11-bit ID is used (CAN 2.0A).
- Bits 11-28 are not used.
- Bit 0-10 COB-ID (see table 1.4.1).

The element “node ID of SDO’s client resp. server” is not supported because just the Default SDOs are used.

1.1.9 COB-ID SYNC Entries

The structure of the 32 bits contained in the COB-ID SYNC communication parameter is:

- Bit 31 = 1 because the CANopen interface card is a “consumer” of SYNC messages.
- Bit 30 = 0 because the interface card does not create SYNC messages.
- Bit 29 = 0 because the 11-bit ID is used (CAN 2.0A).
- Bits 11-28 are not used.
- Bit 0-10 COB-ID (see table 1.4.1).

1.1.10 COB-ID Emergency

The structure of the 32 bits contained in the COB-ID Emergency Message communication parameter is:

- Bit 31 = 0 because the CANopen interface card is not a “consumer” of Emergency messages.
- Bit 30 = 0 because the interface card creates Emergency messages.
- Bit 29 = 0 because the 11-bit ID is used (CAN 2.0A).
- Bits 11-28 are not used.
- Bit 0-10 COB-ID (see table 1.4.1).

1.2 CANopen management

The user interface of the CANopen protocol is performed via the drive parameters. The parameters are controlled via hierarchical menus. All the writing parameters referring to the field bus are active only after the drive reset. Here following is a list of drive parameters useful to control the CANopen protocol.
To activate the EXP-CAN-ADV card, set parameter PAR 4000 **Fieldbus type** to CANopen or DS402.
The following parameters are available in the COMMUNICATION->FIELDBUS CONFIG menu:

PAR	Nome Par	Type	Default value	Attr
4004	Fieldbus baudrate	Enum	None	Write
4006	Fieldbus address	2 byte unsigned	0	Write
4010	Fieldbus M->S enable	Enum	On	Write
4012	Fieldbus alarm mode	2 byte unsigned	0	Write
4014	Fieldbus state	Enum	Stop	Read only

- Fieldbus baudrate = Sets the network baud rate. Values available for CANopen: 125k, 250k, 500k, 1M
- Fieldbus address = address of this slave node in the network, accepted values from 1 to 127
- Fieldbus M->S enable = if set to Off data in the RPDOs are not processed by the drive
- Fieldbus alarm mode = if set to 1 the drive generates Opt Bus Fault errors relating to the loss of communication (Bus Loss) even when the drive is not enabled.
- Fieldbus state = state of the communication for this node on the CANopen network: Stop, Pre-Operational, Operational.

1.3 Process Data Channel Control

This function allows to allocate the drive parameters or application variables to the Process Data Channel data.
As for the CANopen protocol, the PDC is performed via the PDO messages ((Process data Object).
The CANopen protocol uses a number of words for the Process Data Channel (abbr. PDC Process Data Channel), which can always be set.
The fieldbus Process Data Channel configuration is the following:

Data 0 Data... Data n

The drive can both read and write the Process Data Channel data.
A datum can be made both of 2 and 4 bytes. The word “data” refers to any quantity of bytes included between 0 and 16, if the byte total number required is not higher than 32.

Example:
It is possible to have:

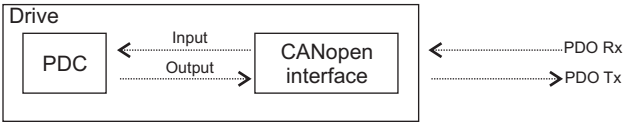
- from 0 to 16 data with 2 bytes
- 1 datum with 4 bytes + from 0 to 14 data with 2 bytes
- 2 data with 4 bytes + from 0 to 12 data with 2 bytes

...

- 8 data with 4 bytes

The data exchanged via the PDC can be of two types:
- drive parameters
- variables of an MDPlc application. The use of the MDPlc variables is described in par. 1.3.1 and 1.3.2.

The master writes the data defined as PDC input and reads the data defined as PDC output.



1.3.1 PDC Input Configuration (FB XXX MS Parameter)

Data exchanged in RPDOs are configured using the parameters in the COMMUNICATION->FIELDBUS M2S menu (refer to the drive manual).
Data mapping in PDOs is performed on the basis of the data format set in **Fieldbus M->Sn sys** according to the following rules:

– *PDOs are filled starting from RPDO1*
– *When the PDO contains 4 words it is full and the next RPDO is filled with a maximum of 4 PDOs.*
– *32-bit data (long or float) cannot be split among PDOs, they must be placed inside the PDO (an alarm is generated).*
– *PDOs containing fewer than 4 words can be created, using **Fieldbus M->Sn ipa** = 0 but assigned (**Fieldbus M->Sn sys** other than Not Assigned, Fill16 or Fill32) after an assigned datum.*
(N.B.: if assigned as Fill16 or Fill32, the datum is included in the PDO anyway)
– *At the first **Fieldbus M->Sn sys** = Not Assigned parameter the PDOs are complete. The size of the last PDO thus depends on the data that have been assigned.*
– ***Example: 2-word RPDO1** and **2-word RPDO2:***
Fieldbus M->S1 ipa = 610 (Ramp ref 1 src)
Fieldbus M->S1 sys = Eu
Fieldbus M->S2 ipa = 4452 (Word decomp src)
Fieldbus M->S2 sys = Count 16
Fieldbus M->S3 ipa = 0
Fieldbus M->S3 sys = Fill 32

Fieldbus M->S4 ipa = 3660 (Compare input 1 src)
Fieldbus M->S4 sys = Count32
Fieldbus M->S5 sys = Not Assigned

1.3.2 PDC Output Configuration (FB XXX SM Parameter)

Data exchanged in RPDOs are configured using the parameters in the COMMUNICATION->FIELDBUS S2M menu (refer to the drive manual).
Data mapping in PDOs is performed on the basis of the data format set in **Fieldbus M->Sn sys** according to the following rules:

– PDOs are filled starting from TPDO1
– When the PDO contains 4 words it is full and the next TPDO is filled with a maximum of 4 PDOs.
– 32-bit data (long or float) cannot be split among PDOs, they must be placed inside the PDO (an alarm is generated).
– PDOs with fewer than 4 words can be created, using **Fieldbus S->Mn ipa** = 0 but assigned (**Fieldbus M->Sn sys** other than Not Assigned, Fill16 or Fill32) after an assigned datum.
– At the first **Fieldbus S->Mn sys** = Not Assigned parameter the PDOs are complete. The size of the last PDO thus depends on the data that have been assigned.

1.3.3 Use of the PDC in MDPlc Applications

It is possible to configure both the PDC input and output data in order to allow the data direct access via the MDPlc application code.
For read data simply set **Fieldbus M->Sn sys** to MDPLC16 or MDPLC32, leaving **Fieldbus M->Sn ipa** = 0.
The MDPLC application can now read the incoming datum directly from the **Fieldbus M->Sn mon** parameter.
Write data are configured by setting **Fieldbus S->Mn ipa** = [4184 +(n-1)*10] **(Dig Fieldbus S->Mn**, $1 \leq n \leq 16$).
Fieldbus S->Mn sys is automatically set to MDPLC. The application writes the datum in the **Dig Fieldbus S->Mn** parameter to send it to the bus.

1.4 SDO management

The SDO service is always available.
The drive parameters can be accessed via the “MSPA” Manufacturer Specific Profile Area (2000hex< index <5FFFhex).
The index to be shown in the SDO command to access a drive parameter is obtained via the following rules:
SDO index = PAR + 2000h
SDO subindex = 1

The Data field must contain the value of the drive parameter.

Example:
Writing the value 1000 in the PAR 600 **Dig Ramp ref 1** parameter (258hex).

The following information is required:
1) The SDO index resulting from the formula is:
   2000hex + 258hex = 2258h
2) The value to be written is 1000, corresponding to 03E8 hex.

Index		Subindex				
22h	**58h**	**01h**	**E8h**	**03h**	**00h**	**00h**
Drive parameter index		Subindex	Drive parameter value to be assigned to SDO			

In case an error occurs during the parameter reading or setting, the CANopen interface sends an Abort domain transfer message; the value of Application-error-codes has the following meanings:

Error class	Error code	Additional code (hex)	Meaning
6	0	0	Parameter doesn't exist
8	0	22	Acces failed because of present device state
6	1	2	Read/Write only error
8	0	0	Generic error
6	9	32	Minimum value
6	9	31	Maximum value
5	4	0	SDO time_out
5	4	1	Invalid command
3	9	30	Invalid value

1.5 Alarms

Fieldbus alarms

The bus failure is signaled via the "Opt Bus Fault" alarm. As for CANopen, the possible failure causes are:

- "Bus-off" condition of the CAN line;
- th drive has not been enabled in the "Operational" mode;
- exceeding of monitoring limit (heartbeat, node guard, communication cycle period).
- drive configuration error

This alarm becomes active only when the drive is enabled.
If ON, the PAR 4014 **Fieldbus alarm mode** parameter enables the generation of the "Field bus failure" alarm also when the drive is disabled.

Code	Cfg	Description	Actions
0		Bus Loss	Check line for noise, terminations , problems with cabling
FF01	*	Fieldbus type does not match expansion card	Verify if EXP-CAN-ADV card is properly installed
FF02	*	Wrong baudrate selected	Check "Fieldbus baudrate" is one of 125k, 250k, 500k, 1M
FF03	*	Invalid address for node	Check "Fieldbus address"
FF04	*	Error initializing CAN interface	Internal error, contact manufacturer
FF14..FF23	*	Wrong object selected for mapping in channel M2S n	Check "Fieldbus M->Sn Dest

Code	Cfg	Description	Actions
FF24..FF33	*	More than 1 Src pointing to M2S Channel n	Check for multiple destinations on “Fieldbus M->Sn Dest”
FF34..FF43	*	M2S Channel n , data size is wrong (16 bits on 32 bits or 32 bits on 16 bits parameter)	Check “Fieldbus M->Sn sys”
FF44..FF53	*	Invalid parameter in channel S2M n	Check “Fieldbus S->Mn src”
FF54..FF63	*	S2M Channel n , data size is wrong (16 bits on 32 bits or 32 bits on 16 bits parameter)	Check “Fieldbus S->Mn sys”
FF64..FF73	*	Wrong object selected for mapping in channel S2M n	Check “Fieldbus S->Mn src”
FF74..FF83	*	M2S Channel n : too many words in PDC	“Fieldbus M-Sn dest” & “Fieldubs M->Sn sys” address more than 16 words in PDC
FF84..FF93	*	S2M Channel n : too many words in PDC	“Fieldbus S->Mn src” & “Fieldubs S->Mn sys” address more than 16 words in PDC
FFB4..FFC3	*	Internal database error on channel n	Internal error, contact manufacturer
8110		CAN msg overflow	Too many packets for selected baudrate
8130		LifeGuard/HeartBeat error	Software timeout from master
FFC5		Wrong NMT message length	Check NMT packets
FFC6		Invalid NMT command	Check NMT packets
FFC7		CAN bus off	Check line state for problems
8100		CAN bus off	Check line state for hardware problems

Drive alarm messages

Drive alarms are managed by means of an Emergency message containing the error code relating to the alarm that is generated, according to the table below:

Selection	Code
No alarm	0x0000
Overvoltage	0x3210
Undervoltage	0x3220
Ground fault	0x2110
Overcurrent	0x2310
Desaturation	0x2130
MultiUndervolt	0xFF06
MultiOvercurr	0xFF07
MultiDesat	0xFF08
Heatsink OT	0x4210
HeatsinkS OTUT	0x4310
Intakeair OT	0x4130
Motor OT	0xFF0C
Drive overload	0x8311

Selection	Code
Motor overload	0x7121
Bres overload	0x7112
Phaseloss	0xFF10
Opt Bus fault	0xFF11
Opt 1 IO fault	0xFF12
Opt Enc fault	0x3130
External fault	0x9000
Speed fbk loss	0x7310
Overspeed	0x8400
Plc1 fault	23
Plc2 fault	24
Plc3 fault	25
Plc4 fault	26
Plc5 fault	27
Plc6 fault	28
Plc7 fault	29
Plc8 fault	30
Emg stop alarm	31
Watchdog	32
Trap error	33
System error	34
User error	35
Power down	36
Speed ref loss	37
Not Used1	38
Opt 2 IO fault	39
Not Used2	40
Not Used3	41
Not Used4	42
Not Used5	43
Not Used6	44
Param error	45

1.6 Configuration example

This chapter provides an example of how to configure the parameters of ADV200 drives so that they can be read and written by a CANopen master via the processing channels (PDO). See the chapter 1.4 for the configuration channels (SDO). The paragraph 1.6.1 provides the information required on a CANopen master controlling a machine. The paragraph 1.6.2 contains basic information for programming the ADV200 drive starting from the factory settings.

1.6.1 CANopen Master

This section contains an example of data exchange seen from the master side. This is the data normally contained in the machine specifications in the case of applications controlled by a CANopen master.

1.6.1.1 Description of Master -> Slave PDO Communication

There are two parameters to be written via the processing channels. The first is a control word, in which the single bits contain certain commands (e.g. enable, start, etc.). The second processing channel contains the ramp reference 1 (RampRef1) in rpm.

CANopen PDO: Master -> Drive (max 16 word)

Position	Description	Format	Unit of Measure
Word1 M -> S	Control word	16 bit Word	...
Word2 M -> S	Ramp Ref 1	Int 16 bit	rpm
Word3 M -> S			
Word4 M -> S			
Word5 M -> S			
Word6 M -> S			
Word7 M -> S			
...			
...			
Word16 M > S			

CONTROL WORD

Bit	Description	Remarks
0	EnableCmd	Enable command from CANopen master
1	StartCmd	Start command from CANopen master
2	Free	
3	Free	
4	Free	
5	Free	
6	Free	
7	Free	
8	Digital Out3	Digital output 3 command from CANopen master
9	Digital Out4	Digital output 4 command from CANopen master
10	Free	
11	Free	
12	Free	
13	Free	
14	Free	
15	Free	

1.6.1.2 Description of Slave -> Master PDO Communication

The CAN master reads three parameters from the drive. The first contains a status word in which the single bits carry information about the status of the drive (e.g. DriveOk). The second parameter is the actual speed in rpm. The third parameter contains the value of analog input 2.

CANopen PDO Slave > Master (max 16 Word)

Position	Description	Format	Unit of Measure
Word1 S -> M	Status Word	16 bit Word	BitWide
Word2 S -> M	Actual Speed	Int 16 bit	rpm
Word3 S -> M	Analog Input 2	Int 16 bit	
Word4 S -> M			
Word5 S -> M			
Word6 S -> M			
Word7 S -> M			
...			
...			
Word16 S -> M			

STATUS WORD

Bit	Description	Remarks
0	EnableState	Drive enabled
1	Drive Ok	Drive Ok
2	Speed is zero	Zero speed threshold
3	Free	
4	Free	
5	Free	
6	Free	
7	Free	
8	Digital Input 4	ADV200 digital input 4 status
9	Digital Input 5	ADV200 digital input 5 status
10	Free	
11	Free	
12	Free	
13	Free	
14	Free	
15	Free	

1.6.2 ADV200 Configuration

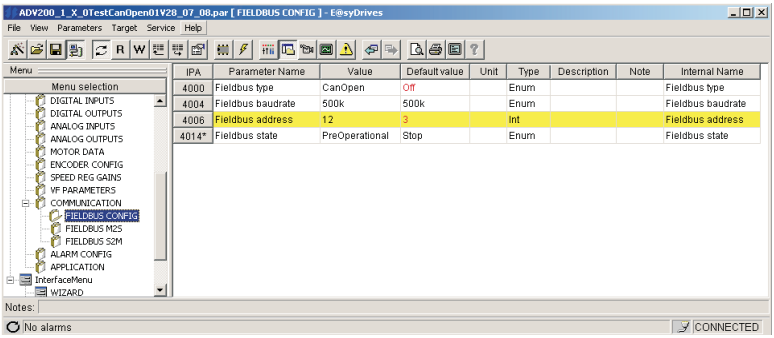
The example given in this section is based on the assumption that the parameters of the ADV200 drive are the factory settings (**Default parameter** command).

1.6.2.1 FIELDBUS CONFIG

The example assumes that the drive is node 12 and the CANopen communication baudrate is 500k.

Note: The drive must be reset to make all fieldbus settings and configurations effective.

Configure the fieldbus menu parameters as shown below:



The pre-operational status of the CANopen expansion card LEDs is shown in the relative column:

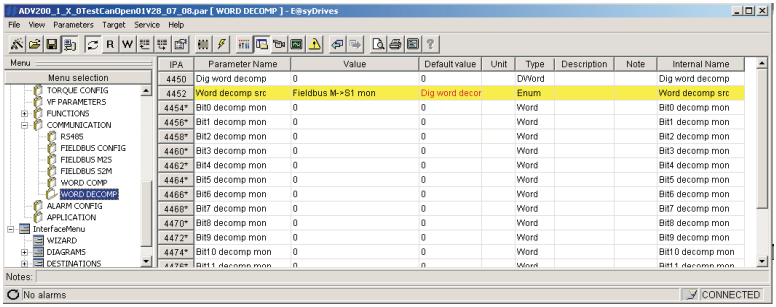
Led	Status = Pre-operational		Status = Operational
AL (Red)	ON	alarm condition	OFF
OP (Green)	OFF	Pre-Operational condition	ON
PWR (Green)	ON	expansion card powered and active	ON
+5VC (Green)	ON	CAN opto-isolated power	ON

Processing channel communication is not active in these conditions.
When drive configuration is complete (see following sections) use the NMT “start node” command to activate communication by the master.

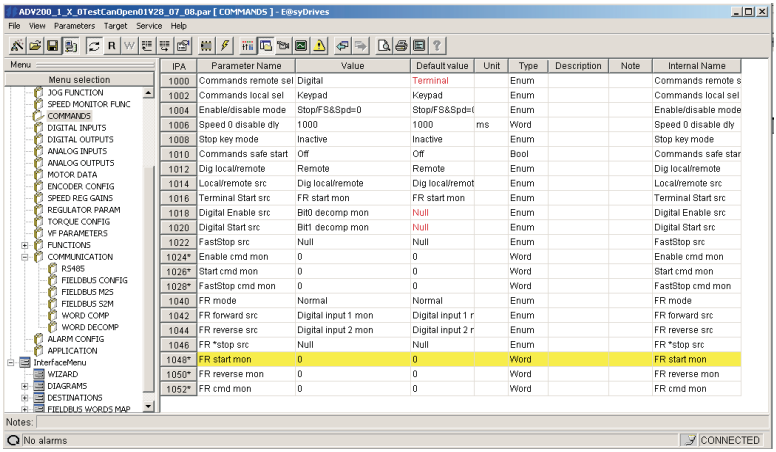
Upon receiving this command the FieldBus State parameter moves to Operational and the green OP LED is set to “ON”. Only at this point are the processing channels active.

1.6.2.2 MASTER -> SLAVE channel configuration

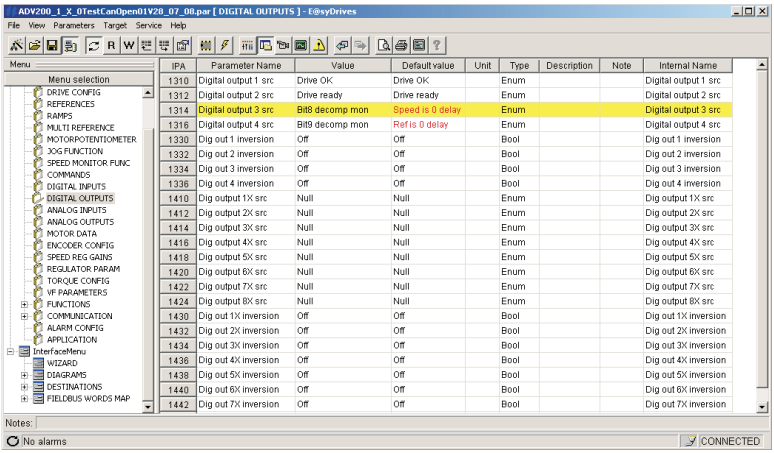
Wdecomp is used to **configure the control word**. The Wdecomp configuration on the first M -> S word (“Export” mode) is shown below:



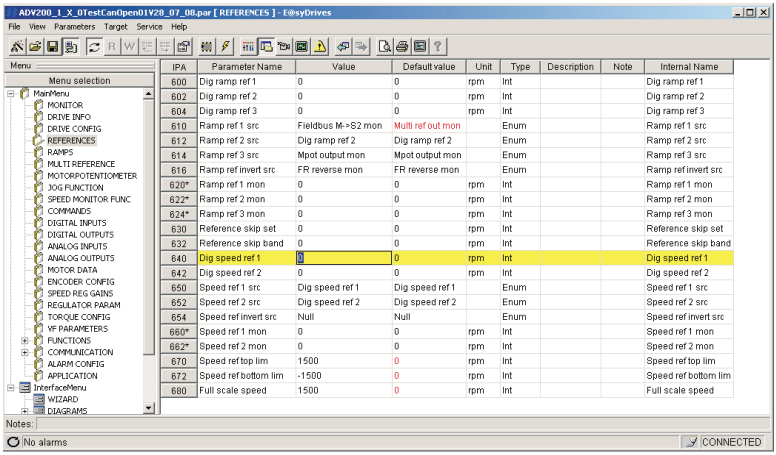
Now simply connect the single Wdecomp bits. For Commands the drive must be set to “**Remote**” and “**Digital**” mode, as explained in the ADV200 manual. Configure the first two bits in the commands menu as shown below:



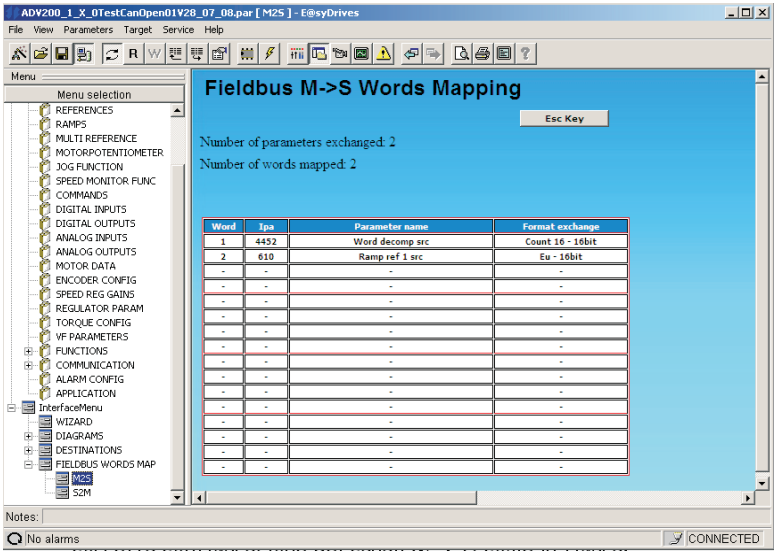
Configure bits 8 and 9 of the “Command word” as shown below (Digital Outputs menu):



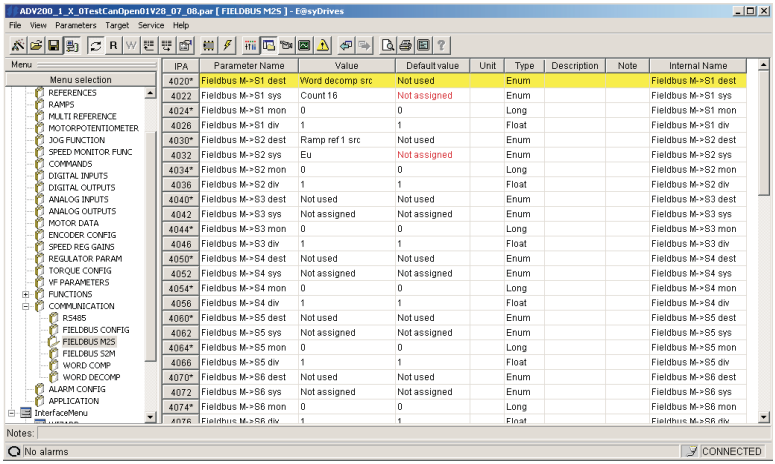
The second word is configured in the “References” menu:



After sending a save command and re-starting the drive, check that the M -> S channels have been configured correctly as shown (Html page):

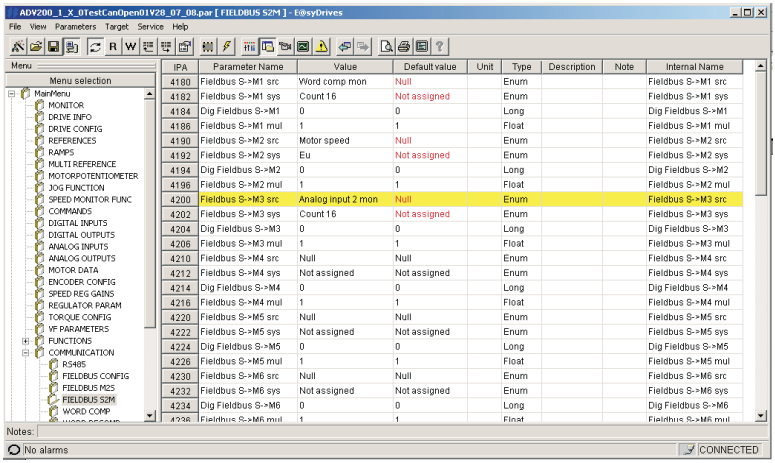


The same information is also available in the Fieldbus M2S menu:

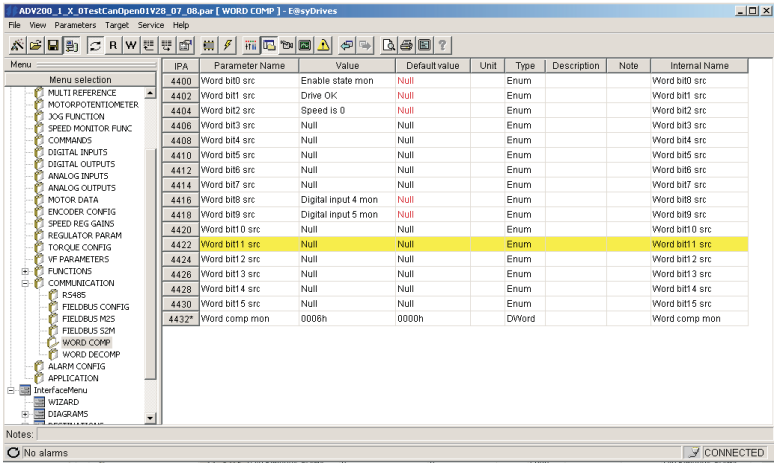


1.6.2.3 SLAVE -> MASTER channel configuration

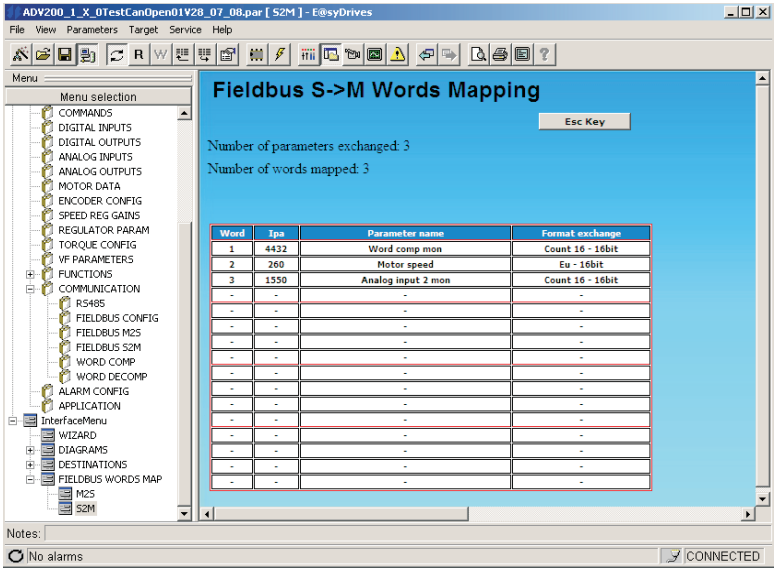
These channels are configured in the Fieldbus S2M menu. Use Wcomp to configure the first channel.
S2M configuration is shown below:



Wcomp configuration is shown below:



Save and then re-start the drive to check the correct configuration of the Slave -> Master channels in the same way:



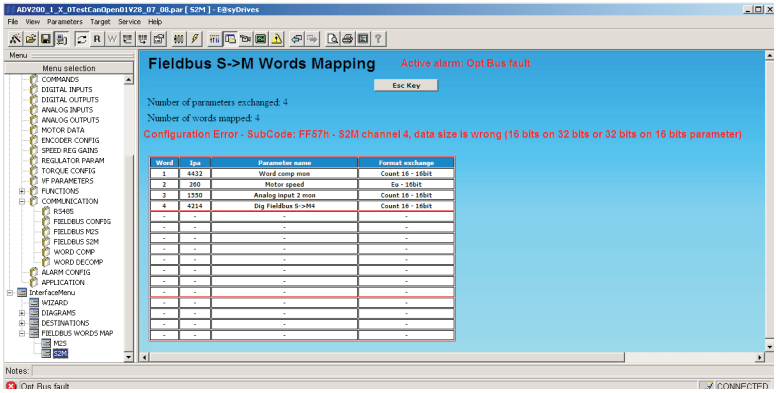
1.6.2.4 Communication check

Some notes/suggestions for checking communication.

- PDO communication is only active in “Operational”. Check the status using Easydrive or the expansion card LED.
- For Master -> Slave communication in the FIELDBUS M2S menu you can check the value received by the communication channel (e.g. for the first channel it is the Fieldbus M->S1 Mon parameter).
- For EU (engineering unit) communication, remember that the value read on the Mon parameters of FIELDBUS MS2 is in internal units (see conversion tables on chapter 5.0 SYSTEM INTERNAL VARIABLES, “ADV200, Write the applications with the MDPlc” manual available on www.weg.net).

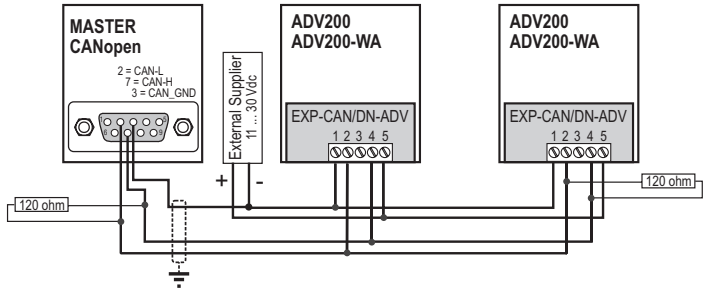
1.6.2.5 Configuration Errors

In the event of a channel configuration error an “Option bus fault” alarm condition occurs when the drive is switched on
with an error code indicating the channel that generated the alarm. The expansion card manual contains a table listing all the error codes.
With WEG_eXpress software configuration simply log-on to the relative HTML page as shown below:



1.7 Connecting the master control panel to ADV200 nodes

The following example shows a connection between the Control panel CANopen Master and ADV200 slave nodes.



This is a 3-wire connection: CAN line and unipotential cable on terminals 1 (V- / CAN_GND) of the EXP-CAN/DN-ADV cards, 0 V power supply and pin 3 (CAN_ GND) CAN connector on Vedo terminal.
The shielding must be continuous along the entire CAN line.
The shielding is grounded at a single point, in proximity of the CAN line power supply. This supply can also be used to power the Vedo terminal. It is NOT advisable to use this power supply for other purposes, especially auxiliary circuits with relays.
The CAN line shielding may also be grounded at two or more points, for example if the CAN line nodes are distributed in separate electrical panels, provided the ground connections are made correctly.

2.0 Operation according to the DS402 profile

If the **Fieldbus type** parameter is set to DS402 the drive works with the standard profile for Drives & Motion Control Ver 2.0 and contains Device Identity 192H (402) in object 1000h.
The ADV200 drive supports Velocity Mode.
In the default configuration the drive is automatically set to use PDOs No. 6 (DS402 section 7.2.1 & 7.2.2), mapped onto RPDO1 and TPDO1 with COB-ID 200h & 180h +NodeId

PDO	Object Number	Object Name	Description
6	6040h	Controlword	controls the state machine and the nominal speed (vl)
	6042h	Target velocity (vl)	
6	6041h	Statusword	shows status and the current speed (vl)
	6044h	vl control effort	

The remaining PDOs can be set by the user.

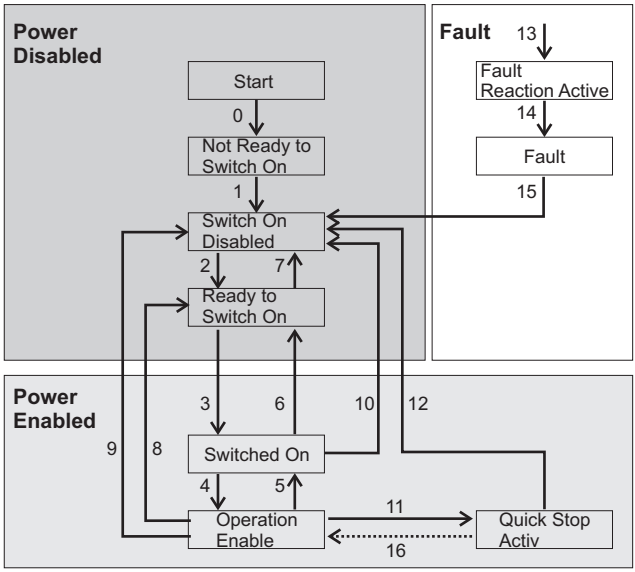
The following DS402 objects are implemented:

Object no.		Description	Type	Access	Mandatory
6040h	VAR	Controlword	UNSIGNED16	rw	M
6041	VAR	Statusword	UNSIGNED16	ro	M
6042h	VAR	vl target velocity (1)	INTEGER16	rw	M
6043h	VAR	vl velocity demand (1)	INTEGER16	ro	M
6044h	VAR	vl control effort(1)	INTEGER16	ro	M
6046h	ARRAY	vl velocity min max amount	UNSIGNED32	rw	M
6048h	RECORD	vl velocity acceleration	vl velocity acceleration	rw	M
6049	RECORD	vl velocity deceleration	vl velocity deceleration	rw	M
6060h	VAR	Modes of operation (2)	INTEGER8	rw	M
6061h	VAR	Modes of operation display	INTEGER8	ro	M

(1) The unit of measure for objects 6042h, 6043h, 6044h is expressed in rpm

(2) Object 6060h is only available in that it is mandatory. As the drive only supports Velocity Mode, the value of this object is not modifiable.

The device operates as a DS402 state machine (refer to CiA DSP 402 V 2.0, section 10.1.1):


Power Disabled
Start
0
Not Ready to Switch On
1
Switch On Disabled
2
7
Ready to Switch On
Fault
13
Fault Reaction Active
14
Fault
15
Power Enabled
3
6
10
12
9
8
Switched On
4
5
11
Operation Enable
Quick Stop Activ
16

3.0 DeviceNet Interface

This chapter describes the connecting of ADV200 drives to DeviceNet networks. It is intended for design engineeres and technicians responsible for the maintenance, commissioning and operation of DeviceNet systems.
A basic knowledge of DeviceNet is assumed and may be found in the following manuals:

- DeviceNet Specifications. Volume 1 - DeviceNet Communication Model and Protocol (Issued by ODVA).
- DeviceNet Specifications. Volume 2 - DeviceNet Device Profiles and Object Library (Issued by ODVA)

3.1 General description of DeviceNet

DeviceNet is a profile of communication for industrial systems based on CAN.
As protocol CAN (ISO 11898) is used CAN2.0A with the 11 bit identifier.
The ADV200 driver is developed as “Slave UCMM Capable Device” for operating only in “Predefined Master/Slave Connection Set”.
The data transfer is carried out cyclically; the Master unit reads the data supplied by the Slaves and writes the Slave reference data; the Baud Rate supported by the SBI card are: 125 kbit, 250 kbit, 500 kbit .

The physical support is given by the RS485 serial line; a maximum of 64 Slaves can be connected to the Bus.

3.2 DeviceNet function

In this chapter are described the functions of DeviceNet managed by the driver.
The main characteristics of the card are:

1. The drive operates only as Slave in “Predifined Master/Slave Connection Set”.
2. Within the “Predefined Master/Slave Connection Set” the driver is a “UCMM Capable Device”.
3. The “Explicit Messaging” is managed.
4. The “Polling” for the fast cyclical data exchange Master/Slave is managed.
5. The detection mechanism of the “Duplicate MAC ID” is implemented.

Regarding the “Explicit Messaging” the fragmentation of the data frame, with a total of max. 32 byte, is managed.

Connection sizes

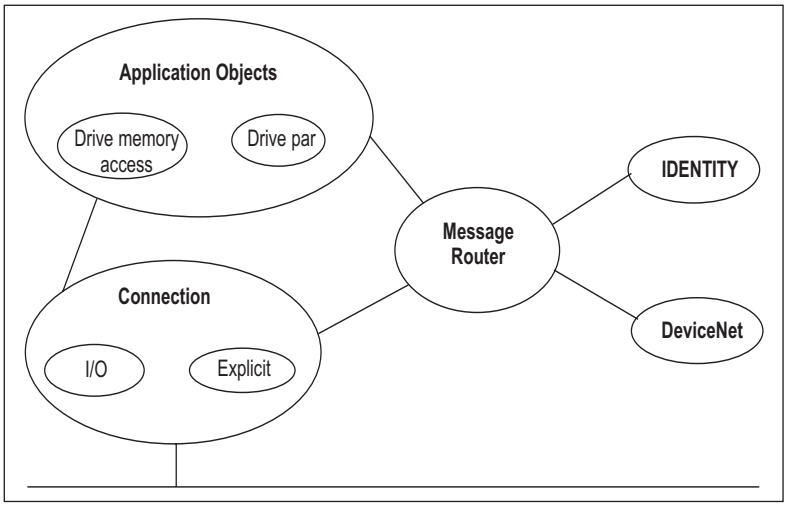
CONNECTION INSTANCE	PRODUCED	CONSUMED
Polled I/O	Depending on frame setting	
Explicit messaging	32	32

3.3 Object description

Hereafter you find the description of the objects managed by the ADV200 drive.

3.3.1 Object Model

The following figure shows the ADV200 “Object Model”.



The following table shows:
1. The object classes of EXP-CAN-ADV card.
2. If the class is mandatory.
3. The number of instances included in every class.

See “DeviceNet Specifications” for the Standard classes.

Object	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
DeviceNet	Required	1
Connection	Required	1 I/O, 3 Explicit
Parameter	Optional	many
Drive Parameter Access	Optional	many
Drive memory Access	Optional	many

3.3.2 How Objects Affect Behavior

The “Affect Behaviour” of the objects is reported in the following table:

Object	Effect on Behavior
Identity	Supports Reset Service
Message Router	No effect
DeviceNet	Port attributes configuration
Connection	Contains the number of logical ports
Parameter	Drive parameters read/write
Drive Parameter Access	Drive parameters read/write
Drive Memory Access	Drive parameters read/write

3.3.3 Defining Object Interface

The object interface of the ADV200 drive is the following:

Object	Interface
Identity	Message router
Message Router	Explicit Messaging Connection Instance
DeviceNet	Message router
Connection	Message router
Parameter	Message router
Drive Parameter Access	Message router
Drive memory Access	Message router

3.4 Data transfert via Explicit Messaging

The data transfer via Explicit Messaging is made through two new objects: one for accessing the Drive parameters, the other to direct access the drive memory.

3.4.1 Parameter Access

For read/write of Drive parameters, the Drive Parameter Access object is defined with the following properties:

- Class ID: Fh.
- Class Attribute: Revision
- Instance Attribute: This instance does not have attributes.

3.4.1.1 Class Code

Class code: F hex

3.4.1.2 Class attributes

Number	Need in implementation	Access Rule	Name	DeviceNet Data Type	Description of Attribute	Semantics of values
1	Optional	Get	Revision	UINT	Revision of this object	

3.4.1.3 Instance Attributes

Number	Need in implementation	Access Rule	Name	DeviceNet Data Type	Description of Attribute	Semantics of values
This instance provide attributes and correspond to the PAR						

3.4.1.4 Common Services

This object has no common services.

3.4.1.5 Object Specific Services

Service Code	Need in implementation		Service Name	Description of Service
	Class	Instance		
0hex	n/a	Required	Get_Attribute_Single	Read drive parameter value
10hex	n/a	Required	Set_Attribute_Single	Writes drive parameter value

3.4.1.6 Behavior

This object is the interface between the DeviceNet and all drive parameters.
The Drive parameter is accessed via the parameter index itself.
*For example, reading a parameter (PAR 600 **Dig Ramp ref 1**):*

- Run a Get_Attribute_Single from class Fh
- instance =600 (258 hex)
- class 1 attribute
- the drive responds with 4 bytes (Dword format).

*For example, writing a parameter (PAR 600 **Dig Ramp ref 1**):*

- Run a Set_Attribute_Single from class Fh
- instance = 600 (258 hex)
- class 1 attribute
- to set value 1000, select “Word 2 byte” (parameter format is INT, 16 bit)
- the drive does not respond if there is an error (timeout).

byte	**VALUE**	**XX**	Low byte - Low word drive parameter drive
			High byte - Low word drive parameter drive
			Low byte - High word drive parameter drive
			High byte - High word drive parameter drive

The number of bytes in the “Value” field depends on the length of drive parameter.

Example:
if the type of drive parameter is “Integer” the length of VALUE is 2 bytes.

3.4.2 Drive Parameter Access

For read/write of Drive parameters, the Drive Parameter Access object is defined with the following properties:
- Class ID: 66h.
- Class Attribute: Revision
- Instance Attribute: This instance does not have attributes.

3.4.2.1 Class Code

Class code: 66 hex

3.4.2.2 Class attributes

Number	Need in implementation	Access Rule	Name	DeviceNet Data Type	Description of Attribute	Semantics of values
1	Optional	Get	Revision	UINT	Revision of this object	

3.4.2.3 Instance Attributes

Number	Need in implementation	Access Rule	Name	DeviceNet Data Type	Description of Attribute	Semantics of values
This instance does not provide attributes						

3.4.2.4 Common Services

This object has no common services.

3.4.2.5 Object Specific Services

Service Code	Need in implementation		Service Name	Description of Service
	Class	Instance		
32hex	n/a	Required	Get_Drive_Value	Get parameter from Drive
33hex	n/a	Required	Set_Drive_Value	Set parameter into Drive
34hex	n/a	Required	Get_Typed_Drive_ Value	Read drive parameter value indicating the data type
35hex	n/a	Required	Set_Typed_Drive_ Value	Writes drive parameter value indicating the data type

3.4.2.6 Behavior

This object is the interface between the DeviceNet network and all Drive parameters. The access to the Drive parameter is carried out by the parameter index; if the parameter does not exist or may not be accessed for any reason (for example: try to write a read only parameter) an error code will be returned.
Drive parameters in text format cannot be accessed.
In the following are repeted patterns of how the data frame of data has to be composed for reading/writing Drive parameters.

A) Write Drive Parameter

In this example the writing of a Drive parameter is shown; the cases of positive or wrong writing are distinguished.

A-1) Write Drive Parameter Request

The data frame for writing a drive parameter is composed as follows:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	33hex	Set Drive Parameter - Object Specific Service
See Note 1)	Class ID	66hex	Drive Parameter Access Class Object
	Instance ID	XXXX	Drive Parameter Index in format Low byte-High byte
Byte 2)	VALUE	XX	Low byte-Low word drive parameter value
			High byte-Low word drive parameter value
			Low byte-High word drive parameter value
			High byte-High word drive parameter value

1) Byte or Word depending on the type of allocation executed by the Master.
2) The number of bytes of the “Value”-field depends on the length of the Drive parameter; i.e.: if the Drive parameter type is “Integer” the length of VALUE is 2 bytes.

A-2) Write drive parameter - Reply OK

If the Drive parameter is written correctly, the response is:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	33hex OR 80hex	Set Drive Parameter Reply code- Object Specific Service.
Word	Result	0000	Result field equal to zero means writing correctly executed.

A-3) Write drive parameter - Reply Error

If the writing of the drive parameter has been rejected, the response is the following:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	33hex OR 80hex	Set Drive Parameter Reply code- Object Specific Service.
Word	Result	XXXX 1	Drive specific error code.

1) For error codes see table 3.4.1.

B) Read Drive Parameter

In this example is shown the reading of a Drive parameter; the cases of positive or wrong reading are distinguished.

B-1) Read Drive Parameter Request

The data frame for the Drive parameter reading is composed as follows:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	32hex	Get Drive Parameter - Object Specific Service.
See Note 1)	Class ID	66hex	Drive Parameter Access Class Object.
See Note 1)	Instance ID	XXXX	Drive Parameter Index in format Lowbyte-High byte.

1) Byte or Word depending on the type of allocation executed by the Master.

B-2) Read drive parameter - Reply OK

If the Drive parameter is read correctly, the response is:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	32hex	Get Drive Parameter Reply code- Object Specific Service.
Word	Result	0	Result field equal to zero means reading correctly executed.
Byte 1)	VALUE	XX	Low byte-Low word drive parameter value.
			High byte-Low word drive parameter value.
			Low byte-High word drive parameter value.
			High byte-High word drive parameter value.

1) The number of bytes of the Value-field depends on the length of the Drive parameter; i.e. if the Drive parameter type is “Integer” the length of VALUE is 2 bytes.

B-3) Read drive parameter - Reply Error

If Drive parameter reading is rejected, the response is the following:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	32hex	Get Drive Parameter Reply code- Object Specific Service.
Word	Result	XXXX 1	Drive specific error code.

1) For error codes see table 3.4.1.

C) Write Typed Drive Parameter

In this example the writing of a Drive parameter is shown; the cases of positive or wrong writing are distinguished.

In this case, it is shown the parameter IPA number, the value and the data type used in the data transmission.

The optional data type conversion is automatically executed by the firmware.

C-1) Write Drive Parameter Request

The data frame for writing a drive parameter is composed as follows:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	35hex	Set Drive Parameter - Object Specific Service
See Note 1)	Class ID	66hex	Drive Parameter Access Class Object
	Instance ID	XXXX	Drive Parameter Index in format Low byte-High byte
Byte 2)	DATA TYPE	XX	Value data type
Byte 3)	VALUE	XX	Low byte-Low word drive parameter value
			High byte-Low word drive parameter value
			Low byte-High word drive parameter value
			High byte-High word drive parameter value

1) Byte or Word depending on the type of allocation executed by the Master.
2) The coding of the possible data type is listed in table 3.4.2.
3) The number of bytes of the “Value”-field depends on the length of the Drive parameter; i.e.: if the Drive parameter type is “Integer” the length of VALUE is 2 bytes.

C-2) Write drive parameter - Reply OK

If the Drive parameter is written correctly, the response is:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	33hex	Set Drive Parameter Reply code- Object Specific Service.
Word	Result	0000	Result field equal to zero means writing correctly executed.

C-3) Write drive parameter - Reply Error

If the writing of the drive parameter has been rejected, the response is the following:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	33hex	Set Drive Parameter Reply code- Object Specific Service.
Word	Result	XXXX 1)	Drive specific error code.

1) For error codes see table 3.4.1.

D) Read Drive Parameter

In this example is shown the reading of a Drive parameter; the cases of positive or wrong reading are distinguished.

In this case, it is shown the parameter IPA number, the value and the data type used in the data transmission.

The optional data type conversion is automatically executed by the firmware.

D-1) Read Drive Parameter Request

The data frame for the Drive parameter reading is composed as follows:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	36hex	Get Drive Parameter - Object Specific Service.
See Note 1)	Class ID	66hex	Drive Parameter Access Class Object.
	Instance ID	XXXX	Drive Parameter Index in format Lowbyte-High byte.
Byte 2)	DATA TYPE	XX	Value data type

1) Byte or Word depending on the type of allocation executed by the Master.
2) The coding of the possible data type is listed in table 3.4.2.

D-2) Read drive parameter - Reply OK

If the Drive parameter is read correctly, the response is:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	32hex	Get Drive Parameter Reply code- Object Specific Service.
Word	Result	0	Result field equal to zero means reading correctly executed.
Byte 1)	VALUE	XX	Low byte-Low word drive parameter value.
			High byte-Low word drive parameter value.
			Low byte-High word drive parameter value.
			High byte-High word drive parameter value.

1) The number of bytes of the Value-field depends on the length of the Drive parameter; i.e. if the Drive parameter type is “Integer” the length of VALUE is 2 bytes.

D-3) Read drive parameter - Reply Error

If Drive parameter reading is rejected, the response is the following:

DATA TYPE	FIELD	VALUE	MEANING
Byte	Service Code	32hex	Get Drive Parameter Reply code- Object Specific Service.
Word	Result	XXXX 1	Drive specific error code.

1) For error codes see table 3.4.1.

Table 3.4.1: Error codes for the parameter access

Code	Descrption
1	Incorrect parameter number
9	Maximum value exceeded
10	Minimum value exceeded
11	Value not allowed for the parameter
12,13	Read-only parameter
16,31	Parameter cannot be written with drive enabled
20	Parameter loading error
21	Error saving parameter
23	Parameter timeout
Other	Generic error, request technical assistance

Table 3.4.2: Parameter format

FORMAT	VALUE	MEANING
DB_T_VOID	0	Ritorno al valore nel formato originale
DB_T_INT	3	16 bit con segno
DB_T_WORD	6	16 bit senza segno
DB_T_LONG	4	32 bit con segno
DB_T_DWORD	7	32 bit senza segno
DB_T_FLOAT	8	Formato Float IEEE 754

3.5 Polling function

This type of DeviceNet-function is used for a fast cyclic exchange of Drive-parameters between Master and ADV200 drive.
The characteristics of the Polling-function are:
1. The data frame length is configurable through specific drive parameter (see

COMMUNICATION menu) and can vary from 1 to 16 word for both directions (Slave->Master and Master->Slave).

2. The card, as it is a Slave, during the Polling consumes Output data and produces Input data as response.

The configuration of the Drive parameters transferred via Polling is set by using configuration parameter allocated in the drive (see COMMUNICATIONS menu).

3.6 Devicenet Interface configuration

The DeviceNet interface configuration is performed via the drive parameters. The parameters are controlled via hierarchical menus. All the writing parameters referring to the DeviceNet interface are active only after the drive reset. Here following is a list of drive parameters useful to control the DeviceNet interface

To activate the EXP-CAN-ADV card, set parameter PAR 4000 **Fieldbus type** to "DeviceNet".

The following parameters are available in the COMMUNICATION->FIELDBUS CONFIG menu:

PAR	Parameter Description	Type	Default value	Attr
4004	Fieldbus baudrate	Enum	None	Write
4006	Fieldbus address	2 byte unsigned	0	Write
4010	Fieldbus M->S enable	Enum	On	Write
4012	Fieldbus alarm mode	2 byte unsigned	0	Write
4014	Fieldbus state	Enum	Stop	Read only

- Fieldbus baudrate = Sets the network baud rate. Values available for DeviceNet: 125k, 250k, 500k
- Fieldbus address = address of this slave node in the network, accepted values from 1 to 63
- Fieldbus M->S enable = if set to Off, master to slave Polling data are not managed
- Fieldbus alarm mode = if set to 1 the drive generates Opt Bus Fault errors relating to the loss of communication (Bus Loss) even when the drive is not enabled.
- Fieldbus state = state of the communication for this node on the DeviceNet network
  - Stop: No communication with the master
  - Pre-Operational: Recognition by master in progress
  - Operational: I/O Polling active

3.7 Alarms

3.7.1 DeviceNet Alarms

The **Opt Bus Fault** alarm indicates a bus malfunction. In case of DeviceNet, possible reasons for faults are:

- CAN line in Bus-off state;
- drive enabled in a state other than I/O Polling
- connection timeout limit exceeded.

This alarm is only activated when the drive is enabled.
If parameter PAR 4014 **Fieldbus alarm mode** is set to ON, the **Opt Bus Fault** alarm can be generated even with the drive disabled.

Code	Cfg	Description	Action
0		Bus Loss	Check line for noise, terminations, problems with cabling
FF01	*	Fieldbus type does not match expansion card	Verify if EXP-CAN-ADV card is properly installed
FF04	*	Error initializing CAN interface	Internal error, contact manufacturer
FF24..FF33	*	More than 1 Src pointing to M2S Channel n	Check for multiple destinations on "Fieldbus M->Sn Dest"
FF34..FF43	*	M2S Channel n , data size is wrong (16 bits on 32 bits or 32 bits on 16 bits parameter)	Check "Fieldbus M->Sn sys"
FF44..FF53	*	Invalid parameter in channel S2M n	Check "Fieldbus S->Mn src"
FF54..FF63	*	S2M Channel n, data size is wrong (16 bits on 32 bits or 32 bits on 16 bits parameter)	Check "Fieldbus S->Mn sys"
FF74..FF83	*	M2S Channel n : too many words in PDC	"Fieldbus M-Sn dest" & "Fieldubs M->Sn sys" address more than 16 words in PDC
FF84..FF93	*	S2M Channel n : too many words in PDC	"Fieldbus S->Mn src" & "Fieldubs S->Mn sys" address more than 16 words in PDC
FFB4..FFC3	*	Internal database error on channel n	Internal error, contact manufacturer
8110		CAN msg overflow	Too many packets for selected baudrate
FFC5		Wrong baudrate	Wrong baudrate
FFC6		Wrong MacID	Check "Fieldbus address" is 1 to 63
FFC7		CAN bus off	Check line state for problems
FFC8		System error on connection	Check master for proper connection
FFC9		Duplicate MacID Check failed	Check "Fieldbus address" is unique in the network

3.7.2 Drive alarm handling

The "drive alarm status" is not foreseen.
The "drive alarm status" is not therefore given any special treatment.
The ADV200 firmware, provides a series of parameters capable of detecting the drive status. Refer to drive manual for more information.

3.8 Process Data Channel Control

This function allows to allocate the drive parameters or application variables to the Process Data Channel data.
The ADV200 DeviceNet interface uses a number of words for the Process Data Channel (abbr. PDC Process Data Channel), which can always be set.
The Process Data Channel configuration for the ADV200 interface is the following:

DATA 0 DATA... DATAn

The Slave can both read and write the Process Data Channel data.
The DeviceNet data read by the Slave are defined as input data; the data written in DeviceNet by the Slave are defined as output data.

A datum can be made both of 2 and 4 bytes. The word “data” refers to any quantity of bytes included between 0 and 16, if the byte total number required is not higher than 32.

Example:
It is possible to have:
- from 0 to 16 data items of 2 byte
- 1 datum of 4 bytes + from 0 to 14 data items of 2 bytes
- 2 data items of 4 bytes + from 0 to 12 data items of 2 bytes
...
- 8 data items of 4 bytes

The data exchanged via the PDC can be of two types:
-drive parameters
-variables of an MDPlc application

The composition of the PDC input and output data is defined via suitable parameters as described in the paragraphs 3.8.1 and 3.8.2.
The master cyclically writes the data defined as PDC input and cyclically reads the data defined as PDC output.

3.8.1 PDC Input Configuration (SYS_FB_XXX_MS parameter)
See section 1.3.1.

3.8.2 PDC Output Configuration (SYS_FB_XXX_SM Parameter)
See section 1.3.2.

3.8.3 Configuration of the Virtual Digital I/Os
The ADV200 firmware, provides the Word Comp and Word Decomp functions, which allows to exchange discrete signals between the master and the slave and vice versa.
Commands can be sent to the drive using the functions of PAR 4452 **Word decomp**. The meaning of the single bits is programmable. It can be set on a **Fieldbus M->Sn** channel as Count 16.
The drive state is read in PAR 4432 **Word Comp**, programmable on any **Fieldbus S->Mn** channel as Count 16. The meaning of each single bit can be selected by the user using PAR 4400 **Word Bit 0 src** ... PAR 4430 **Word Bit 15 src**.

For a detailed description of these parameters see the drive manual.

3.8.4 Use of the PDC in MDPlc Applications
Refer to section 1.3.3 Using the PDC in MDPLC applications.

3.9 Configuration example

See chapter 1.6.

On paragraph 1.6.2.2 : P4000 **Fieldbus Type** = DeviceNet.

“Polling I/O” input / output area dimensions: with the examples reported on paragraphs 1.6.1.1 and 1.6.1.2 there are 4 bytes for writing and 6 bytes for reading.

References

CiA :	CAN in Automation, user international group.
CAN :	Controller Area Network.
PDO:	Process Data Object, service messages without confirmation used for the real time data transfer from/to the device.
DBT:	Distributor. It is a service element of the CAN Application Layer in the CAN Reference Model; the DBT has the task to assign COB-ID to the COBs used by the CMS.
SDO:	Service Data Object, service messages with confirmation used for the acyclic data transfer from/to the device.
NMT:	Network Management. It is a service element of the CAN Application Layer in the CAN Reference Model; it initializes, configures and controls the errors of a CAN network.
CS:	Command Specifier; it defines the NMT service.
COB-ID	COB-Identifier. It identifies a COB inside the network. It also states the COB priority.