BAUMÜLLER

Control Engineering
Ωmega Drive-Line II

E	5.00005.02

Technical Description
and Operating Instructions

Edition: July 2001

BAUMÜLLER

CONTROL ENGINEERING ΩMEGA DRIVE-LINE II

Technical Description and Operating Instructions

Edition: July, 1st 2001

Document no. 5.00005.02

This operation manual is intended as a complement to the technical description and the operation manual of the apparatus.

BEFORE CARRYING OUT COMMISSIONING, CAREFULLY READ AND OBSERVE THE OPERATING INSTRUCTIONS AND SAFETY INFORMATION

This document contains all the information necessary to correctly use the products it describes. It is intended for specially trained, technically qualified personnel who are well-versed in all warnings and commissioning activities.

The equipment is manufactured using state-of-the-art technology and is safe in operation. It can safely be installed and commissioned and functions without problems if the safety information is followed.

You may not carry out commissioning until it has been established that the machine into which this component is to be installed complies with the specifications of the EC machine guidelines.

This technical description/these operating instructions invalidate all previous descriptions of the corresponding product. Within the scope of further development of our products, Baumüller GmbH reserves the right to change their technical data and handling.

Manufacturer and Supplier's Address: Baumüller Nürnberg GmbH
Ostendstr. 80
D-90482 Nürnberg
Tel. ++49 (0)9 11/54 32 - 0
Fax ++49 (0)9 11/54 32 - 1 30

Country of Origin: Germany

Date of Manufacture: Determined from the serial number on the equipment

TABLE OF CONTENTS

Table of contents

1 SAFETY INFORMATION

General Information

These operating instructions contain all the information necessary for correct operation of the products described. The document is intended for specially trained, technically qualified personnel who are well-versed in all warnings and commissioning activities.

The equipment is manufactured using state-of-the-art technology and is safe in operation. It can safely be installed and commissioned and functions without problems if the safety information in these operating instructions is followed.

Danger Information

On the one hand, the information below is for your own personal safety and on the other to prevent damage to the described products or to other connected equipment.

In the context of the operating instructions and the information on the products themselves, the terms used have the following meanings:

DANGER

This means that **death, severe personal injury**, or **damage to property will** occur unless appropriate safety measures are taken.

WARNING

This means that **death, severe personal injury**, or damage to property **may** occur unless appropriate safety measures are taken.

NOTE

This draws your attention to **important information** about the product, handling of the product or to a particular section of the documentation.

Qualified Personnel

In the context of the safety-specific information in this document or on the products themselves, qualified personnel are considered to be persons who are familiar with setting up, assembling, commissioning and operating the product and who have qualifications appropriate to their activities:

- Trained or instructed or authorized to commission, ground and mark circuits and equipment in acordance with recognized safety standards.
- Trained or instructed in accordance with recognized safety standards in the care and use of appropriate safety equipment.

Appropriate Use

WARNING

You may only use the equipment/system for the purposes specified in the operating instructions and in conjunction with the third-party equipment and components recommended or authorized by BAUMÜLLER NÜRNBERG GmbH.

For safety reasons, you must not change or add components on/to the equipment/system.

The machine minder must report immediately any changes that occur which adversely affect the safety of the equipment/system.

2 TECHNICAL DATA

2.1 General

The Ωmega Drive-Line II is a drive-integrated PLC for implementing distributed intelligent drive technology. For this, you can optionally add to the V-controller the Ωmega Drive-Line II.

The Ωmega Drive-Line II implements the functionality of a drive-integrated PLC, e.g. configurable control engineering, cam disk, position acquisition, digital and analog inputs and outputs or synchronous bus system.

The CANsync synchronous bus system is available for bus communication with peripheral modules. You can link HMIs like operator panels, touchscreens, etc.) via the integrated RS485 port by means of a software interface module to the 3964R® procedure (data block link). As an alternative, you can operate this interface via a software interface module to the USS protocol®, with the Ωmega Drive-Line II functioning as the master that can activate several USS protocol®-capable slaves.

The 3964R® procedure and the USS protocol® are registered trademarks of Siemens AG.

You can carry out open- and closed-loop programming of the Ωmega Drive-Line II either via the standard RS232 port as a point-to-point connection or via the optional Ethernet interface (TCP/IP including the corresponding networking options of several Ωmega Drive-Line IIs in the machine system).

In addition, it is possible to extend the range of functions using two option boards in two option slots. The following option boards are available, for example:

- MFM-01 for digital and analog inputs and outputs.
- IEI-02 for acquiring positions and print marks via two channels.
- CAN-M-01 for the CAN bus link.

You carry out open- and closed-loop programming in a modular way using the PROPROG wt II IEC 61131-3 programming environment in the following programming languages:

- Sequential Function Chart, SFC
- Structured text, ST
- Instruction List, IL
- Function Block Diagram, FBD
- Ladder Diagram, LD

Apart from this, you can use libraries to implement intelligent drive functionality, like:

- Cam disk
- Register controller
- Winder

In addition to the PROPROG wt II IEC 61131-3 programming environment, you can integrate into the global machine concept an OPC server for linking visualization tasks and parameterizations via OPC clients.

2.2 Functionality

- 120 MHz 32-bit RISC-CPU
- 2046 kB of program memory for
  - A maximum of 400,000 IL lines (LD/ST statements to global variables)
  - Typically 120,000 IL lines (typical IL statements to structures and instance variables)
- 2 MB variable RAM (= total storage area of **non retain flags**)
- 1460 kB of dynamic memory for debug and logic analyzer functions
- cycle time of 100 µs per 1000 lines of Instruction list (IL)
- 56 kB of NOVRAM non-volatile data memory (= total storage area of **retain flags**)
- RS232 serial programming interface at 38,400 bps, optically isolated from the Ωmega Drive-Line II
- RS485 terminal interface at a maximum of 38,400 bps, optically isolated from the Ωmega Drive-Line II
- Two option slots for extending the system

  You can fit two types of option boards:

  I/O boards, e.g. IEI (incremental counter module), MFM (digital and analog inputs and outputs)

  Field bus boards, e.g. CAN

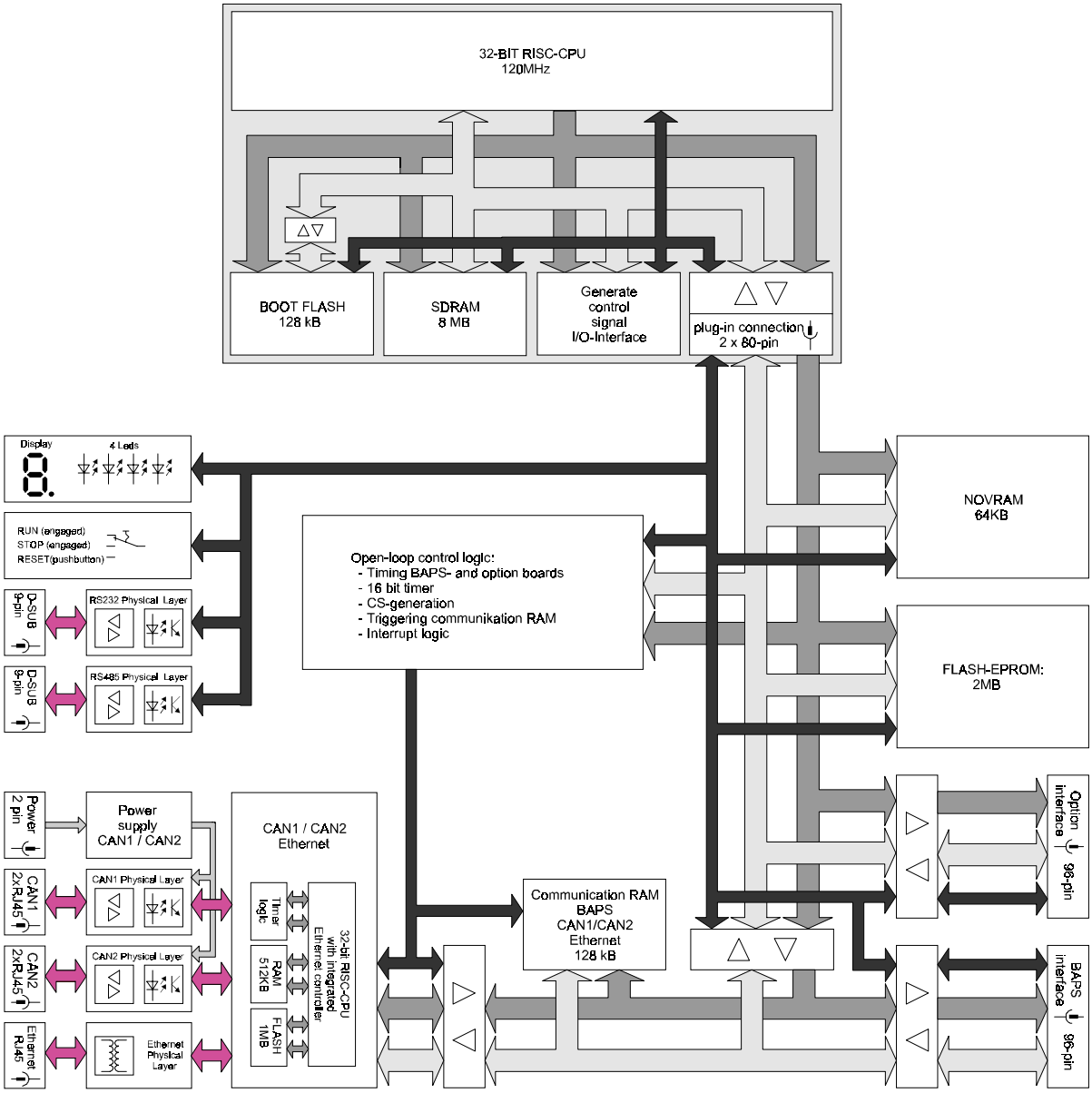
  The following combinations of option boards are possible:

  two I/O boards

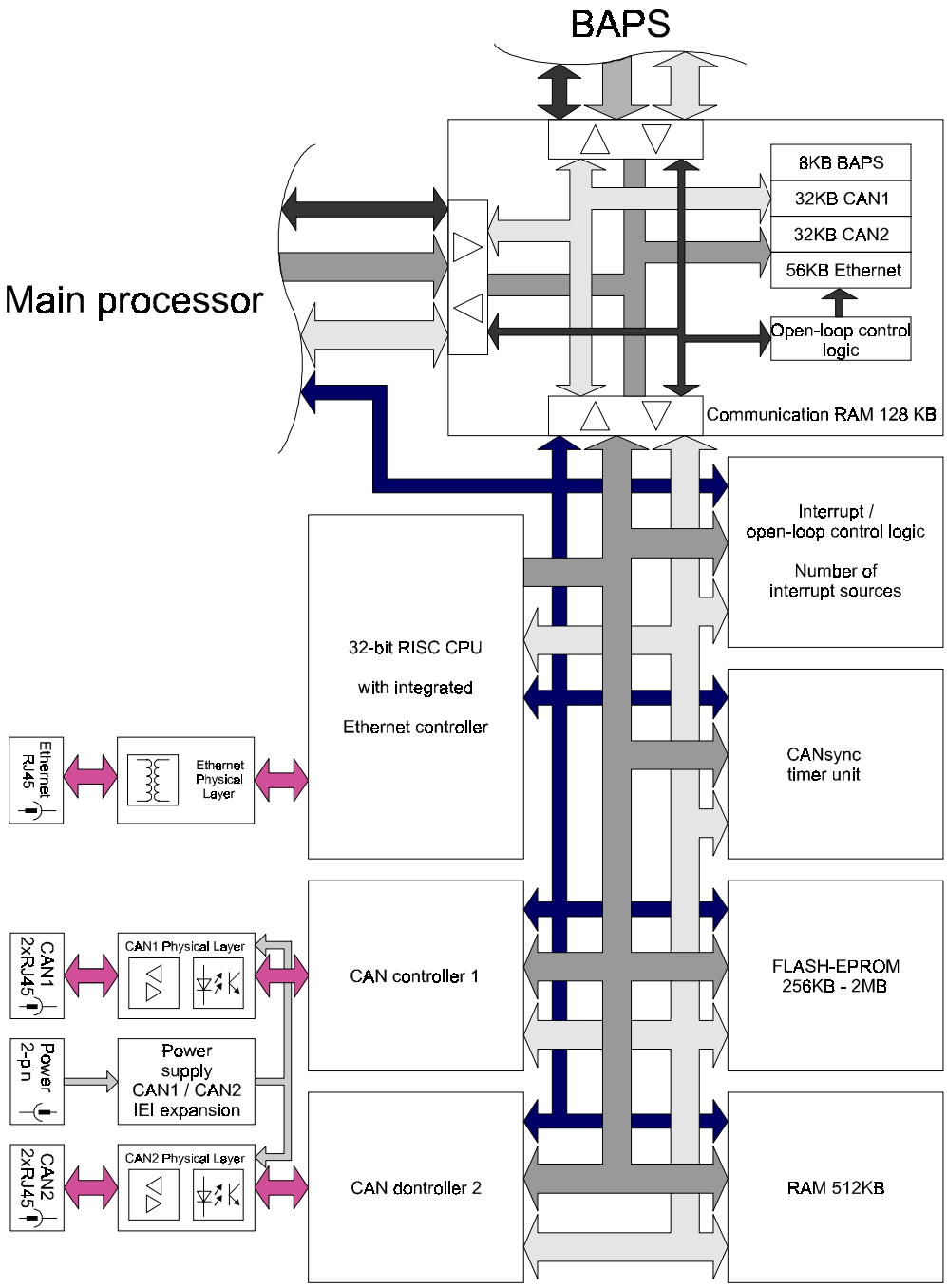
  one I/O board and on field bus board

  You cannot combine two field bus boards.
- Two CANsync nodes at a maximum of 500 kbps optically isolated from the Ωmega Drive-Line II
- 10/100 Mbit Ethernet interface (optional)
- Power consumption 5.5 W

2.3 Functional Structure



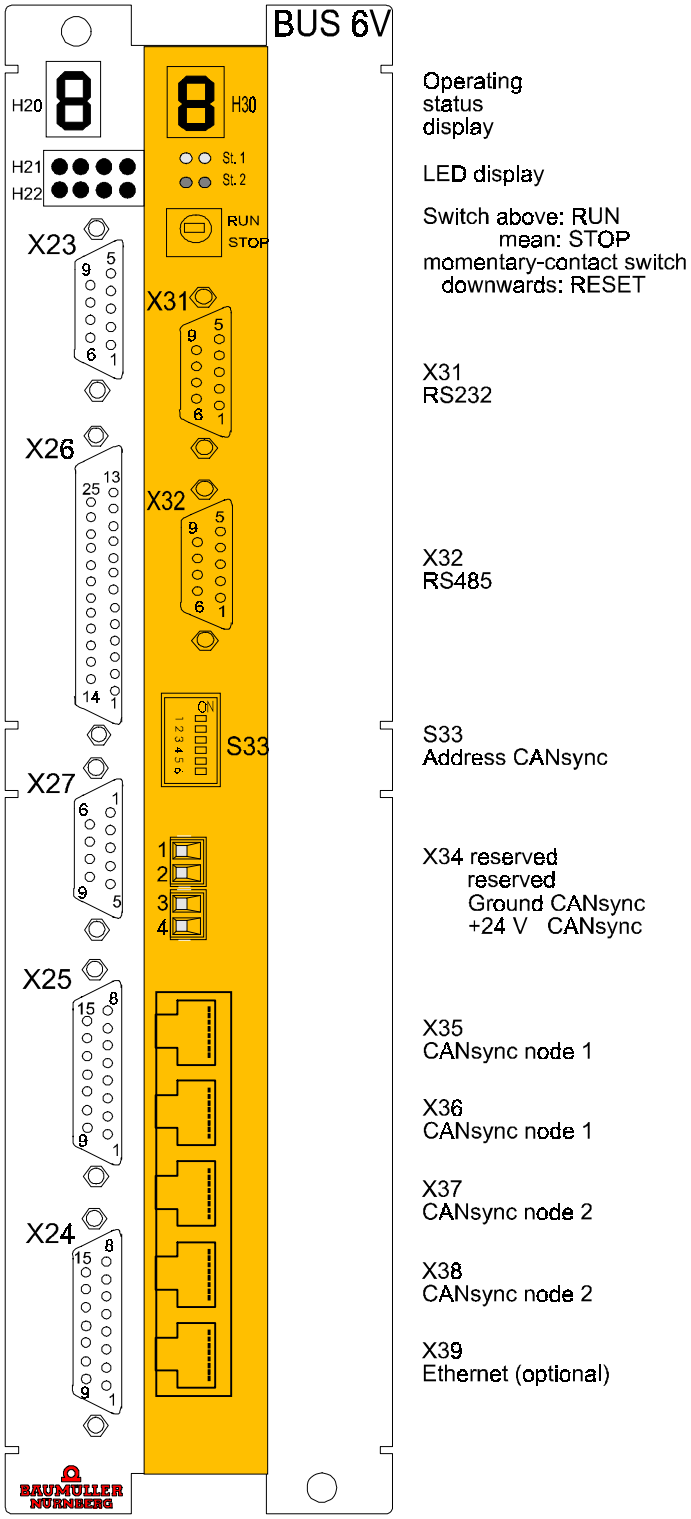
Block diagram of Ωmega Drive-Line II



Block diagram of Ethernet

3 INSTALLATION

3.1 Displays and Operator Controls



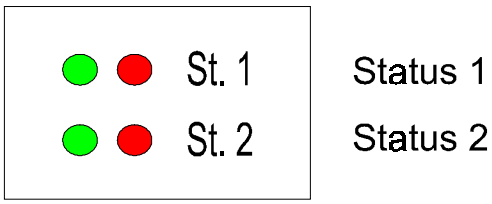
Ωmega Drive-Line II

3.2 Display

3.2.1 Seven-Segment Display

For the meanings of the displayed numbers and letters, refer to The Ωmega Drive-Line II Board Seven-Segment Display on Page 36.

3.2.2 LED Display



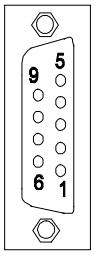
LEDs of the Ωmega Drive-Line II

You can freely program the four LEDs (the green ones on the left and the red one on the right). For programming, refer to The Ωmega Drive-Line II Board Functions on Page 43.

3.3 Pin Assignments

RS232 port (PROPROG wt II)

X 31 SUB-D female connector



SUB-D female connector

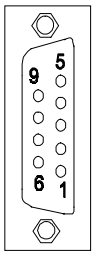
Pin No.	Assignment
1	Not assigned
2	TxD (Transmit Data)
3	RxD (Receive Data)
4	Connected to pin 6
5	GND (Signal Ground)
6	Connected to pin 4
7	CTS (Clear to Send)
8	RTS (Request to Send)
9	Not assigned

NOTE

The signal grounds of the RS232 and RS485 ports are connected together.

RS485 port (terminal interface)

X 32 SUB-D female connector



SUB-D female connector

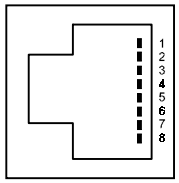
Pin No.	Assignment
1	TxD- (Transmit Data negative)
2	VCC (+5V output for supplying external senders/ receivers)
3	GND (Signal Ground RS232/RS485)
4	GND (Signal Ground RS232/RS485)
5	RxD- (Receive Data negative)
6	RxD+ (Receive Data positive)
7	GND (Signal Ground RS232/RS485)
8	GND (Signal Ground RS232/RS485)
9	TxD+ (Transmit Data positive)

NOTE

The signal grounds of the RS232 and RS485 ports are connected together.

CANsync node 1

X 35/X36 RJ45 female connector



RJ45 female connector

The pins of X35 and X36 with the same numbers are connected together on the printed circuit board, i.e. the CANsync interface is made available on both female connectors.

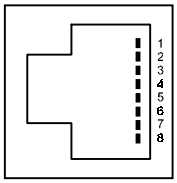
Pin No.	Assignment
1	GND-CAN (Signal Ground CAN)
2	GND-CAN (Signal Ground CAN)
3	Not assigned
4	CAN1-SYNC- (SYNC signal negative node 1)
5	CAN1-SYNC+ (SYNC signal positive node 1)
6	Not assigned
7	CAN1H (CAN bus line dominant high node 1)
8	CAN1L (CAN bus line dominant low node 1)

NOTE

The signal grounds of CANsync node 1 and node 2 are connected together.

CANsync node 2

X37/X38 RJ45 female connector



RJ45 female connector

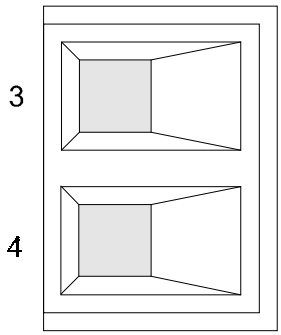
The pins of X3 and X38 with the same numbers are connected together on the printed circuit board, i.e. the CANsync interface is made available on both female connectors.

Pin No.	Assignment
1	GND-CAN (Signal Ground CAN)
2	GND-CAN (Signal Ground CAN)
3	Not assigned
4	CAN2-SYNC- (SYNC signal negative node 2)
5	CAN2-SYNC+ (SYNC signal negative node 2)
6	Not assigned
7	CAN2H (CAN bus line dominant high node 2)
8	CAN2L (CAN bus line dominant low node 2)

NOTE

The signal grounds of CANsync node 1 and node 2 are connected together.

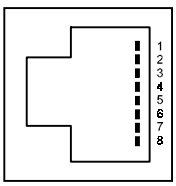
24-V supply voltage of Ωmega Drive-Line II (X34) for SYNC signal



Pin No.	Assignment
3	24 V ground
4	+24 V supply voltage

10/100 Mbit Ethernet interface module (optional)

X39 RJ45 female connector

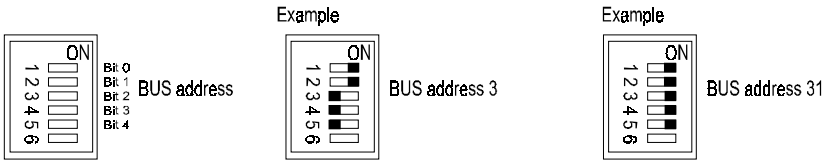


RJ45 female connector

Pin No.	Assignment
1	TX+ (Transmit line +)
2	TX- (Transmit line -)
3	RX+ (Receive line +)
4	Not assigned
5	Not assigned
6	RX- (Receive line -)
7	Not assigned
8	Not assigned

3.3.1 Setting the Slave Number

Using DIP switches 1-5 (S 33) on the Ωmega board (the middle board in the cassette), you set the binary-coded slave number with which the CANsync slave interface module is to be operated on the Ωmega Drive-Line II. You must set for each CANsync slave interface modules on a CANsync bus a unique slave number that must be different from all the rest (one CANsync master interface module and up to 32 CANsync slave interface modules).



3.3.2 Information on Configuration

There are two CANsync interface modules on the printed circuit board. They are used to connect a CANsync bus to a CANsync bus of a sublevel. This forms a network with several levels.

You must configure on the printed circuit board one interface module as a CANsync slave and one interface module as a CANsync master (slave for SYNC signal).

The functionality of the CANsync master and the CANsync slave interface modules is described in chapter 6.

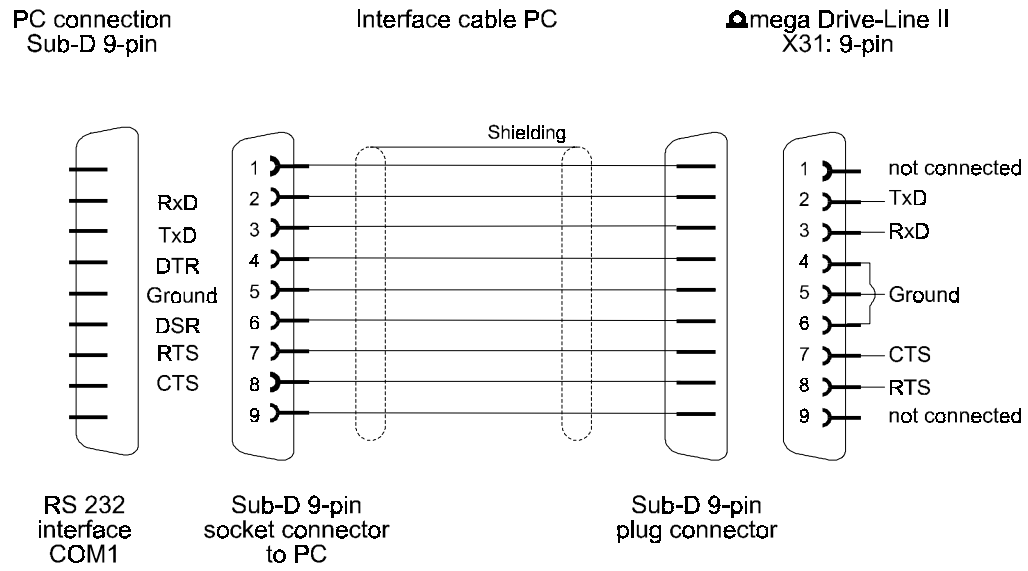
NOTE

You can change in software the preset IP address for Ethernet (192.168.1.“1+DIP switch“, DIP switch = Bit 4 ... 0). See “Setting the IP Address and the IP Mask” on page 73 and “Communication via Ethernet (optional)” on page 30.

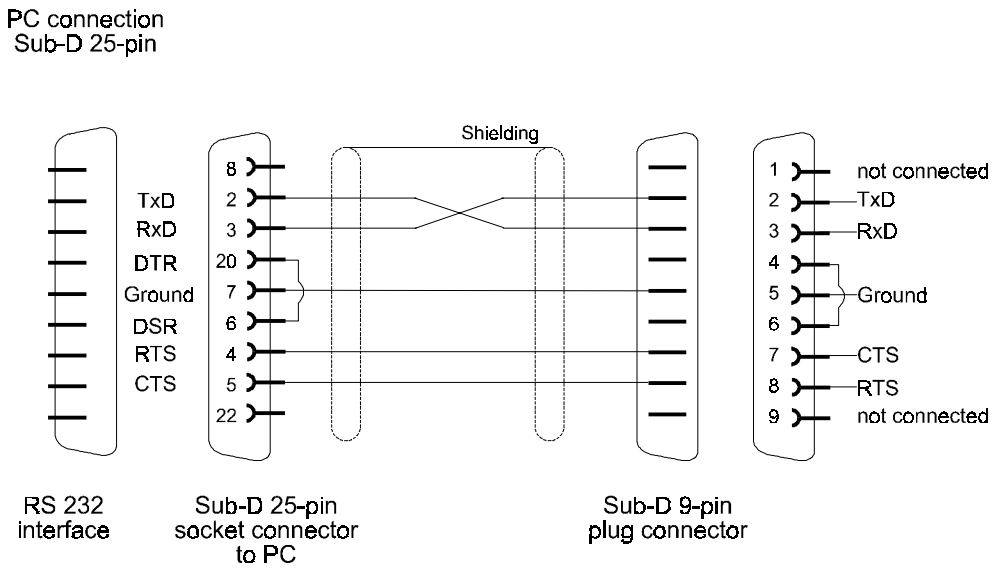
3.4 Connecting cables

Serial connecting cable for PC to Ωmega Drive-Line II

- 9-pin PC connection

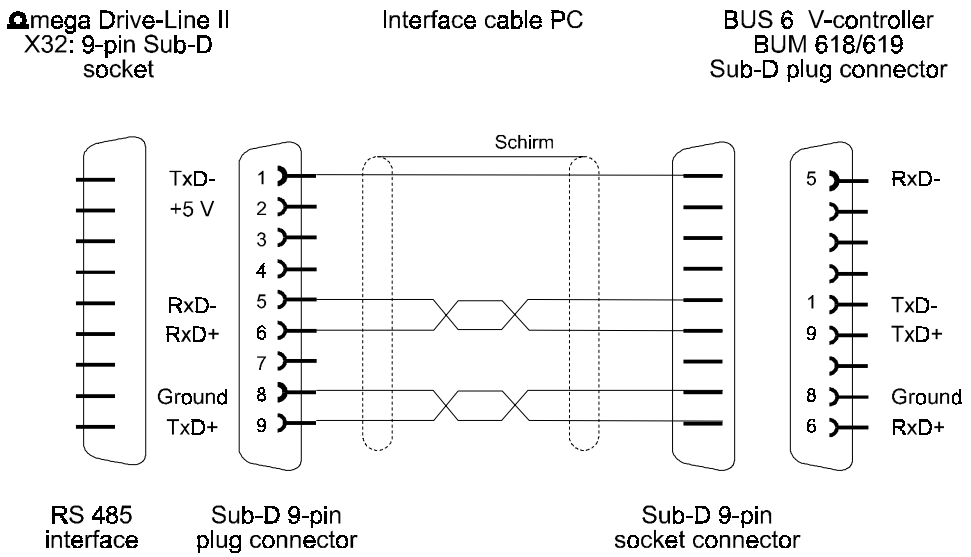


- 25-pin PC connection

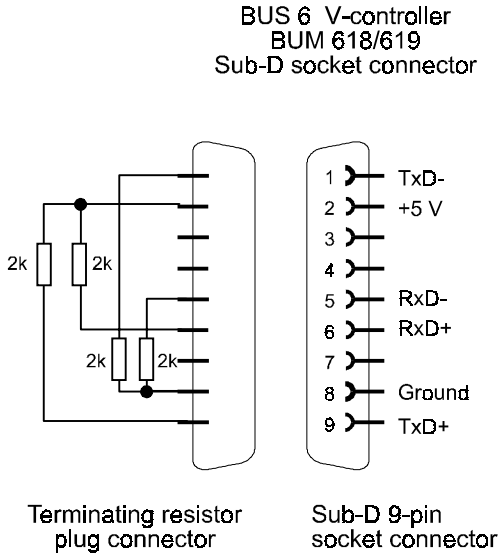


Serial Connecting cable for RS485 (drive networking)

- Ωmega ⇔ BUS 6 V-controller (4-wire connection and ground with 9-pin male connector)



- Interconnection of the last node in the ring (e.g. 9-pin terminal female connector at the controller)



NOTE

The +5 V are for supplying RS485/RS232 adapters and **must not** be connected together in the ring.

The last node in the ring should be terminated with the resistor network shown above.

3.5 Accessories

Cable for programming interface (serial RS232)

Line type: K-SS-01-xx (9-pin male, 9-pin female)

TYPE	Length [m]	Article Number
K-SS-01-03	3	213 846
K-SS-01-05	5	213 283
K-SS-01-15	15	231 086

Cable for CANsync/CAN

Line type: K-CAN-33-xx (RJ male connector, RJ male connector):

TYPE	Length [m]	Article Number
K-CAN-33-0-0,5	0.5	324 497
K-CAN-33-0-01	1	324 498
K-CAN-33-0-02	2	324 499
K-CAN-33-0-03	3	324 500
K-CAN-33-0-04	4	324 501
K-CAN-33-0-05	5	324 502
K-CAN-33-0-10	10	324 503

Line type: K-CAN-13-xx (RJ male connector, SUB-D male connector):

TYPE	Length [m]	Article Number
K-CAN-13-0-0,5	0.5	324 504
K-CAN-13-0-01	1	324 505
K-CAN-13-0-02	2	324 506
K-CAN-13-0-03	3	324 507
K-CAN-13-0-04	4	324 508
K-CAN-13-0-05	5	324 509
K-CAN-13-0-10	10	324 510

Terminating resistor connector for CANsync/CAN

K-CAN-T1-S (D-SUB 9-pin, male) Art. No. 313 910

K-CAN-T2-S (D-SUB 9-pin, female) Art. No. 313 911

K-CAN-RJ Art. No. 313 264

Cable for Ethernet

Line type: K-ETH-33-xx (RJ male connector, RJ male connector, cable CAT5):

TYPE	Length [m]	Article Number
K-ETH-33-0-0,5	0.5	325 160
K-ETH-33-0-01	1	325 161
K-ETH-33-0-02	2	325 162
K-ETH-33-0-03	3	325 163
K-ETH-33-0-04	4	325 317
K-ETH-33-0-05	5	325 164
K-ETH-33-0-10	10	325 165

4 ΩMEGA DRIVE-LINE II AND PROPROG WT II

4.1 PROPROG wt II – an Efficient, Powerful and Comprehensive Programming Tool

PROPROG wt II is a standard programming system that is based on the IEC 61131-3 standard.

The programming system provides powerful functions for the various development stages of PLC applications, like:

- editing
- compiling
- debugging
- printing

The PROPROG wt II programming system is based on modern 32-bit Windows technology and allows users to operate the system easily using tools like zoom, scroll, special toolbars, drag & drop, a shortcut manager and dockable windows.

In particular, the system makes possible processing of several configurations and resources within a project as well as integration of project libraries. Apart from this, it has a powerful system for debugging. Using the easy-to-use project tree editor, you can display and edit projects. This makes it possible to represent easily and transparently the complex structure of the IEC 61131-3 standard. This feature allows users to easily paste and edit in the project tree POUs, data types, libraries and configuration elements.

The PROPROG wt II programming system consists of a PLC-independent core for programming in the various IEC programming languages: these include the text languages Structured Text (ST) and Instruction List (IL) as well as the graphic languages Function Block Diagram (FBD), Ladder Diagram (LD) and Sequential Function Chart (SFC).

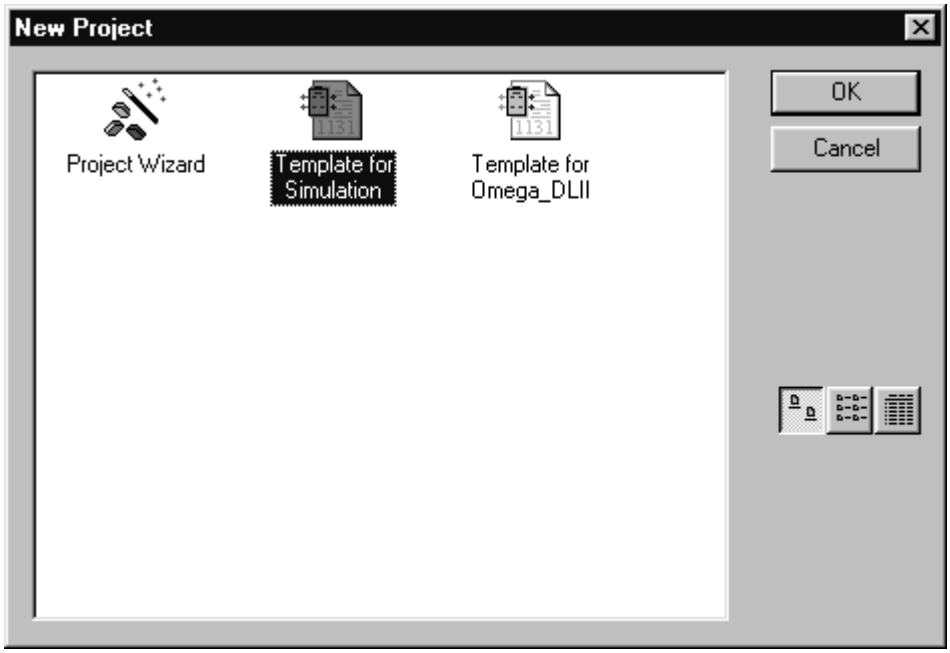
An editor wizard is available for programming in each of the languages, which allows you to quickly and easily paste prepared keywords, statements, operators, functions and function blocks into the individual work sheets. You can also use the editor wizard for declaring variables and data types. The independent core of the system is complemented by special sections that are matched to various PLCs.

The new easy online handling and the 32-bit simulation allow you options for debugging the address status and a real-time multitasking test environment.

An easy-to-use tool for project documentation allows you to print the project in a time-saving optimized form (using less paper) with a page layout that can be specified by individual users.

4.2 Ωmega Drive-Line II Project

You start a new Ωmega Drive-Line II project under PROPROG wt II by choosing menu item New project. Using „Template for Omega_DLII“, you open an Ωmega Drive-Line II project. The Ωmega Drive-Line II CPU in this project is the default setting in the configuration.



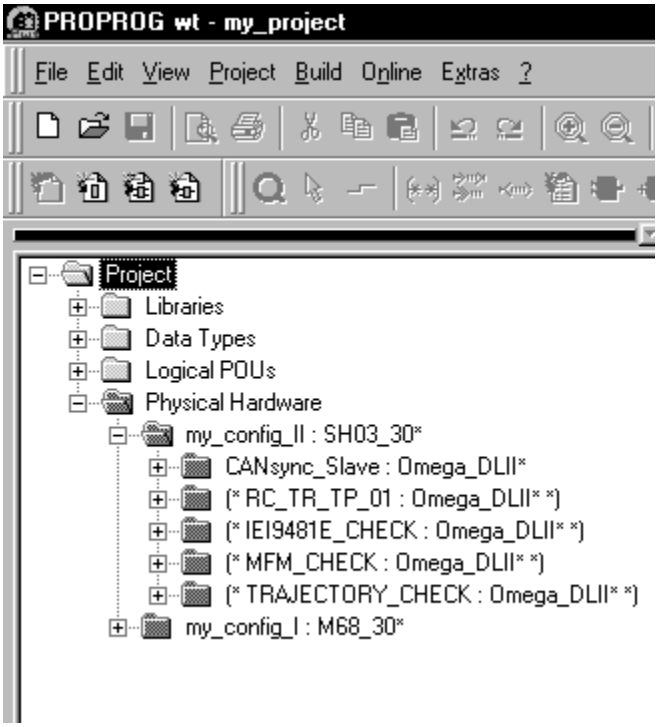
Ωmega Drive-Line II project template under „New project“

4.3 Ωmega Drive-Line II Configuration

As an IEC 61131-3 programming environment, you can use PROPROG wt II to program different target systems (CPUs). It is also possible to program different target systems in one project. You generate a program for the Ωmega Drive-Line II target system under „Physical Hardware“ using the Ωmega Drive-Line II configuration.

Using the template, you open an Ωmega Drive-Line II configuration. Under its Properties, the Ωmega Drive-Line II configuration has PLC type SH03_30.

A configuration consists of at least one resource. The resource contains the Ωmega Drive-Line II-specific data area, the communications source, the global variable worksheets and the tasks with the program.



Setting under the „Physical Hardware“ property of an Ωmega Drive-Line II configuration.

Overview of the settings for inserting an Ωmega Drive-Line II target system within the physical hardware:

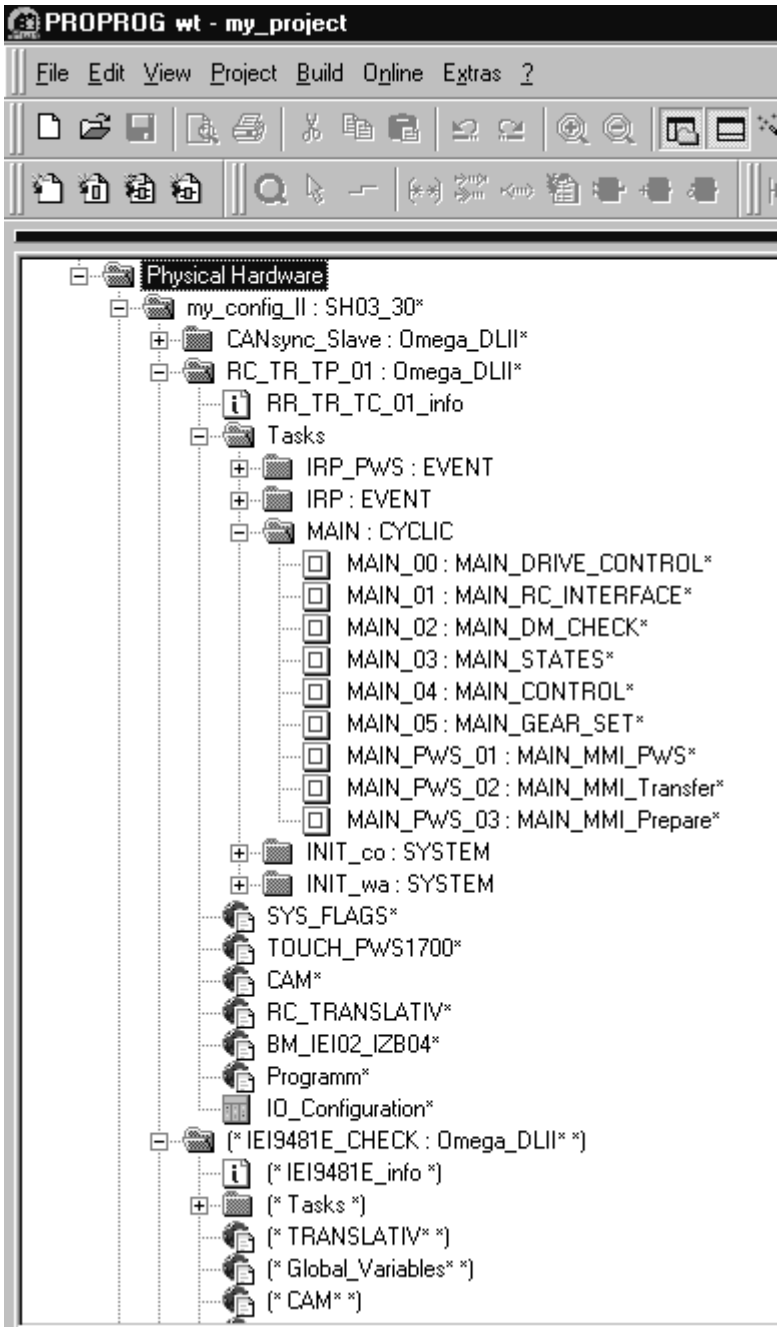
Project template	Configuration	Resource
Omega_DLII	SH03_30	Omega_DLII

4.4 Ωmega Drive-Line II Resource

The resource contains the Ωmega Drive-Line II-specific settings for a program:

- Data area
- Communications source
- Global variable worksheets
- Various tasks using the program
- Documentation worksheets and I/O configuration
  (The I/O configuration is already set correctly and you don't have to change it.)

You can assign several resources to the Ωmega Drive-Line II configuration. This makes it possible to implement a complete application with several drives in one project.



Example of an Ωmega Drive-Line II configuration with several resources.

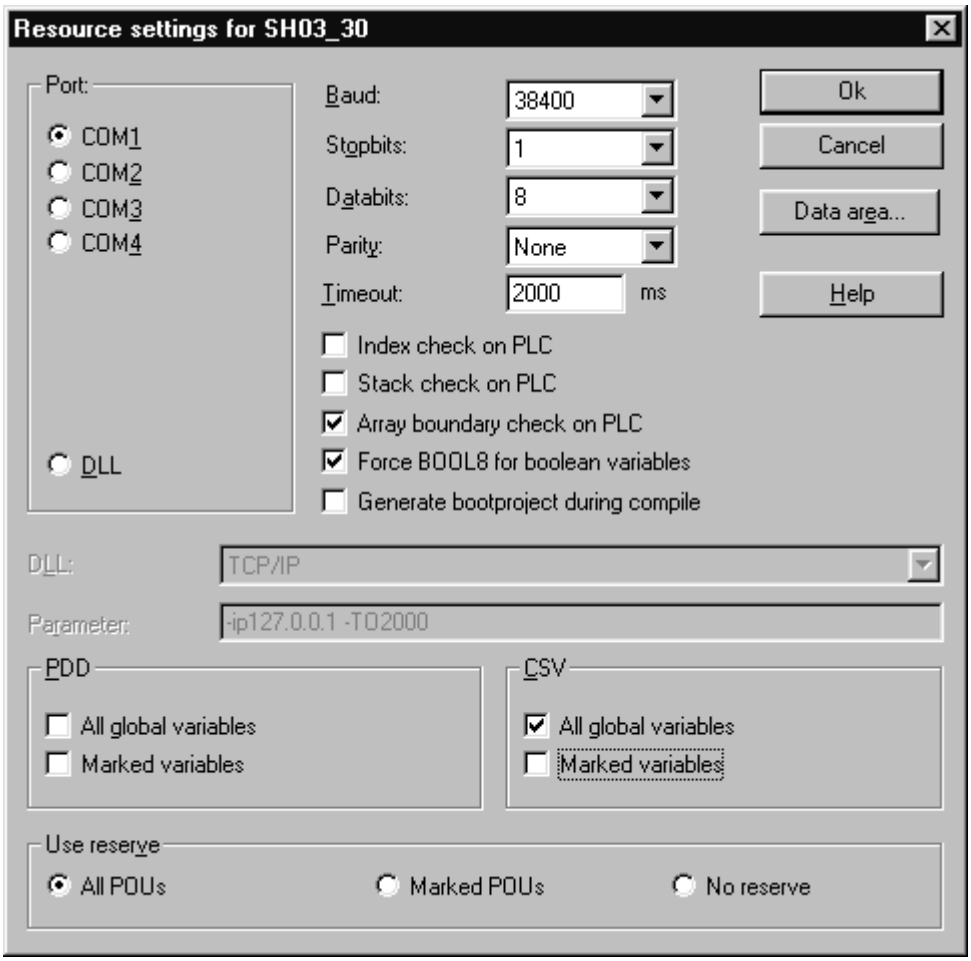
4.4.1 Communication and Connection

You configure communication with data transfer under „Setting“ in the resource's context menu.

You set communication via the selected RS232 port as follows:

- Baud: 38400
- Stop bits: 1
- Data bits: 8
- Parity: None
- Timeout: Default is 2000 ms; communications monitoring during online representation.

The connection is established via X31 on the Ωmega Drive-Line II.



Resource setting within an Ωmega Drive-Line II configuration.

- „Index check on PLC“: The system checks the declared field size (index) of an ARRAY at runtime. Important: this increases the code execution time!
- „Stack check on PLC“: The system checks for a stack overrun at runtime. The stack memory is reserved with the program task. There is an increase in the data on the stack if you program nested FB instances, for example. Important: this increases the code execution time!
- „Array boundary check on PLC“: With absolute addressing, the system checks whether the field violates the parameterized data area limits (retain, non retain). This check is carried out during compilation on the PC. This does not increase the code execution time.

- „Force BOOL8 for boolean variables“: Activate 8-bit access to Boolean variables. This setting must be activated for the Ωmega Drive-Line II.
- „Generate bootproject during compile“: With this function activated, the system generates for each resource a bootfile.pro at compiling of the project and not just at activation of the Send boot project function via the resource control.
- „PDD": Settings for the process data directory. (See also the PROPROG wt II manual).
- „CSV": Settings for providing variables for the OPC server. (See also the PROPROG wt II programming manual).
- „Use reserve“: Use spare memory to be able to make changes in the function blocks and functions using the „Patch POU function“. (See also the PROPROG wt II programming manual).

You can make the resource setting separately for each Ωmega Drive-Line II resource. Serial or Ethernet communications source

- COM1
- COM2
- COM3
- COM4
- DLL

is used

  – for resource control.
  – for sending the compiled project.
  – for debugging.
  – for connecting to the OPC server.

Communication via Ethernet (optional)

You optionally set communication via Ethernet (for application see “Ethernet” on page 71.) on the Ωmega Drive-Line II socket X39 as follows:

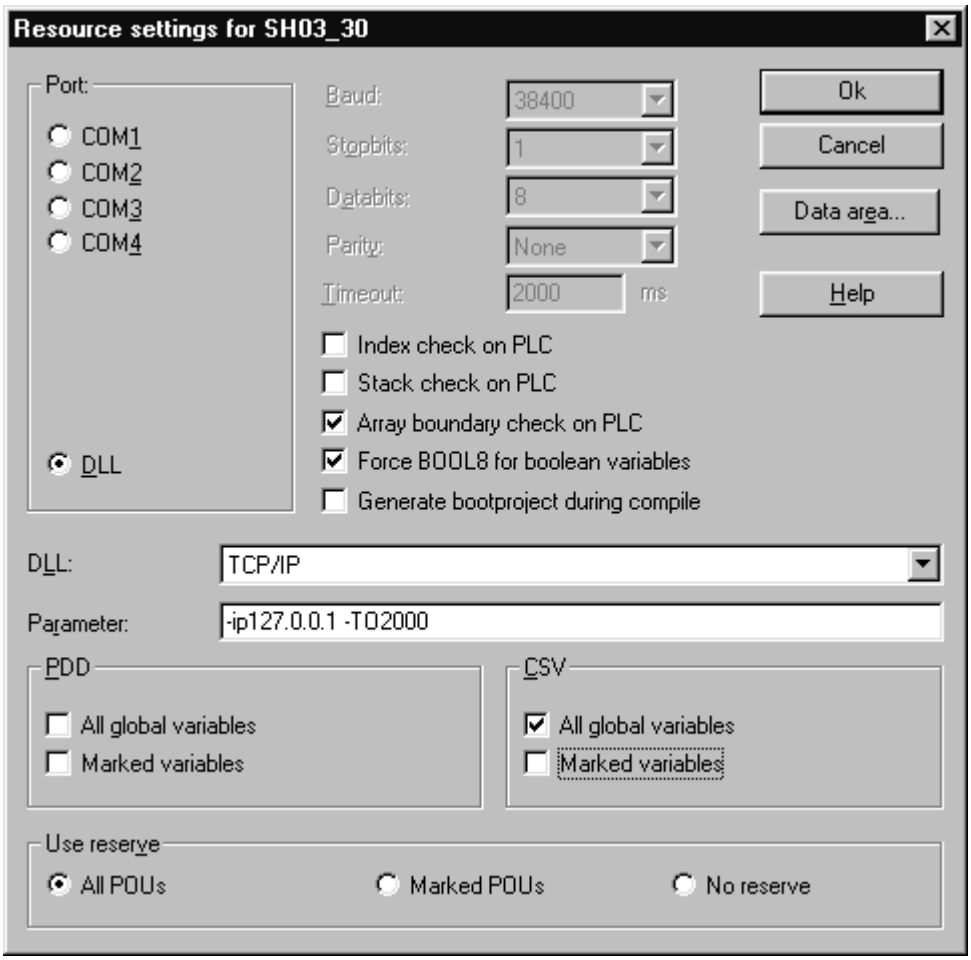
- In Settings, you must choose DLL. In the DLL window, you can choose from the named saved IP addresses or enter an IP address manually in the Parameters window (e.g. -ip 192.168.1.1 for the default setting with DIP-switch = 0).
- To save IP addresses with their names, you must enter the IP addresses in the PROPROG wt II file (\ProProgwt\PROPROG wt\) **mwt.ini.** For this, you must enter names under the country code, e.g. English 001, German 049.

Example: mwt.ini with Ethernet address entries for selection via a synonym.

```
[COMMUNICATION]
DLL001=plc\socomm.dll -ip 127.0.0.1 -TO2000
DLL003=plc\socomm.dll -ip 192.075.191.183 -TO2000
DLL002=plc\socomm.dll -ip 192.075.191.184 -TO2000
DLL004=plc\socomm.dll -ip 192.075.191.185 -TO2000

[COMMUNICATION001]
NAME001=TCP/IP
NAME002=TCP/IP DLII-Address 1
NAME003=TCP/IP DLII-Address 2
NAME004=TCP/IP DLII-Address 3

[COMMUNICATION049]
NAME001=TCP/IP
NAME002=TCP/IP DLII-Adresse 1
NAME003=TCP/IP DLII-Adresse 2
NAME004=TCP/IP DLII-Adresse 3
```

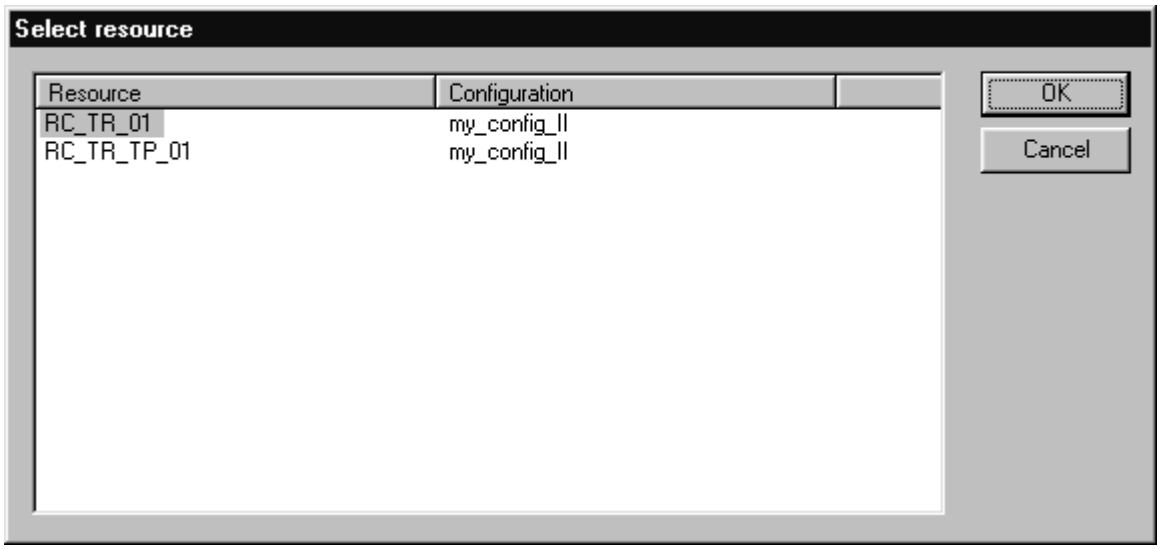


Setting a DLL for communication via Ethernet with address via a synonym or manual input.

4.4.2 Control Dialog for Resources

Using the control dialog for resources, you set program transfer to the Ωmega Drive-Line II and the operating status of the Ωmega Drive-Line II.

If several resources are active in a project, you must choose the desired resource.

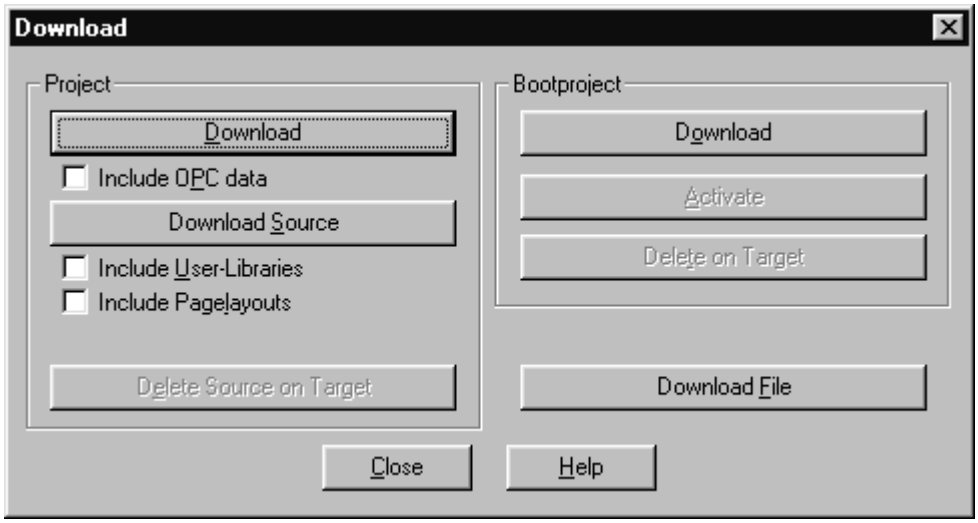


After you choose the resource, the system displays the control dialog for the resource of the assigned Ωmega Drive-Line II.

Functions of the control dialog of the selected resource in the "RUN" status.

Clicking on **„Download“** transfers the compiled project to the target system.

Downloading the project to the target system.



Transferring Ωmega Drive-Line II resource to flash memory or RAM.

Clicking on „**Download Bootproject“**, deletes the resource's current bootproject, sends the compiled project as a boot project and saves it in the Ωmega Drive-Line II's flash memory. Clicking on **Activate,** loads the project from the Ωmega Drive-Line II's flash memory to RAM.

Clicking on **„Delete on Target“**, deletes the bootproject in flash memory.

Clicking on **„Download Project“**, transmits the resource's compiled project. The bootproject stays unchanged in the Ωmega Drive-Line II's flash memory. After the next hardware reset or the next time you switch the controller off and on again, the bootproject is active again!

NOTE

The system does not currently support menu items „Download Source“, „Download File“ and the „Include OPC data“ attribute and you mustn’t select these items.

After clicking on **Dowloand Project,** you can start the project on the Ωmega Drive-Line II.
If a boot project is sent, you can use **Download** and **Activate Bootproject** to load the project in the Ωmega Drive-Line II and then start it:

The Ωmega Drive-Line II resource control in the "STOP" status.

- **Stop:** Stops execution of the program.
- **Reset:** Deletes the project on the Ωmega Drive-Line II (not the boot project!)
- **Download:** Calls program transfer.
- **Cold boot:** The system carries out the system task cold boot with all the variables being initialized with their default values. If a new project is sent, the system carries out a cold boot once when the hardware switch is set to "RUN". At the same time, retain variables are set to their initialization values.
- **Warm boot:** The system carries out the system task warm boot with the hardware switch set to RUN. The global retain variables retain their last values. This is only possible when the hardware switch is set to "RUN".

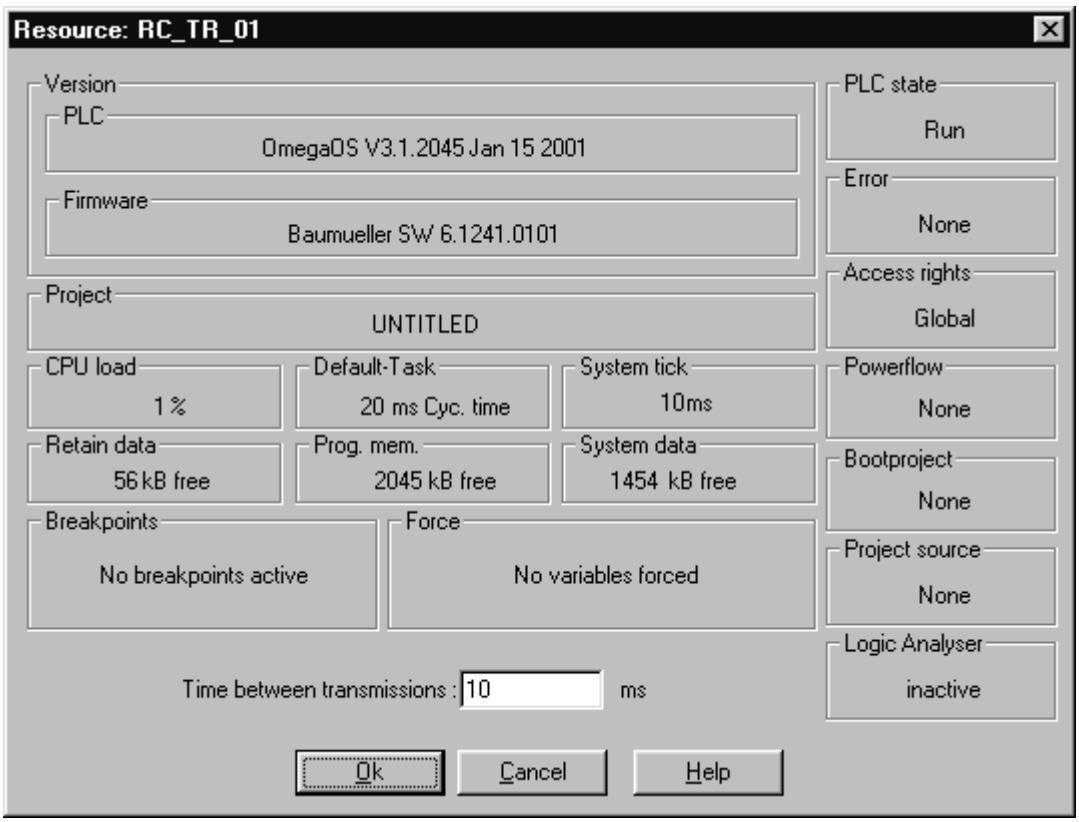
NOTE

If you set the hardware switch from STOP to RUN after a „Download Project", the system carries out a cold boot once. Every other time that you change from STOP to RUN, the system carries out a warm boot.

- **Hot boot:** The system does not run through an initialization task, i.e. neither a cold boot nor a warm one, but rather executes the cyclical project directly.
- **Error:** Here, you can read out error and warning messages that are pending on the controller if the pushbutton is activated. The button is activated if the system generates a runtime error message. Clicking on the active error pushbutton inquires the controller's error entries and displays them in the error or warning message window.
- **Upload:** Upload function: is not currently supported.
- **Info:** Information about the OmegaOS version, the firmware number, the used memory configuration and the internal system status conditions of the controller.

Under „Info“, the control dialog for the resource provides general information about the project on the Ωmega Drive-Line II. The following content is shown:

- „Version PLC“ with the version of the Ωmega Drive-Line II runtime system.
- „Version Firmware“ with the version of the Ωmega Drive-Line II firmware for the firmware library.
- „Project" with the name of the active project.
- Operating and error status as well as the active debug mode, breakpoints, forces.
- Total storage area of retain data.
- Free program memory.
- Free memory for system data runtime system (for loading, oscilloscope function).
- CPU loading, e.g. to check loop repetitions in a parameterized task time.
- Update time of the online representation in debug online mode.



Ωmega Drive-Line II resource information.

4.4.3 The Ωmega Drive-Line II Board Seven-Segment Display

You can execute the Start/Stop functions either using the resource's control dialog of the PROPROG wt II or by means of the RUN/STOP switch. If you intend to carry out a start using PROPROG wt II, you must set the RUN/STOP switch to the "RUN" position.

The RUN/STOP switch on the Ωmega Drive-Line II board has the following switching positions:

Pushbutton/switching position	Status	Seven-Segment Display
Bottom (pushbutton)	**RESET**	Not lit while you keep pressing the pushbutton. When you release the pushbutton it automatically changes to the STOP middle position.
Middle (switch)	**STOP**	
	Without project	**0**
	With project	**1**
Top (switch)	**RUN**	
	(Control of PROPROG wt II resource manager active)	
	No project on the controller.	**0**
	Using the resource's control dialog, a Stop was carried out or a new project was activated.	**1**
	The system runs through initialization. (Cold or warm boot)	**2**
	Initialization phase is completed; the system is currently executing the cyclical program sections.	**3**
	The system is currently deleting the boot project on the controller.	**C**
	The system is currently copying a new boot project into flash memory.	**L. < ---- > L**
	The system has completed copying a new boot project into flash memory. You can now activate the boot project.	**A**

4.4.4 Data Area

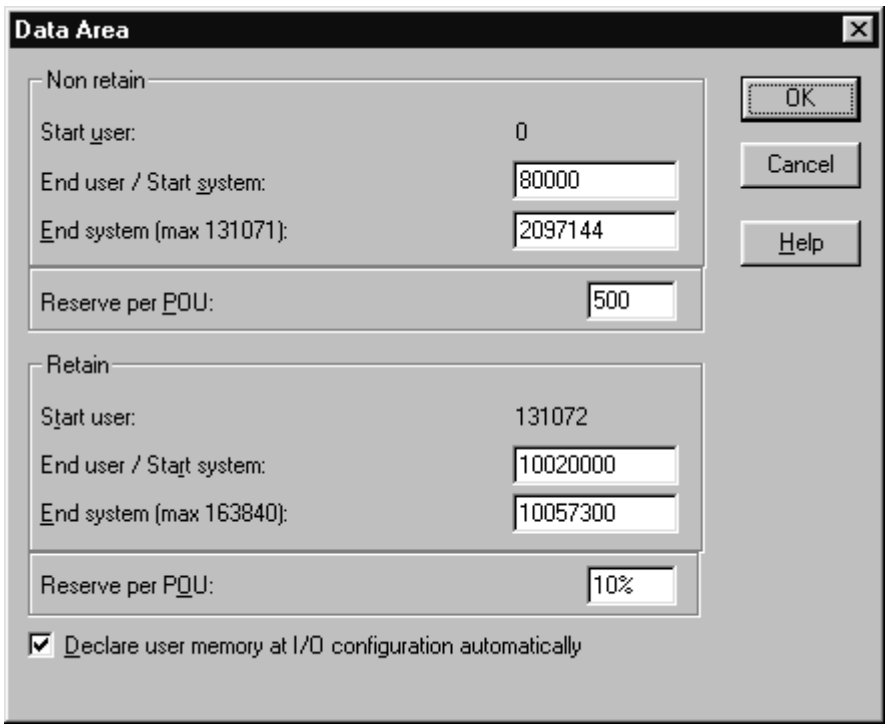
The data area contains the Ωmega Drive-Line II-specific setting of the physical address range.

The data area is divided into **non retain flags** and **retain flags.** A system area is located in both areas. In the system area, the system assigns the addresses of the symbolic program variables via the compiler.

Users can assign absolute addresses in both the non retain and retain user areas.

Apart from this, there is an Ωmega Drive-Line II-specific area for interfaces, e.g. option interfaces (see “The Base Addresses of the Option Interfaces” on page 56.).

The reserve is for the send Online Changes compiler functionality and reserves program memory.



Resource Ωmega Drive-Line II-specific data range without a retain flag user area.

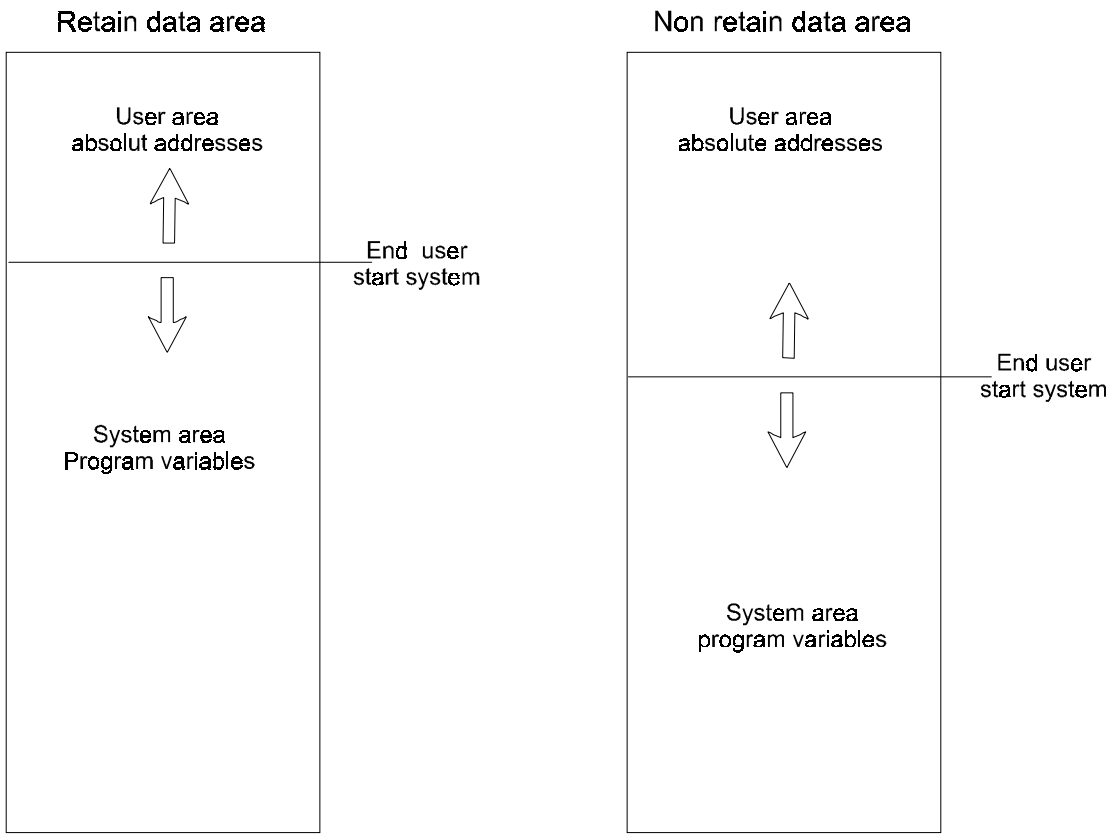
The Ωmega Drive-Line II data range is divided into retain and non retain data areas. You must set the area of the absolute addresses project-dependently.

The standard user area for assigning absolute addresses within an Ωmega Drive-Line II resource in the **non retain** area is:

- %MB 0 - %MB 79999

The standard user area for assigning absolute addresses within an Ωmega Drive-Line II resource in the **retain** area is:

- %MB 10000000 - %MB 10019999



Breakdown and setting of the Ωmega Drive-Line II data range.

Assigning addresses in the program:

An absolute Ωmega Drive-Line II address or a variable field with data type

- 16-bit (WORD) can only be assigned for an address that can only be divided without remainder by two and zero.
- 32-bit (DWORD) can only be assigned for an address that can only be divided without remainder by four and zero.

Example:

You want to declare a variable of data type DWORD in the non retain data range.

```
dw_abs  AT  %MD12       : DWORD;      (* symbolic variable to absolute
                                      address *)
```

4.4.5 The Ωmega Drive-Line II Event Tasks

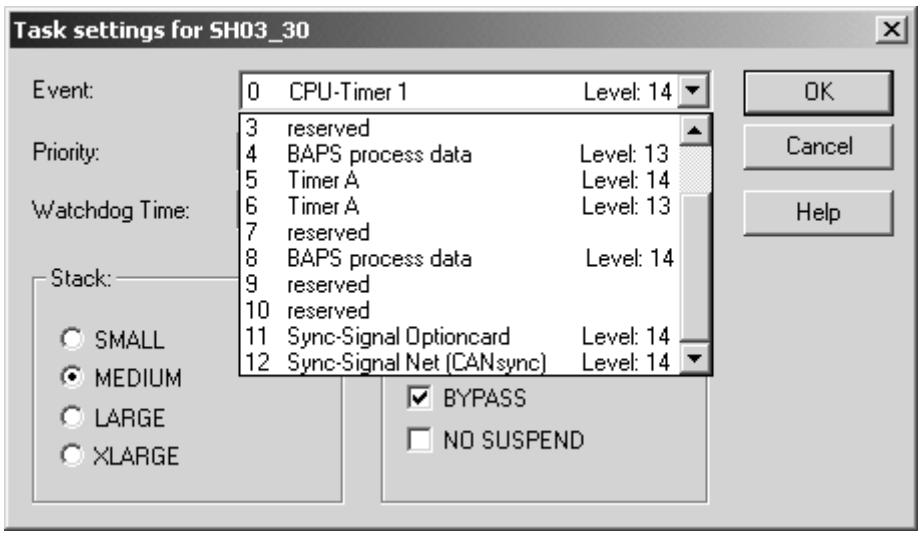
The Ωmega Dreference valuerive-Line II event tasks are for event-driven calling of the program. Their type and code runtime determine the real-time response.

Implementation of the real-time response depends on the type of reference value setting and the V-controller's operating mode. Reference value setting can, for example, be implemented as follows:

- In a standalone drive via a BAPS event task.
- In a networked drive via a CANsync event task.

The property of the event task is assigned via its event number:

Property	**Ωmega Drive-Line II event**	**Ωmega Drive-Line II interrupt level**
Event 0	CPU timer 1	Level 14
Event 1	Reserved	
Event 2	CPU timer 2	Level 13
Event 3	Reserved	
Event 4	BAPS process data	Level 13
Event 5	Timer A	Level 14
Event 6	Timer A	Level 13
Event 7	Reserved	
Event 8	BAPS process data	Level 14
Events 9, 10	Reserved	
Event 11	SYNC signal option board	Level 14
Event 12	SYNC signal network (CANsync)	Level 14



The events of the Ωmega Drive-Line II event tasks.

NOTE

All the Ωmega Drive-Line II event tasks depend on the resource and need the **Bypass** attribute. The initialization FBs within the program initialize and call the declared event task and their priorities. This means that with bypass event tasks, the priority, watchdog time, "SAVE FPU" and "NO SUSPEND" are meaningless.
A higher interrupt level means a higher priority.
In the case of several event tasks that can mutually interrupt one another or with multiple-nested function blocks, you should adapt the stack to Large or XLarge.

4.5 Ωmega Drive-Line II User Libraries

The PROPROG wt II user libraries are divided into firmware and user libraries that can be hardware-dependent or hardware-independent. You can only use hardware-dependent libraries in resources of the specified target system. The hardware dependence of Ωmega Drive-Line II libraries is indicated by **_DLII_** in the library designation.

All Ωmega Drive-Line II user libraries start with Version 2.0 Build 0. The version is stated as follows: 20bd00.

- 20 is the incompatible version.
- 00 is the compatible version.

In the case of compatibility of the input and output variables of the function blocks, the version is incremented by one, e.g. 20bd01, 20bd02, etc. if you add the library to an FB, e.g. an input or output, the version before the "bd" is incremented by one and set to zero after the "bd", e.g. 20bd03 to 21bd00.

The user libraries are divided into:

- Firmware: Ωmega Drive-Line II board functionality, e.g. starting a PROPROG wt II bypass event task
- Data types: Ωmega Drive-Line II-specifically assembled data types and fields, e.g. register structure of option boards
- Standard FBs: Elementary FBs for local control engineering
- Technology components: Completely deployable drive functionality.

Overview of Ωmega Drive-Line II libraries under PROPROG wt II (subject to change):

Data types:

BM_TYPES_20bd00 assembled data types and fields.

Ωmega Drive-Line II system FBs:

SYSTEM1_DLII_20bd00 BAPS and serial communication, trigger signal interconnection, code runtime measurement.

Ωmega Drive-Line II firmware:

SYSTEM2_DLII_20bd00 The entire Ωmega Drive-Line II firmware.

Local control engineering:

UNIVERSAL_20bd00 Hardware-independent FBs like controllers or reference value generators.

IEI-02 option board (optional):

IEI_DLII_20bd00 IEI-02 initialization.

Asynchronous CAN bus with CAN-M-01 option board (optional):

CAN_DLII_20bd00 FBs for a CAN bus link with CAN-M-01 option board.

CANsync synchronous bus system:

CANsync_DLII_20bd00 FBs for a CANsync bus link.

Technology components (optional):

WINDER_DLII_20bd00 winder.

CAM_DLII_20bd00 cam disk.

REGISTER_DLII_20bd00 register controller.

You must state the directory path for libraries under PROPROG wt II, Options, Directories. The Ωmega Drive-Line II libraries are inserted in the PROPROG wt II project tree under libraries.

You can call HTML help for each FB that gives you a description of the inputs and outputs (see the PROPROG wt II Manual).

4.5.1 Ωmega Drive-Line II Firmware

The Ωmega Drive-Line II firmware consists of function blocks (FBs) that communicate with functions on the Ωmega Drive-Line II-CPU via transfer parameters. You can only use these FBs in a resource-dependent way, i.e. in dependence on the target system.

You must insert the Ωmega Drive-Line II firmware in a project with the library

SYSTEM2_DLII_20bd00 (or above)

The library contains the following range of functions:

- Start bypass event task, and freely programmable LEDs on the Ωmega Drive-Line II.
- P, PI controller, 48-bit division via electronic transmission, integration, differentiation.
- Function blocks for interface module to USS® and 3964R® protocols.

NOTE

The Ωmega Drive-Line II firmware is used by several Ωmega Drive-Line II user libraries. This means that with an Ωmega Drive-Line II user library, it may be necessary to insert the requested firmware library SYSTEM2_DLII_20bd00 or above (See description of the user library).
To be able to use the basic functionality, you should always insert firmware SYSTEM2_DLII_20bd00 (or above) in an Ωmega Drive-Line II project in addition to user library SYSTEM1_DLII_20bd00 (or above).

4.5.2 The Ωmega Drive-Line II Board Functions

Firmware library SYSTEM2_DLII_20bd00 (or above) contains function blocks (FBs) for checking event signals for interrupts and board LEDs. Function block INTR_SET is used to initialize and start Ωmega Drive-Line II event tasks (bypass). Function block LED is used to allow the use of freely programmable LEDs.

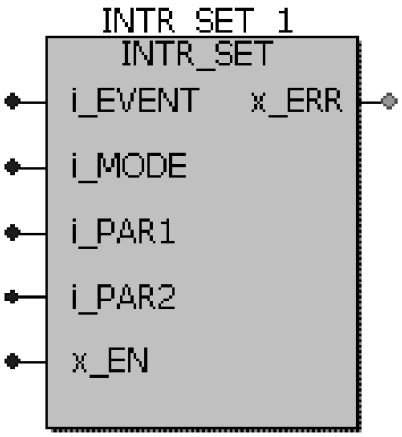
Two FBs for code runtime measurement are provide to optimize code runtimes within Ωmega Drive-Line II resources. The FBs are inserted via user library **SYSTEM1_DLII_20bd00** and above.

You must limit the code block within a task that is to be measured by placing FBs TIME_MEASURE_START and TIME_MEASURE_END. The system outputs the result of the measured code block's runtime at FB TIME_MEASURE_END as a time difference in µs (see also online description of FBs TIME_MEASURE_START and TIME_MEASURE_END).

Function block INTR_SET

Function block INTR_SET starts a bypass event task in a start-up task .

You must set the PROPROG wt II event task with the program to the event and to the Bypass attribute.



Initializing and enabling a bypass event task via function block INTR_SET.

Parameter	Input	Value range
i_EVENT	Interrupt hardware program number	8-bit signed
i_MODE	Reserve	16-bit signed
i_PAR1	CPU timer (1,2) value multiplier to base 50µs	16-bit signed
i_PAR2	Reserve	16-bit signed
x_EN	Block/enable the interrupt	1-bit

Parameter	Output	Value range
x_ERR	Error bit	1-bit

Description

Using FB INTR_SET, users can configure and activate various system-internal interrupt sources. You can then use these interrupts in the program to activate event tasks.

An event number must be connected at input i_EVENT. This number specifies the interrupt source of the event task. If the interrupt source in question is a CPU timer interrupt, you must additionally state a factor to the time base 50 µs at input i_PAR1.

You can use x_EN = FALSE to disable a previously activated interrupt.

List of event numbers for event tasks:

i_EVENT	Hardware event(s)	Interrupt level
0	CPU timer 1	Level 14
2	CPU timer 2	Level 13
4	BAPS process data	Level 13
5	Board timer A	Level 14
6	Board timer A	Level 13
8	BAPS process data	Level 14
11	SYNC signal option board	Level 14
12	SYNC signal network (CANsync)	Level 14

The event number at input i_Event must be identical with the setting of the event task within the resource. The Bypass attribute must be activated.

NOTE

A higher-priority interrupt interrupts a lower-priority one. For events 0 and 2, CPU timer 1 (or 2) interrupt, you must additionally state at input i_PAR1 the factor for time base 50 µs.
You cannot use events SYNC signal option board (11) and SYNC signal network (CANsync, 12) at the same time. You cannot use at the same time the BAPS process data low priority (4) and BAPS process data high priority (8).

Example 1:

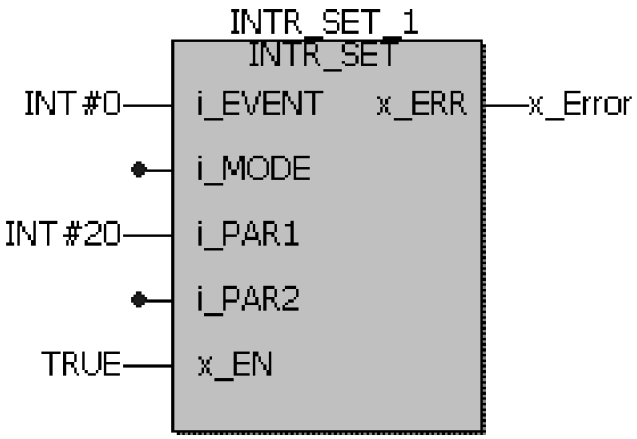
Starting a CPU timer interrupt with a time of 1 ms.

Implementation:

First of all, you set up the event task within the Ωmega Drive-Line II resource with event number 0, or CPU timer 1.

FB INTR_SET is implemented within a start-up task. Assignment of inputs/outputs looks like this:

Interrupt time = i_PAR1 • 50 µs



Function block INTR_SET: Starting a CPU timer 1 interrupt with an interrupt time of 1 ms.

Example 2:

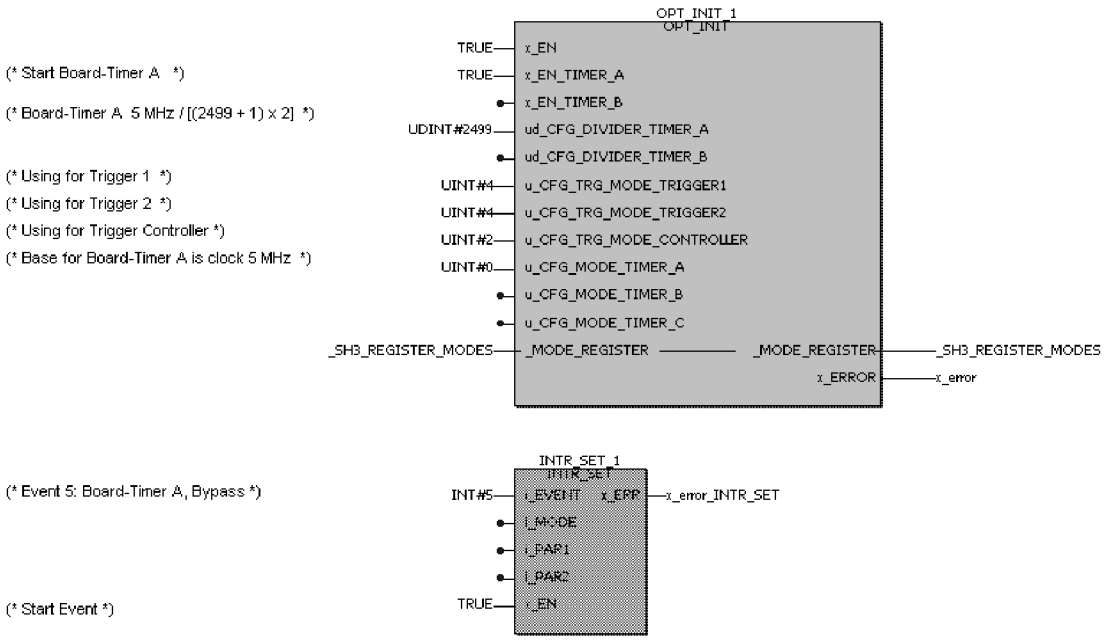
Starting a board timer A interrupt with a time of 1 ms.

By contrast with the CPU timer 1 (or 2), you can also use board timer A as a trigger signal for modules that need trigger signals, since it is only possible with board timer A to issue an interrupt as well as a trigger.

First of all, you set up the event task within the Ωmega Drive-Line II resource with event number 5 (or 6), or event board-timer A.

Within a start-up task, FB OPT-INIT is implemented first and then FB INTR_SET.

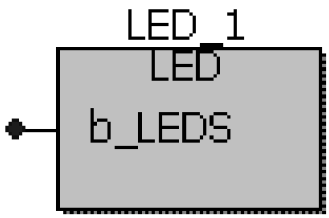
Interrupt time 1 ms	=	frequency divider for timer A set up at FB OPT-INIT	
1 ms	=	((2499 + 1) • 2) / 5 MHz	set up at FB OPT-INIT
Interrupt	=	event board timer A	state at FB INT_SET



Starting a board timer A event task with a time of 1 ms and simultaneous use as a trigger signal for modules that need trigger signals within a start-up task.

Function block LED

Using function block LED from firmware library **SYSTEM2_DLII_20bd00** (or above), you can program the Ωmega Drive-Line II LEDs on the board.



Parameter	Input	Value range
b_LEDS	LED set screen form	8-bit

Description

The left-hand LEDs on the Ωmega Drive-Line II board light up red, the right-hand ones light up green. Bits 0-3 of the 8-bit pattern are output to the four LEDs as follows:

Red LEDs	Green LEDs
⊗	⊗
Bit 0	Bit 1
⊗	⊗
Bit 2	Bit 3

4.5.3 Die Ωmega Drive-Line II Data Types

Frequently, Ωmega Drive-Line II user libraries need assembled data types in the input and output assignment of their function blocks (FBs). The data types are stored in user library

- **BM_TYPES_20bd00** (or above).

User library BM_TYPES_20bd00 (or above) makes available the data types to allow you to use the Ωmega Drive-Line II user function blocks and the firmware blocks. Ωmega Drive-Line II user function blocks and firmware blocks use these data types to map register structures of option boards or to pass on initialization value on a cross-task basis.

To be able to use standard libraries and technology components, you must insert BM_TYPES_20bd00 (or above) in your project. You cannot call the worksheet of BM_TYPES_20bd00. The data types are available in the variable dialog under Properties of assignment. The data types of BM_TYPES_20bd00 are marked **"_BM".**

In their descriptions, the standard libraries and technology components refer to the data types of BM_TYPES_20bd00 (or above). When implementing FBs from these libraries, you must declare variables of these data types and possibly connect them to an address, e.g. an option interface.

Example:

For initialization, option board IEI-02 needs settings in various registers. The complete register structure with its data width and its elements are stored in BM_TYPES_20bd00. With the IEI-02 in option slot 1, this yields the following variable declaration.

```
_IEI_write_register  AT  %MD3.1000000       : IEI_WRITE_BMSTRUCT;
```

The elements of variable `_IEI_write_register` now map the registers of the option board. You can symbolically program the registers via the elements of the variable (see technical description of option board IEI-02 for Ωmega Drive-Line II).

NOTE

In the variable dialog, the system makes available the data types for assignment in a selection dialog. In this connection, the data types of library BM_TYPES_20bd00 (or above) are indicated by the abbreviation "_BM" (_BMARRAY, _BMSTRUCT, etc.) The worksheet of the library is write-protected and you cannot view it.

4.5.4 The Standard Function Block Libraries

The standard libraries contain function blocks (FBs) with the basic functionality for local programming and configuration of the real-time response.

These function blocks are located in:

- **SYSTEM1_DLII_20bd00** (or above)

  BAPS communication

  Trigger signal interconnection of FB OPT-INIT

  Code runtime measurement

  3964R® protocol interface module

  USS® protocol interface module
- **SYSTEM2_DLII_20bd00** (or above)

  Ωmega Drive-Line II firmware:
- **UNIVERSAL_20bd00** or above (regardless of the hardware)

  Drive status and control via FB DRIVE1

  Extrapolators, ramp generators, position generators, Min-Max and limitation FBs

  Virtual leading axle FB TRAJECTORY_GEN1.

4.5.5 The Ωmega Drive-Line II Technology Components

You can extend the standard user libraries by adding complete drive functionality, i.e. the technology components. These are:

- The cam disk technology component: User library CAM_DLII_20bd00 (or above)
- Register controller technology component: User library REGISTER_DLII_20bd00 (or above)
- The winder technology component: User library WINDER_DLII_20bd00 (or above)

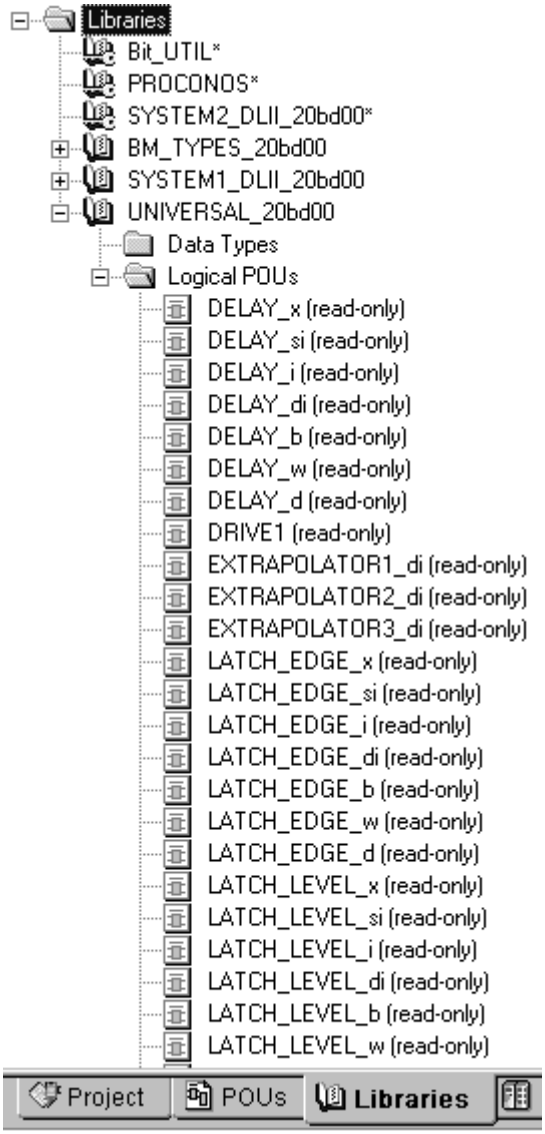
The technology components offer drive functionality that provide a large number of application solutions due to interconnection and multiple instantiation.

NOTE

For integration, all the user libraries of the technology components need data types BM_TYPES_20bd00 and above.

4.5.6 Inserting a User Library into a Project

You insert user libraries in PROPROG wt II in the project tree under libraries. If the library in question is firmware, you must set .fwl when choosing the file format.



Standard selection of user libraries, firmware and data types using the libraries filter

NOTE

A PROPROG wt II library is supplied in compressed form (as a ZWT file). You must unpack the ZWT file under PROPROG wt II. When unpacking the file, the system automatically stores the write-protected library in the specified PROPROG wt II library path (Tools > options menu) and displays an untitled project that you should close without saving it.

The system unpacks the firmware libraries to the firmware library directory of PROPROG wt II.

4.6 Ωmega Drive-Line II Option Interfaces, Interrupt Sources and Trigger Signals

4.6.1 The Interrupt Sources and Trigger Signals

In Ωmega Drive-Line II, modules, timers and option boards can be **interrupt sources** and trigger event-driven tasks. The necessary program interrupts are application-dependent. For the system to be able to process the input and output values of all the Ωmega Drive-Line II components in a real-time-capable way, all of the component values that are used in the set event task must be synchronized with it. This is carried out by a **trigger signal,** that the module either supplies or needs.

Modules that need trigger signals are:

- **Trigger 1:** IEI-02 incremental encoder board for latching the position actual values of incremental encoders.
- **Trigger 2:** MFM-01 digital/analog input and output board for starting analog/digital conversion.
- **Trigger Controller:** V-controller for synchronizing the control time slices.

Clock signals that modules make available:

- **Basic 5 MHz cycle:** Cycle for frequency dividers or board timers

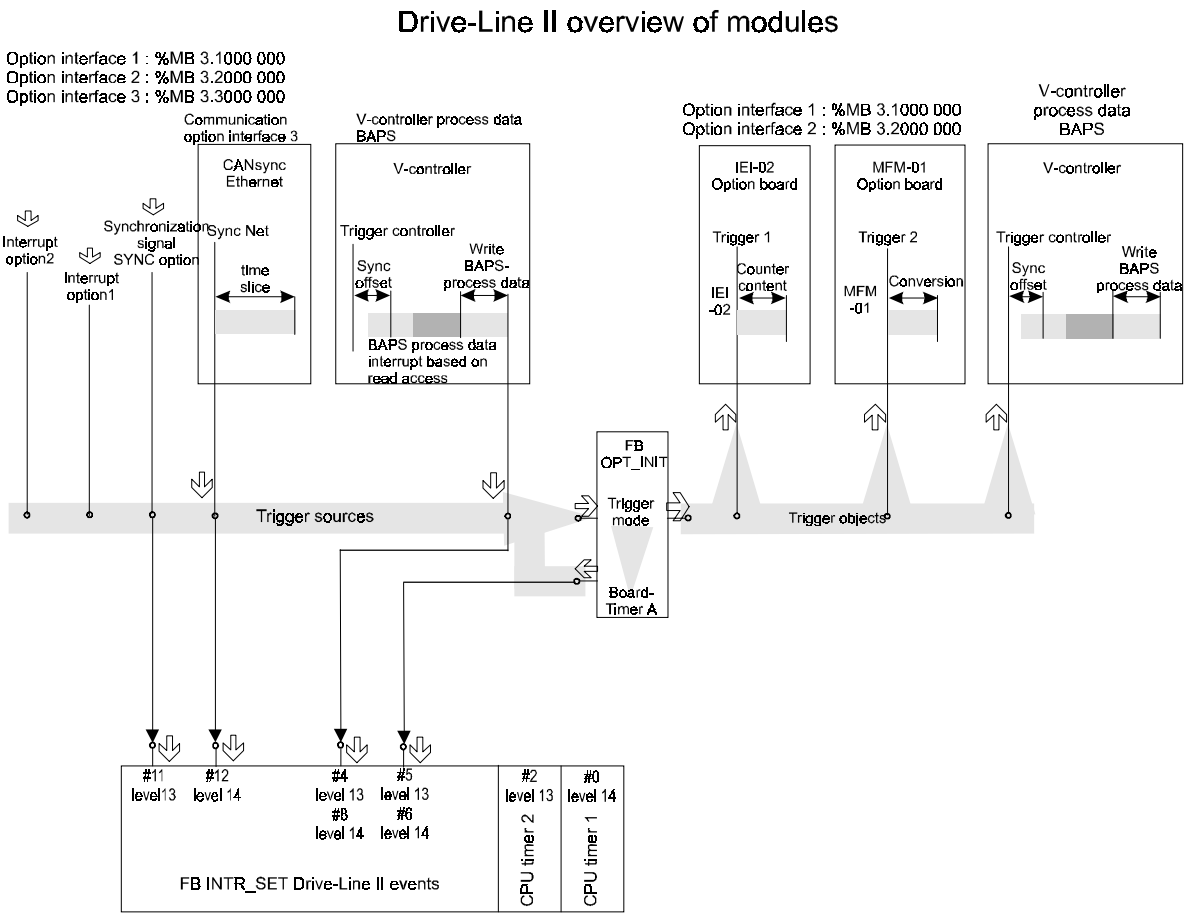
Interrupt signals that modules make available:

- **Interrupt Option 1:** Interrupt of an option board
- **Interrupt Option 2:** Interrupt of an option board
- **Interrupt Net:** Reserved
- **Interrupt timer A:** Timer A interrupt
- **Interrupt timer B:** Reserved

Synchronization signals that modules make available:

- **Sync Net:** SYNC signal of CANsync.
- **Sync Option:** SYNC signal of the option board.

The Ωmega Drive-Line II offers the option of switching trigger signals, SYNC signals and interrupt signals to modules that need trigger signals (function block OPT_INIT).



Overview of modules and trigger signals

4.6.2 Trigger Signal Interconnection and Timer Configuration via Function Block OPT_INIT

You must interconnect the modules that need trigger signals via function block (FB) OPT_INIT. The FB is integrated into the project via library SYSTEM1_DLII_20bd00 (or above). Modules that need trigger signals are:

- **Trigger 1:** IEI-02: for latching the position actual values on the option board.
- **Trigger 2:** MFM-01: for starting analog/digital conversion on the option board.
- **Trigger Controller:** V-controller: for synchronizing the control time slices on the V-controller.

If a trigger signal is not needed, the interconnection can stay open. You can use the trigger signals, interrupt signals or SYNC signals that other modules make available as the source of the trigger signals to be interconnected.

Necessary Trigger 1 for the IEI-02 option board in option slot 1 or 2:

Mode for Trigger 1	Synchronization/triggering using	Source
0	Reserved	Reserved
1	Reserved	Reserved
2	SYNC signal from the option board	Sync Option
3	CANsync synchronization signal	Sync Net
4	Board timer A via OPT_INIT	Interrupt timer A
5	Reserved	Interrupt timer B
6	Interrupt of an option board	Interrupt option 1

Necessary Trigger 2 for the MFM-01 option board in option slot 1 or 2:

Mode for Trigger 2	Synchronization/triggering using	Source
0	Reserved	Reserved
1	Reserved	Reserved
2	SYNC signal from the option board	Sync Option
3	CANsync synchronization signal	Sync Net
4	Board timer A via OPT_INIT	Interrupt timer A
5	Reserved	Interrupt timer B
6	Interrupt of an option board	Interrupt option 1

Necessary trigger controller (V-controller BAPS):

Mode for Trigger Controller	Synchronization using	Source
0	SYNC signal from the option board	Sync Option
1	CANsync synchronization signal	Sync Net
2	Board timer A via OPT_INIT	Interrupt timer A
3	Reserved	Interrupt timer B
4	Interrupt of an option board	Interrupt option 1
5	Interrupt of an option board	Interrupt option 2

The OPT-INIT Ωmega Drive-Line II function block is used for two tasks:

1. switching trigger signals for modules need the trigger signals to make available data from the modules on an interrupt call.
2. Specify the time interval of board timer A and start:
   time interval = [(divisor +1) • 2] / clock signal frequency

   Example: 1 ms = ((2499 + 1) • 2) / 5 MHz (basic cycle of 5 MHz as the clock signal)

You can set timers A and B to a fixed frequency and start them separately. Timer A can be used as an event for bypass event tasks.

NOTE

If you insert in PROPROG wt II a task of type event and set the Bypass attribute, you can start an interrupt from a start-up task via function block INTR_SET (firmware SYSTEM2_DLII_20bd00 or above). This is conditional on the event task in the IEC 61131-3 resource being set to the event.

4.6.3 Using Function Block OPT_INIT

You place function block (FB) OPT_INIT in the initialization tasks (cold boot, warm boot tasks) or in a cyclical task with the FB only being allowed to be run through once.

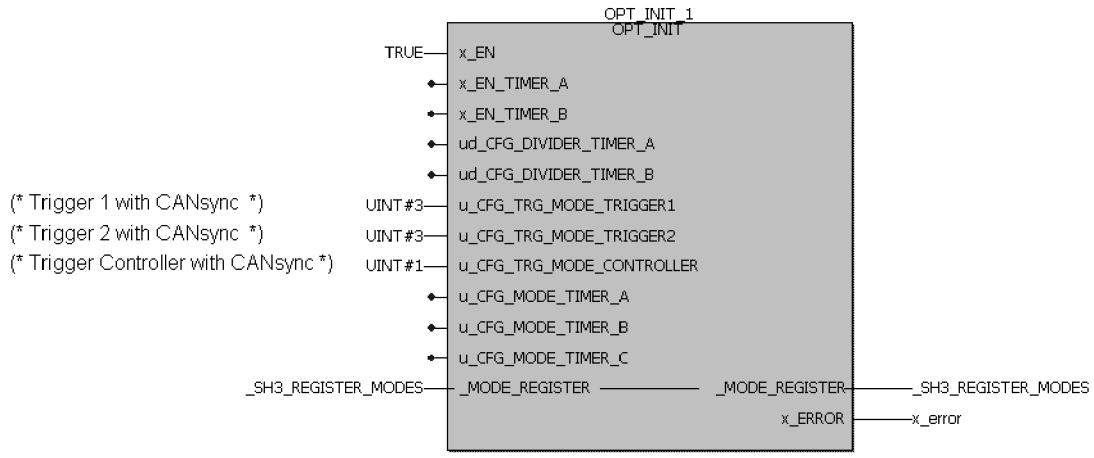
The call can be made for assignment of a needed trigger signal (e.g. IEI-02, MFM-01, V-controller) **after** initialization of the module.

When setting up a board timer (A, B) via frequency divider and synchronization to a clock signal (e.g. 5-MHz basic cycle), you must set up division as follows:

Time interval = [(divisor +1) • 2] / clock signal frequency

Example: FB OPT_INIT used in the initialization section of a CANsync slave-project for triggering the used option boards and the V-controller.

- You must integrate into the program FB OPT_INIT from library SYSTEM1_DLII_20bd00 (or above). You must switch the mode for modules that need the trigger signals to Sync Net (CANsync). Refer to the FB's online help for the mode.
- The system runs through FB OPT_INIT once and saves the configuration that was created at the inputs to the elements in the configuration structure (input/output variable _MODE_REGISTER).



Ωmega Drive-Line II used with CANsync slave event task and triggering of IEI-02, MFM-01 and the V-controller using the SYNC signal from the CANsync bus.

4.6.4 The Base Addresses of the Option Interfaces

Base addresses are assigned to option interfaces 1-3. The registers of the option board are offset addresses to this base address in accordance with their data type widths.

Ωmega Drive-Line II	access in PROPROG wt II: Start to end	assigned hardware address
Option interface 1:	**%MB 3.1000000** - %MB3.1262143 in option slot 1.	16#B4000000 - 16#B403FFFF
Option interface 2:	**%MB 3.2000000** - %MB3.2262143 in option slot 2.	16#B4040000 - 16#B407FFFF
Option interface 3: (communications boards)	**%MB 3.3000000** - %MB3.3262143 in option slot 2.	16#B4080000 - 16#B40AFFFF

NOTE

To allow you to access registers of the option boards in a PROPROG wt II project, data types are defined that map the register structure. The system uses these data types to declare variables that are assigned to the address of the option interface that is used. This makes it possible to access the register structure of the option board via the elements of the declared variables. For further explanations on the register structure and function, refer to the technical description of the respective option board.

4.6.5 Controller-Specific Mapping of the Hardware Areas

The following Mapping of Hardware addresses (base addresses) is implemented in the Ωmega Drive-Line II for PROPROG wt II for BAPS, CANsync and Ethernet communication:

Designation	**Access in PROPROG wt II Start - end**	**Assigned hardware address**
DPRAM BAPS	**%MB3.80000** - %MB3.88191	16#B4140000 - 16#B4141FFF
CANsync node 1	**%MB3.100000** - %MB3.132767	16#B4142000 - 16#B4149FFF
Ethernet configuration	**%MB3.180016** - %MB3.180031	16#B0000010 - 16#B000001F
CANsync node 2	**%MB3.200000** - %MB3.232767	16#B414A000 - 16#B4151FFF
Ethernet	**%MB3.300000** - %MB3.357343	16#B4152000 - 16#B415FFFF

Structures are assigned to the address ranges. If necessary, you should refer to the corresponding documentation for the libraries to get information about the use of the structures.

BAPS:

The V-controller interface BAPS to the communication RAM of the BAPS is implemented using:

- **BAPS_CTRL_BMSTRUCT** (defined in library BM_TYPES_20bd00 or above).

CANsync:

Communication for a CANsync slave is implemented using CANsync node 1. Communication for a CANsync master is implemented using CANsync node 2. The following structures are defined for this in library BM_TYPES_20bd00 (or above):

- **CANsync_INIT_BMSTRUCT**
- **CANsync_MA_CTRL_BMSTRUCT**
- **CANsync_SL_CTRL_BMSTRUCT**

CAN:

Using option interface 3, you can implement CAN communication: The following structures are defined for this in library BM_TYPES_20bd00 (or above):

- **CAN_INIT_BMSTRUCT**
- **CAN_CTRL_BMSTRUCT**

Ethernet:

For Ethernet **configuration** by users, e.g. Ethernet IP addresses and Ethernet IP mask, you can use the following structure in library BM_TYPES_20bd00 (or above):

- **ETHERNET_CONFIG_BMSTRUCT** (AT %MB3.180016)

The Ethernet configuration area is in the buffered NOVRAM area)

For Ethernet **diagnostics,** you can use the following structure from library BM_TYPES_20bd00 (or above):

- **ETHERNET_DIAGNOSE_BMSTRUCT** (AT %MB3.300000)

4.6.6 Sample Configurations

Depending on the option boards that you are using and on the synchronization requirements, you need different configurations for BAPS communication, the trigger signals and the interrupt sources.

You can use the following examples for the majority of applications:

Example A:

Implementing BAPS process data communication within a BAPS event task. BAPS process data communication is synchronized by the BAPS process data event from the V-controller.

NOTE

In this case, it is not possible to synchronize an option board via the trigger signals.

Example B:

Implementing high-precision BAPS process data communication within a timer event task (board timer A) and triggering MFM-01, IEI-02 and the V-controller via this timer.

Example C:

Implementing BAPS process data communication within a CANsync event task and triggering the V-controller, MFM-01 and IEI-02 via the synchronization signal of CANsync.

To transfer data via the 3964R® protocol or the USS® protocol, you need a timer event task.

Example D:

Implementing a timer event task for cyclical serial communication. The function blocks for cyclical communication are placed in a POU that is assigned to this task. If you need option board control signals, you can set them up.

NOTE

Code runtime measurement is recommended for optimizing the run times of process data.
You can carry out this measurement using FBs TIME_MEASURE_START and TIME_MEASURE_END from library SYSTEM1_DLII_20bd00 (or above).

4.6.7 Implementing a BAPS in a BAPS Event Task

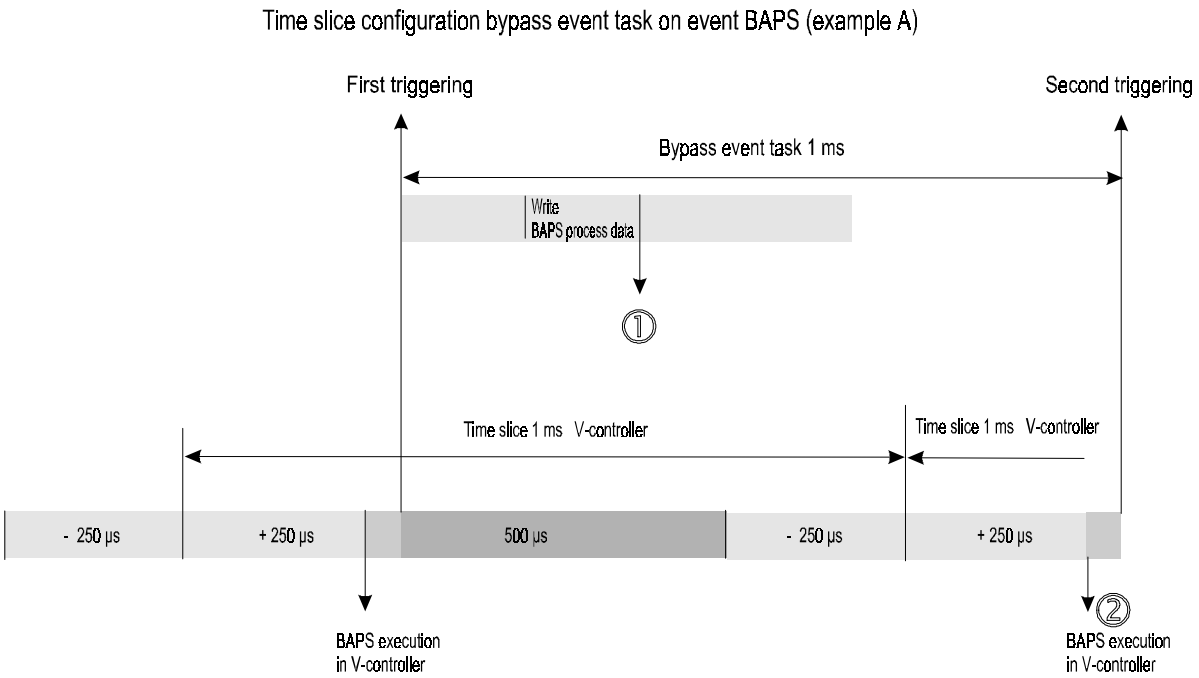
Example A:

The Ωmega Drive-Line II generates target values in a V-controller event task with BAPS process data communication to the V-controller. The event for the task comes from the V-controller. It is not possible to trigger option boards directly to the BAPS event.

BAPS process data communication is carried out directly in an event task using BAPS process data event 4 or 8 (V-controller). The BAPS process data communication event task is initialized and started via FB BAPS_INIT (see “BAPS_INIT” on page 82.). For this, FB BAPS_INIT is placed in a POU that is assigned to a cyclical task. In this connection, you must ensure that the FB is only enabled after five program cycles and that it is run through only once.

- The system initializes the BAPS to the desired time slice and parameter in a cyclical task via FB BAPS_INIT. Firmware block INTR_SET is a component of FB BAPS_INIT.
- For process data communication via FB BAPS_PD_COMM, you insert in the resource a bypass event task with event 4 or 8.

Time slice synchronization corresponds to the following drawing.



Bypass event task in the Ωmega Drive-Line II to BAPS event from the V-controller

NOTE

You can synchronize conversion of the MFM-01 values in „Change by writing mode“ within the BAPS event task.

In the V-controller's „Synchronous operation with synchronous reference value specification“ mode, at BAPS process data communication via parameter P258 (32-bit angle) you must mask (set FALSE) the lowest bits of the transfer value in dependence on the selected BAPS time slice:

BAPS process data communication in

- 500 µs event task: bit 0
- 1 ms event task: bits 1, 0
- 2 ms event task: bits 2, 1, 0
- 4 ms event task: bits 3, 2, 1, 0

4.6.8 Implementing a high-precision BAPS in a timer event task

Example B:

The Ωmega Drive-Line II generates the target values that are transferred in an event task with BAPS process data communication to the V-controller and needs synchronized triggered option boards, e.g. cam disk with register controller application.

The interrupt signal and the trigger signal for modules that need trigger signals must be identical. This implementation is only possible with board timer A. BAPS process data communication is carried out in event task board timer A, i.e. event board timer A is set to the time slice of the necessary BAPS process data communication.

- The trigger signal is interconnected and the event task on board timer A is started in a start-up task. **FB OPT-INIT** configures board timer A's time to the desired BAPS time slice, e.g. 1 ms.

Start the necessary event board timer A with frequency divider to 5-MHz basic cycle:

u_DIVIDER_TIMER_A	x_EN_TIMER_A	ud_CFG_MODE_TIMER_A
2499	TRUE	0
1 ms = [(2499 +1) • 2] / 5 MHz	Timer A started	5 MHz basic cycle

Necessary Trigger 1 for the EI-02 option board in option slot 1 or 2:

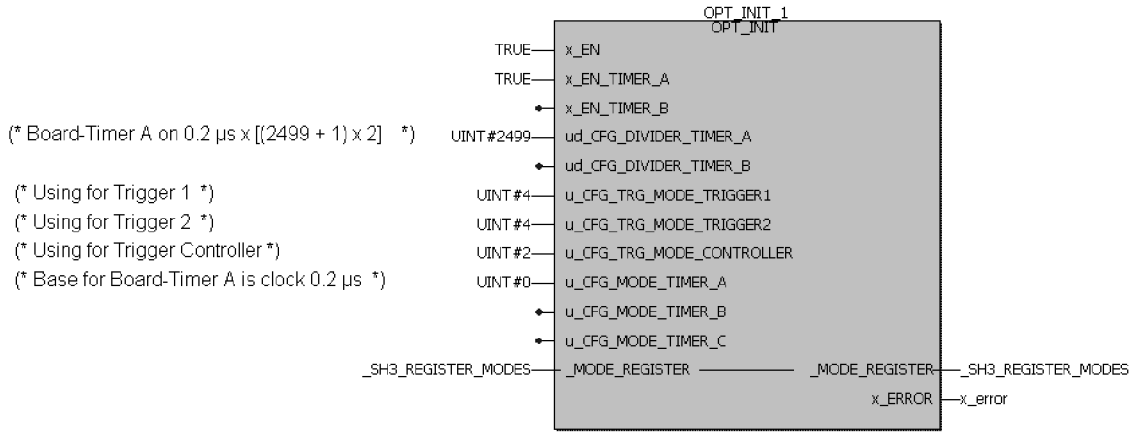
Mode for Trigger 1	Synchronization/triggering using	Source
4	Board timer A via OPT_INIT	Interrupt timer A

Necessary Trigger 2 for the MFM-01 option board in option slot 1 or 2:

Mode for Trigger 2	Synchronization/triggering using	Source
4	Board timer A via OPT_INIT	Interrupt timer A

Necessary Trigger Controller (V-controller BAPS):

Mode for Trigger Controller	Synchronization using	Source
2	Board timer A via OPT_INIT	Interrupt timer A

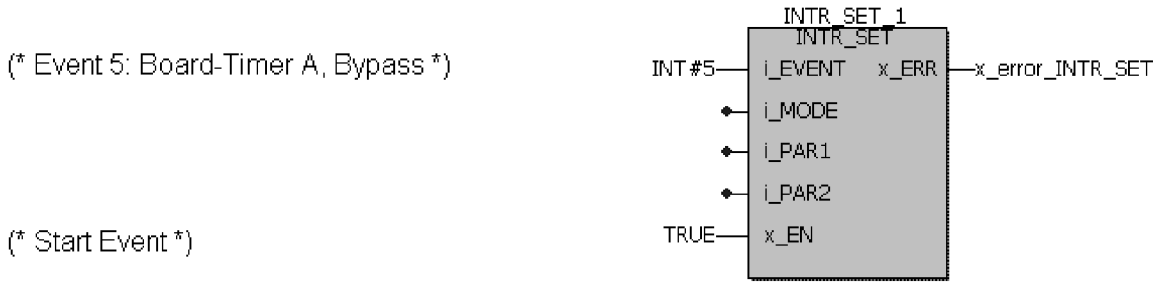


FB OPT_INIT for starting board timer A and trigger signal interconnection in a start-up task.

- In accordance with FB OPT-INIT, the following are inserted in **FB INR_SET:**

i_EVENT	Hardware event(s)	Interrupt level
5	Board timer A	Level 14

i_MODE, i_PAR1, i_PAR2 are not interconnected.



FB INTR_SET for starting a bypass event task on board timer A in a start-up task.

- You must insert in the Ωmega Drive-Line II resource an **event task** to event **board timer A.** First you insert in the task the evaluation of the (IEI-02/MFM-01) option boards and the BAPS process data communication. After this the system carries out reference value generation.
- In the V-controller, you match reference value synchronization via WinBASS on the service user interface: SyncSlotzeit (P167) is set to the **board timer A time** and parameter SyncOffset (P168) is set in accordance with the explanation in the drawing.

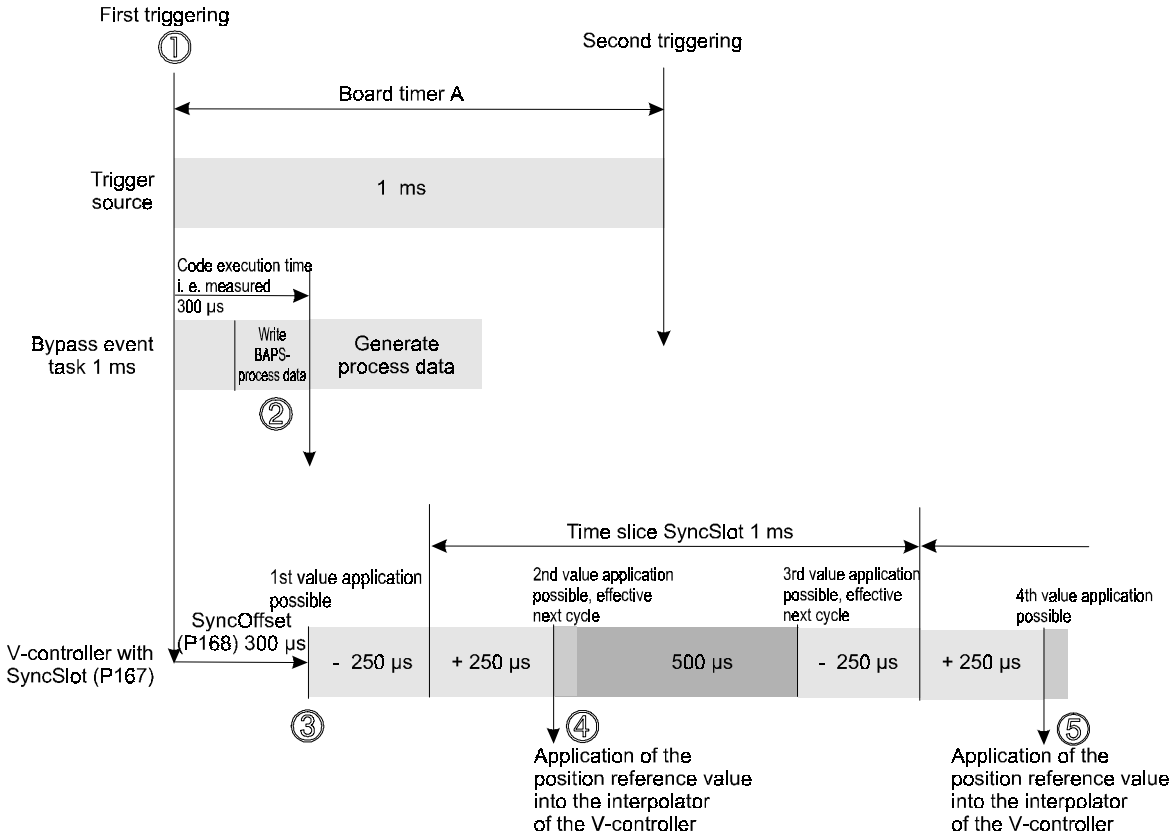
NOTE

In the V-controller's „Synchronous operation mit synchronous reference value specification“ mode, at BAPS process data communication via parameter P258 (32-bit angle) you must mask (set FALSE) the lowest bits of the transfer value in dependence on the selected BAPS time slice:

BAPS process data communication in

- 500 µs event task: bit 0
- 1 ms event task: bits 1, 0
- 2 ms event task: bits 2, 1, 0
- 4 ms event task: bits 3, 2, 1, 0

Time slice configuration high precision BAPS (example B)



Process data communication to the V-controller in board timer A event task and triggering modules to board timer A event.

The SyncOffset is for optimizing the run time until the position reference value is applied in the V-controller. If it is adequate for the value to be applied in the next cycle ⑤, you must set the SyncOffset to

ZERO. Otherwise, you must measure the code runtime, including BAPS process data communication, and parameterize it as the SyncOffset in the V-controller. Then, the system applies the value in point ④.

In this connection, you should note that the values that are not transferred via the BAPS until the second value application do not become effective in point ④, but rather in point ⑤.

In the V-controller, all the parameters, except for the position reference values, become effective directly after value application.

4.6.9 Implementing a BAPS within the CANsync synchronous bus system

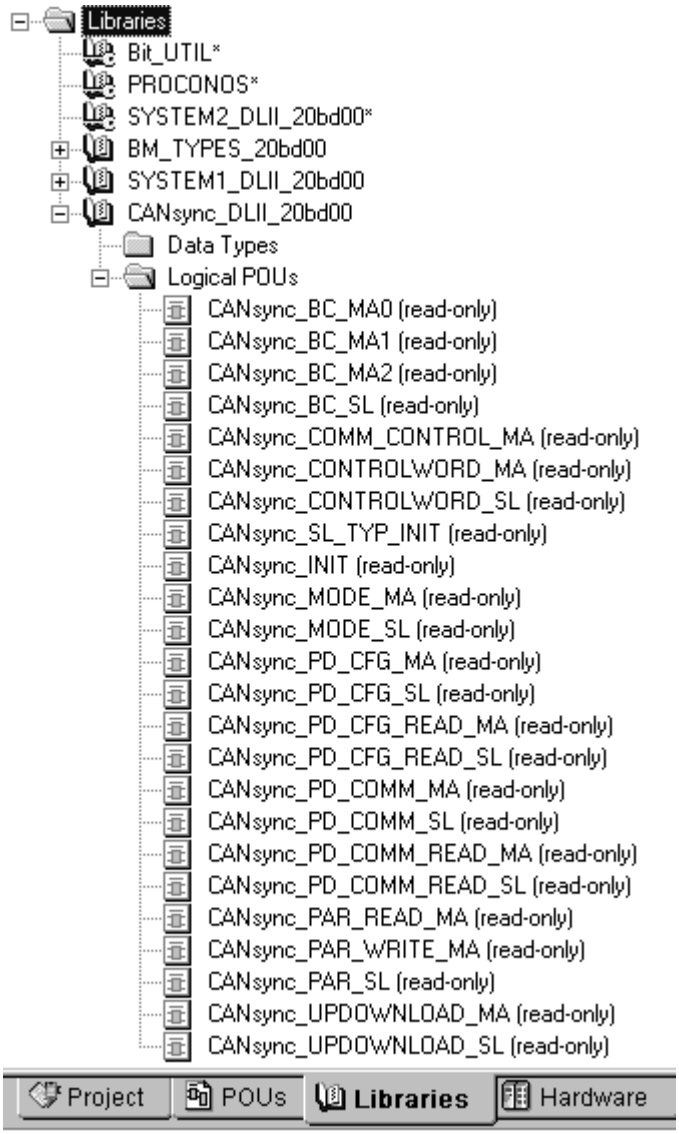
Example C:

The Ωmega Drive-Line II is part of the CANsync synchronous bus system (master or slave). In this case, process data communication via CANsync and BAPS process data communication to the V-controller must be carried out in one task.

The event for the task is the synchronization signal from the CANsync bus. The Ωmega Drive-Line II resource must have a bypass event task with event CANsync (12). You insert BAPS process data communication in this bypass event task.

The V-controller and option board modules that need trigger signals are also triggered using the synchronization signal from the CANsync bus.

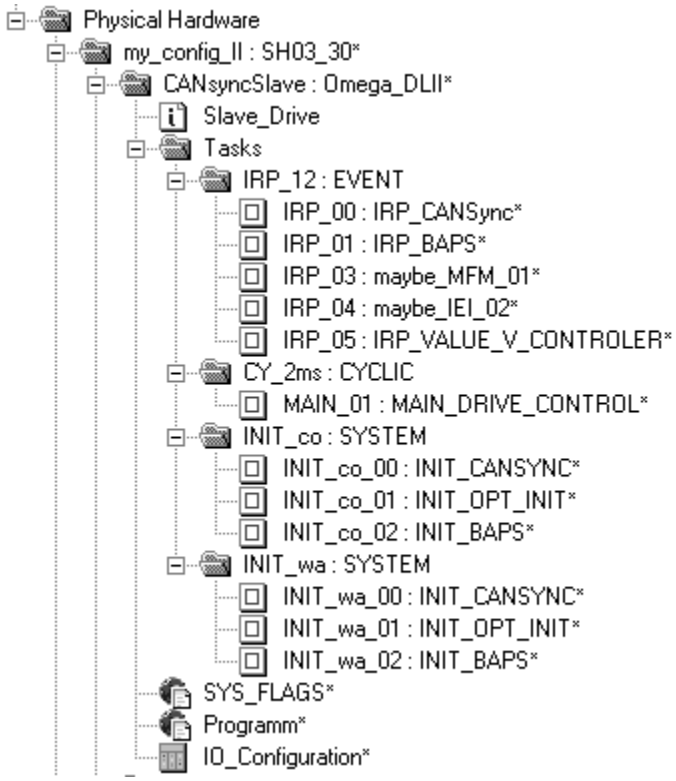
- The bypass event task is inserted in the Ωmega Drive-Line II resource with event **CANsync (12).**
- You use WinBASS to parameterize the V-controller's operating mode to **synchronous operation** with **synchronous reference value setting.** You must set parameter **SyncSlot (P167)** to the CANsync time slice. The **SyncOffset (P168)** is zero.
- To implement BAPS process data communication within the CANsync event task and CANsync process data communication, you must insert the following Ωmega Drive-Line II libraries:
  - BM_TYPES_20bd00 or above: data types for CANsync and BAPS.
  - SYSTEM1_DLII_20bd00 or above: BAPS-FBs and FB OPT_INIT.
  - SYSTEM2_DLII_20bd00 or above: FB INTR_SET for the CANsync library.
  - CANsync_DLII_20bd00 or above: initialize CANsync event task and bus link.



Libraries for implementing a BAPS time slice synchronously with CANsync process data communication

You implement CANsync process data and requirements data communication using the FBs of the libraries. FB CANsync_INIT starts the CANsync bypass event task. After this, FB BAPS_INIT initializes BAPS process data communication.

For information on the deployment of the function blocks for CANsync and BAPS implementation, refer to the respective module description. Implementation should yield the following task and program structure.



Basic structure of a resource with CANsync bus link and BAPS process data communication.

You insert into this basic structure the program POUs of reference value setting to the BAPS. Initialization of the CANsync interface module and the BAPS is completed by interconnecting via FB OPT_INIT the trigger signals for modules that need trigger signals.

All the modules are triggered using the synchronization signal of CANsync.

Necessary Trigger 1 for the EI-02 option board in option slot 1 or 2

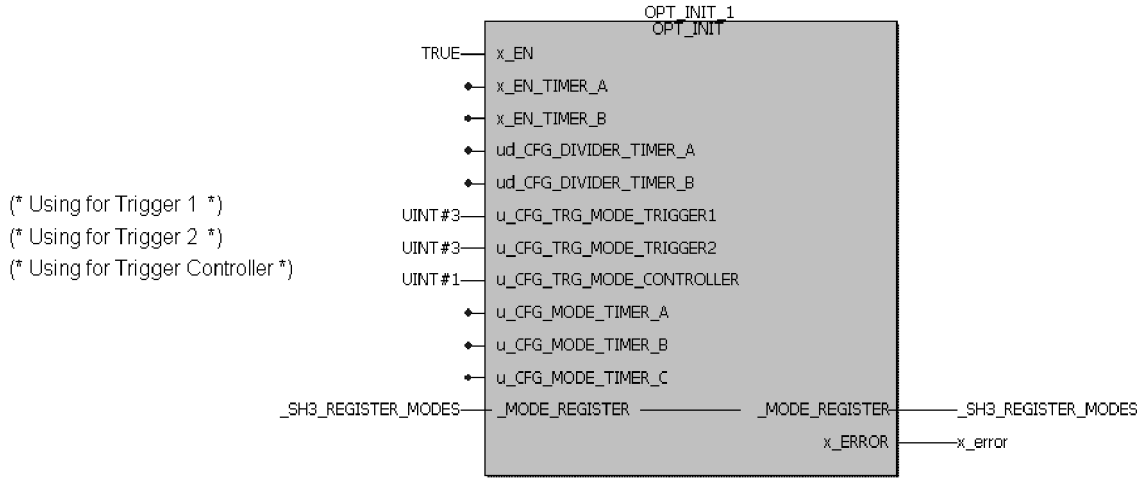
Mode for Trigger 1	Synchronization/triggering using	Source
3	CANsync synchronization signal	Sync Net

Necessary Trigger 2 for the MFM-01 option board in option slot 1 or 2

Mode for Trigger 2	Synchronization/triggering using	Source
3	CANsync synchronization signal	Sync Net

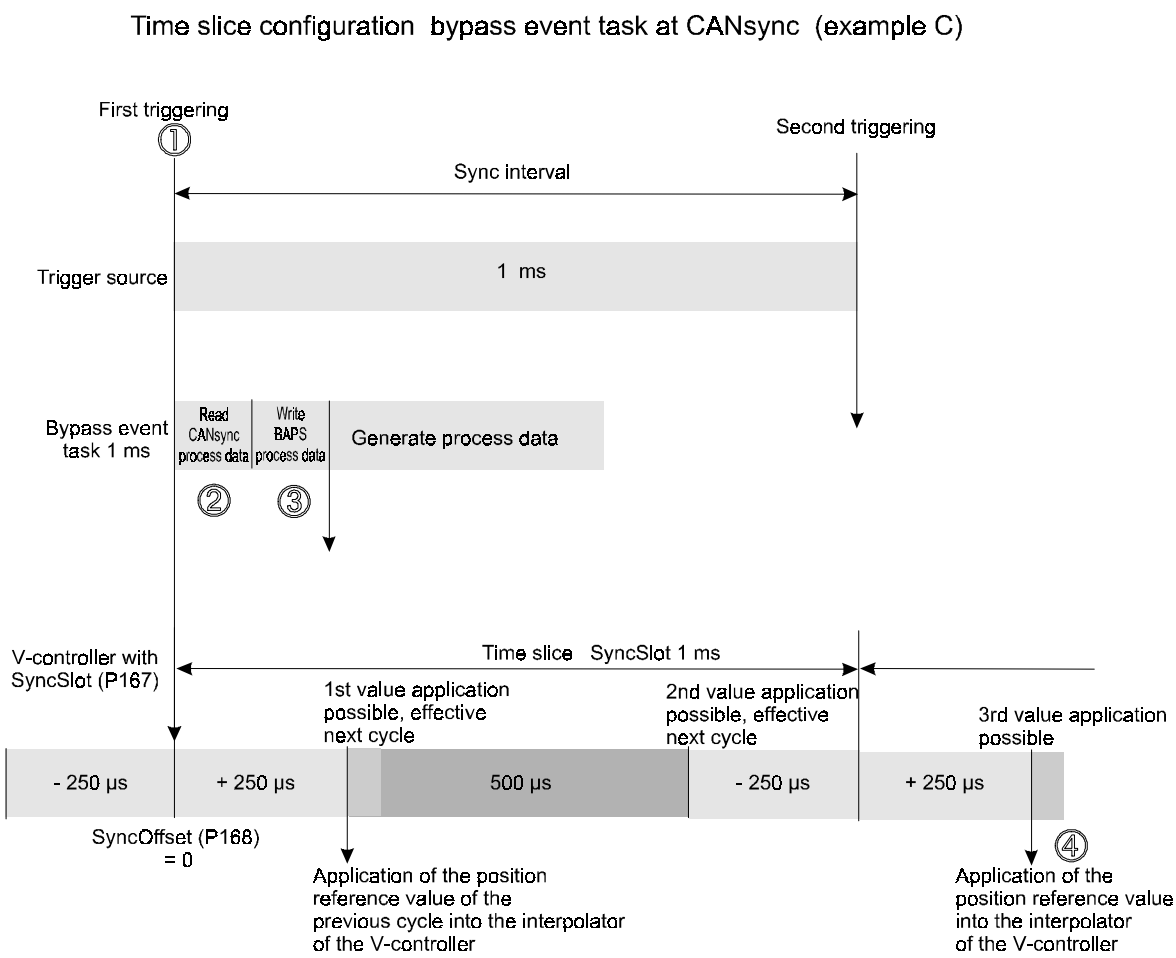
Necessary Trigger Controller (V-controller BAPS)

Mode for Trigger Controller	Synchronization using	Source
1	CANsync synchronization signal	Sync Net



FB OPT_INIT with trigger signal interconnection to the SYNC signal of the CANsync synchronous bus.

This implementation synchronizes the time slice response as shown in the following drawing. It guarantees that the target value is always applied via BAPS to the V-controller after CANsync process data communication.



CANsync standard time slice configuration.

Since the position reference value does not become effective until the next cycle when SyncOffset is equal to zero, it is also possible to execute BAPS process data communication at the end of the event task.

NOTE

In the V-controller's „Synchronous operation mit synchronous reference value specification“ mode, the system applies the position reference value within a 500 µs time slice. The system applies immediately other parameters, e.g. the torque limit or the reference speed value.
In the V-controller's „Synchronous operation mit synchronous reference value specification“ mode, at BAPS process data communication via parameter P258 (32-bit angle) you must mask (set FALSE) the lowest bits of the transfer value in dependence on the selected BAPS time slice:

BAPS process data communication in

- 500 µs event task: bit 0
- 1 ms event task: bits 1, 0
- 2 ms event task: bits 2, 1, 0
- 4 ms event task: bits 3, 2, 1, 0

4.6.10 Implementing a timer event task for cyclical serial communication.

Example D:

You want to set up a bypass event task to a timer in the Ωmega Drive-Line II resource. The following events for timers exist.

i_EVENT	Hardware event(s)	Interrupt level	Trigger source FB OPT-INIT
0	CPU timer 1	Level 14	No
2	CPU timer 2	Level 13	No
5	Board timer A	Level 14	Yes
6	Board timer A	Level 13	Yes

When choosing the timer event, observe whether it is to be used to trigger a module (IEI-02, MFM-01 or V-controller). It is only possible to trigger modules using timer A and not using the internal CPU timer 1 (or 2).

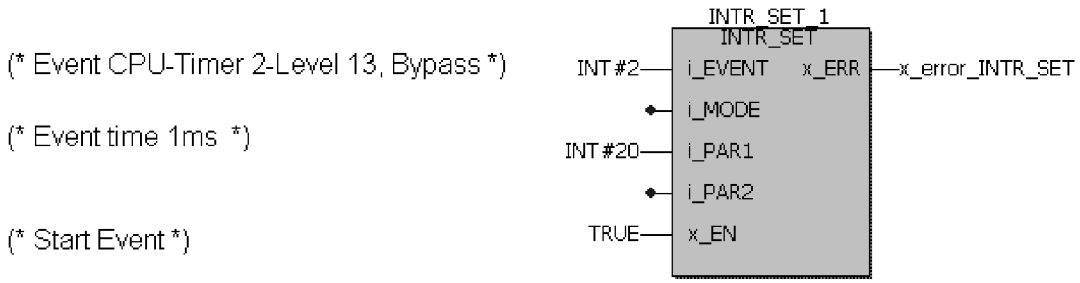
As a result:

- Start bypass event task via FB INTR_SET in a start-up task. If the event is a **CPU timer 1** (or 2), you parameterize the time at Input i_PAR1. It is not possible to trigger modules that need trigger signals; as a result, FB OPT_INIT is not used. The event task runs completely asynchronously to value conversion of the modules. Sample application: time slice for the 3964R® protocol.
- Start bypass event task via FB INTR_SET in a start-up task. If the event is a **board timer A,** input i_PAR1 on FB INTR_SET is not interconnected. You set up the timer time via FB OPT_INIT. It is possible to trigger modules that need trigger signals; for this reason, these modules are interconnected with FB OPT_INIT on timer A. Sample application: high-precision BAPS event task.

In the simplest application, you must insert a fixed time slice as follows:

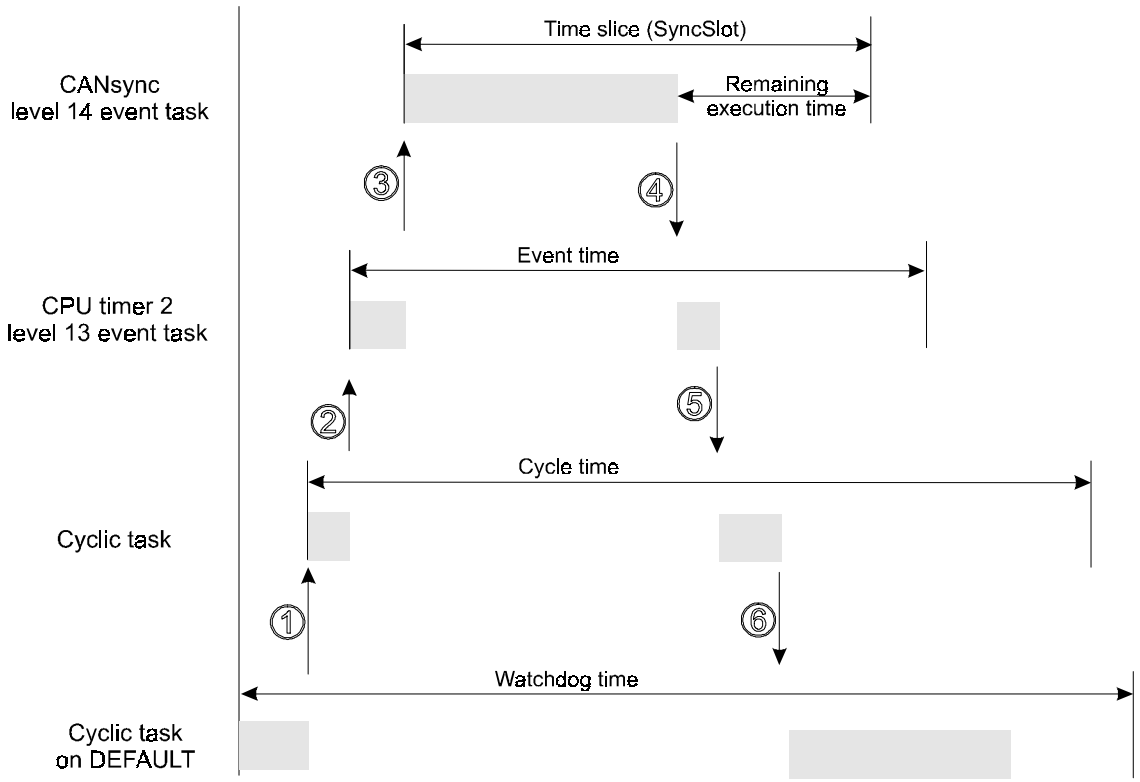
- First of all, you set up a bypass event task in the Ωmega Drive-Line II resource with event CPU timer 1 (or 2) with level 14 or 13. The program POUs are implemented in this event task.
- Since this is a bypass event task, the system starts the event using FB INTR_SET in a start-up task.

Interrupt time = i_PAR1 • 50 µs



Function block INTR_SET in a start-up task: Starts a CPU timer 2 level 13 interrupt with interrupt time of 1 ms.

Code execution time timer level 13 event task with lLevel 14 CANsync event task (example D)



Code run of an Ωmega Drive-Line II resource with event tasks and cyclical tasks.

NOTE

Since a higher-priority event task interrupts a lower-priority one, you should always use timer event tasks of level 13 if they are to be started in parallel with synchronous bus systems. You can use FB INTR_SET with x_EN = FALSE to block a previously started event.

5 ETHERNET (OPTIONAL)

5.1 General

NOTE

The conditions for setting the IP address and the IP mask are as follows:
- program PROPROG wt II
- library BM_TYPES_20bd00 (or above),

as well as knowledge of the hardware of the Ωmega Drive-Line II (see chapter 2) and knowledge of Ethernet (IP address, IP mask and routers)

From the number of the software version (SV) on the second identification plate you can recognize whether you have a device with or without Ethernet interface.
E.g. SV: 0003-I002-0000
The second part of the number (I002) determines the function of the PLC. I0xx means that the device is without Ethernet interface respectively I1xx means that the device is with Ethernet interface.

When communicating with devices via Ethernet using the TCP/IP protocol, you must set a different IP address for each device. See “Communication via Ethernet (optional)” on page 30.

For the Ωmega Drive-Line II, users must set the IP address and the IP mask only once. The factory-set IP address and the IP mask are stored in the Ωmega Drive-Line II's NOVRAM (non-volatile memory) (See “Preset, Variable IP Address (Delivery Status)” on page75.).

Under PROPROG wt II, structure `ETHERNET_CONFIG_BMSTRUCT` is available for setting the Ωmega Drive-Line II's IP address and IP mask.

To set the IP address and the IP mask, users must create in a PROPROG wt II project a global variable of data type

`ETHERNET_CONFIG_BMSTRUCT`

and assign it to address

`%MB3.180016`

Example:

```
_Ethernet_Address AT     %MB3.180016 : ETHERNET_CONFIG_BMSTRUCT
```

Where:

`_Ethernet_Address`	is the variable name with the data type short designation "_" for STRUCT
`%MB3.180016`	is the address
`ETHERNET_CONFIG_BMSTRUCT`	is the data type of the variable

Structure `ETHERNET_CONFIG_BMSTRUCT` is defined as follows:

```
ETHERNET_CONFIG_BMSTRUCT      :     Struct
      d_IP_CONFIG             :     DWORD;
      a_IP_ADDRESS            :     USINT_4_BMARRAY
      a_IP_MASK               :     USINT_4_BMARRAY
      d_ROUTER                :     DWORD;
END_STRUCT;
```

Example of accessing an element of the structure:

```
Ethernet_Address.d_IP_CONFIG
```

Where:

`Ethernet_Address`	is the variable name
`d_IP_CONFIG`	is the element of the structure with the data type short designation "d" for DWORD

In the following description, the variable name is replaced by an asterisk (*).

Structure elements `*.a_IP_ADDRESS` and `*.a_IP_MASK` are each of data type `USINT_4_BMARRAY`. Data type `USINT_4_BMARRAY` is a field containing four entries of data type Unsigned Short Integer:

```
USINT_4_BMARRAY               ARRAY[0..3]  OF     USINT;
```

You enter an IP address in the variables `*` in entry `a_IP_ADDRESS` at places 0 to 3.

Example: Entering the IP address 192.168.75.190 (in structured text (ST))

```
*.a_IP_ADDRESS[0]       :=    USINT#192;
*.a_IP_ADDRESS[1]       :=    USINT#168;
*.a_IP_ADDRESS[2]       :=    USINT#75;
*.a_IP_ADDRESS[3]       :=    USINT#190;
```

You enter an IP mask in the variables `*` in entry `*.a_IP_MASK` at places 0 to 3.

Example: Entering the IP mask 255.255.255.0 (in structured text (ST))

```
*.a_IP_MASK[0]          :=    USINT#255;
*.a_IP_MASK[1]          :=    USINT#255;
*.a_IP_MASK[2]          :=    USINT#255;
*.a_IP_MASK[3]          :=    USINT#0;
```

The Ωmega Drive-Line II offers different options for choosing IP addresses and using the DIP switches of the Ωmega Drive-Line II for automatic IP addresses in a network (containing several Ωmega Drive-Line IIs). You make the selection via entry

```
*.d_IP_CONFIG
```

of structure

`ETHERNET_CONFIG_BMSTRUCT.`

5.2 Setting the IP Address and the IP Mask

Procedure:

– Enter IP address
```
*.a_IP_ADRESS[0] .. *.a_IP_ADRESS[3]
```
– Enter IP mask
```
*.a_IP_MASK[0] .. *.a_IP_MASK[3]
```
– Save the IP address and the IP mask by writing to
```
*.d_IP_CONFIG
```
– Setting the response of routers on the network
```
*.d_ROUTER
```

The following options, which are described below, are available for setting the IP address:

- Self-selected, fixed IP address (that is independent of the DIP switch setting)
- Self-selected, variable IP address (that is dependent on the DIP switch setting)
- Preset, variable IP address (that is dependent on the DIP switch setting)

5.2.1 Self-Selected, Fixed IP Address

`*.d_IP_CONFIG` same as `DWORD#16#12345678`

The IP address is **independent** of the DIP switch setting.

The IP address from

```
*.a_IP_ADDRESS[0] .. *.a_IP_ADDRESS[3]
```

and the IP mask from

```
*.a_IP_MASK[0] .. *.a_IP_MASK[3]
```

are stored in the Ωmega Drive-Line II.

Example (you must keep to the same sequence):

– Enter the IP address 192.168.75.190 (in structured text (ST))
```
*.a_IP_ADDRESS[0] :=     USINT#192;
*.a_IP_ADDRESS[1] :=     USINT#168;
*.a_IP_ADDRESS[2] :=     USINT#75;
*.a_IP_ADDRESS[3] :=     USINT#190;
```
– Enter the IP mask 255.255.255.0 (in structured text (ST))
```
*.a_IP_MASK[0]    :=     USINT#255;
*.a_IP_MASK[1]    :=     USINT#255;
*.a_IP_MASK[2]    :=     USINT#255;
*.a_IP_MASK[3]    :=     USINT#0;
```
– Save the IP address and the IP mask in the Ωmega Drive-Line II
```
*.d_IP_CONFIG     :=     DWORD#16#12345678;
```

Regardless of the DIP switch setting, the Ωmega Drive-Line II then has the IP address 192.168.75.190.

NOTE

IP addresses xxx.yyy.zzz.0 and xxx.yyy.zzz.255 are not allowed for devices.

5.2.2 Self-Selected, Variable IP Address

*.d_IP_CONFIG same as DWORD#16#12345600;

The IP address is **dependent on** the DIP switch setting.

The IP address from

*.a_IP_ADDRESS[0],

*.a_IP_ADDRESS[1],

*.a_IP_ADDRESS[2] and

*.a_IP_ADDRESS[3] + ”number of DIP switch”

and the IP mask from

*.a_IP_MASK[0] .. *.a_IP_MASK[3]

are stored in the Ωmega Drive-Line II.

Example (you must keep to the same sequence):

– Enter the IP address 192.168.75.190 (in structured text (ST))

```
*.a_IP_ADDRESS[0] :=    USINT#192;
*.a_IP_ADDRESS[1] :=    USINT#168;
*.a_IP_ADDRESS[2] :=    USINT#75;
*.a_IP_ADDRESS[3] :=    USINT#190;
```

– Enter the IP mask 255.255.255.0 (in structured text (ST))

```
*.a_IP_MASK[0]    :=    USINT#255;
*.a_IP_MASK[1]    :=    USINT#255;
*.a_IP_MASK[2]    :=    USINT#255;
*.a_IP_MASK[3]    :=    USINT#0;
```

– Save the IP address and the IP mask in the Ωmega Drive-Line II

```
*.d_IP_CONFIG     :=    DWORD#16#12345600;
```

With this set IP base address of 192.168.75.190, the following applies:

The Ωmega Drive-Line with DIP switch setting **"0"** has the IP address 192.168.75.19**0**,

The Ωmega Drive-Line with DIP switch setting **"1"** has the IP address 192.168.75.19**1**,

...,

The Ωmega Drive-Line with DIP switch setting **"30"** has the IP address 192.168.75.2**20**,

The Ωmega Drive-Line with DIP switch setting **"31"** has the IP address 192.168.75.2**21**,

NOTE

IP addresses xxx.yyy.zzz.0 and xxx.yyy.zzz.255 are not allowed for devices.

DIP switch settings can be 0 ... 31 (dip switch 1 – dip switch 5). See “Setting the Slave Number” on page 20.

In this connection, you should note:
IP address xxx.yyy.zzz.224 + 31 corresponds to IP address xxx.yyy.zzz.255 and is not allowed for a device.

NOTE

The Ωmega Drive-Line II takes the DIP switch setting after activation or after a reset. Later changes do not become effective until you switch the device off and on again or after a reset.

Important:
The DIP switch setting is also used for CANsync addressing!

5.2.3 Preset, Variable IP Address (Delivery Status)

The IP address is **-dependent** on the DIP switch setting.

The system uses

IP address 192.168.1.1 + “number of DIP switch” that are preset in the Ωmega Drive-Line II

and the preset

IP mask 255.255.255.0.

With this set IP base address of 192.168.1.1, the following applies:

The Ωmega Drive-Line with DIP switch setting **"0"** has the IP address 192.168.1**1**,

The Ωmega Drive-Line with DIP switch setting **"2"** has the IP address 192.168.1**2**,

...,

The Ωmega Drive-Line with DIP switch setting **"30"** has the IP address 192.168.1**31**,

The Ωmega Drive-Line with DIP switch setting **"31"** has the IP address 192.168.1**32**,

NOTE

DIP switch settings can be 0 ... 31 (dip switch 1 - dip switch 5). See “Setting the Slave Number” on page 20.

NOTE

The Ωmega Drive-Line II takes the DIP switch setting after activation or after a reset. Later changes do not become effective until you switch the device off and on again or after a reset.

Important:
The DIP switch setting is also used for CANsync addressing!

5.3 Setting the Response with Routers on the Network

The Ωmega Drive-Line II can communicate on the Ethernet with devices whose IP addresses are outside the subnet. For this communication, you can set whether a router is to be used that the Ωmega Drive-Line II finds in accordance with the Router Discovery Procedure (RFC 1256).

With the following setting (the delivery status)

*.d_ROUTER not same as DWORD#16#4E6F5F52,

the Ωmega Drive-Line II uses routers after detecting them on the network.

With the following setting

*.d_ROUTER same as DWORD#16#4E6F5F52,

the Ωmega Drive-Line II **does not use any** routers.

5.4 Communication Between Ωmega Drive-Line II and PROPROG wt II via Ethernet

After setting and saving the IP address and the IP mask communication is possible between the Ωmega Drive-Line II and PROPROG wt II. For this, you must set in PROPROG wt II under Resource/Settings communication to DLL and the IP address of the Ωmega Drive-Line II. See “Communication via Ethernet (optional)” on page 30.

6 BAPS BAUMÜLLER DRIVES PARALLEL INTERFACE

6.1 BAPS General

BAPS is an internal communications interface for exchanging data between the V-controller module and the Ωmega Drive-Line II module.

When carrying out communication, a differentiation is made between process data communication and requirements data communication.

Process data communication comprises reading and writing time-critical referenced and actual values and the status and control word in a definable time raster.

Requirements data communication comprises reading and writing non-time-critical parameters of the V-controller.

Ωmega Drive-Line II and V-controller

Process data:

The Ωmega Drive-Line II takes the V-controller's actual values and the status word in a definable time raster from BAPS and transfers the reference values and the control word via BAPS to the V-controller.

At transfer of the actual values and the status word, the system triggers the "BAPS-Prozeßdaten" (BAPS process data event) hardware event in the Ωmega Drive-Line II. This event can trigger an event task in the Ωmega Drive-Line II, in which BAPS process data communication can be carried out.

The system sets the instant or the timing code for communication at initialization of the process data communication in the user program using FB BAPS_INIT.

Requirements data:

The Ωmega Drive-Line II transfers a read parameter or write parameter job to BAPS. The V-controller reads the job from the BAPS, processes the appropriate parameters in the V-controller (see the respective V-controller description) and returns the result to the BAPS. The Ωmega Drive-Line II then reads the result of communication from the BAPS.

Programming BAPS communication on the Omega Drive- Line II

You program the Ωmega Drive-Line II using the PROPROG wt II programming system (see the PROPROG wt II manual).

The Baumüller user libraries SYSTEM1_DLII_20bd00 and SYSTEM2_DLII20bd00 (or above) are available for programming BAPS communication.

In library SYSTEM1_DLII_20bd00 (or above), function blocks are available for initializing process data communication, for process data communication and for requirements data communication.

You need the following FBs for process data communication:

BAPS_INIT	initialization of process data communication.
BAPS_PD_COMM8	process data communication (maximum of 8 reference values and 8 actual values)

or

BAPS_PD_COMM24	process data communication (maximum of 2 reference values and 4 actual values)

or

BAPS_PD_COMM2	process data communication (maximum of 2 reference values and 2 actual values)

For monitoring process data communication (optional):

BAPS_PD_CONTROL	monitoring the call of FB BAPS_PD_COMMxx (and with this, indirect monitoring of the call of the event task)

You need the following FBs for requirements data communication:

BAPS_PAR_READ	requirements data communication (read parameter job)
BAPS_PAR_WRITE	requirements data communication (write parameter job)

and

BAPS_SD_CONTROL	structure and monitoring requirements data communication.

Library SYSTEM2_DLII_20bd00 (or above) contains amongst others, the following function block:

INTR_SET	is used by FB BAPS_INIT; FB for linking and activating a hardware signal with the BAPS process data event).

6.2 Function Blocks for BAPS Overview

In addition to the standard function blocks, you can use manufacturer-defined function blocks if you have logged on manufacturer-defined libraries in a project.

Note: Logging on of libraries is described in the general help.

The following function blocks are available for BAPS:

Function block	**Brief description**
BAPS_INIT	Initialization of the BAPS (Baumüller drive parallel interface) in the Ωmega Drive-Line II
BAPS_PAR_READ	BAPS read parameter
BAPS_PAR_WRITE	BAPS write parameter
BAPS_PD_COMM2	Process data communication via the BAPS in the Ωmega Drive-Line II for a maximum of two referenced and actual values
BAPS_PD_COMM24	process data communication via the BAPS in the Ωmega Drive-Line II for a maximum of two referneced and 4 actual values, preferably in „2 reference and 4 actual values in same cycle mode“ (see w_COMMAND_REG)
BAPS_PD_COMM8	Process data communication via the BAPS in the Ωmega Drive-Line II for a maximum of eight referenced and actual values
BAPS_PD_CONTROL	BAPS process data communication monitoring
BAPS_SD_CONTROL	BAPS requirements data communication

6.2.1 BAPS_INIT

Description

You can use this function block for BAPS to initialize process data communication (cyclical communication) between the V-controller and the Ωmega Drive-Line II via the BAPS interface.

NOTE

FB BAPS_INIT uses library SYSTEM2_DLII_20bd00 or above.

Parameter input	Data type	Description
us_HW_TYPE	USINT 0	Ωmega Drive-Line II
i_EVENT	INT 0, 4, 8	Event
w_COMMAND_REG	WORD	BAPS control register
us_MODE	USINT 0, 1, 2	Mode
u_WR_PAR_NR0	UINT	Reference value parameter number 0
u_WR_PAR_NR1	UINT	Reference value parameter number 1
u_WR_PAR_NR2	UINT	Reference value parameter number 2
u_WR_PAR_NR3	UINT	Reference value parameter number 3
u_WR_PAR_NR4	UINT	Reference value parameter number 4
u_WR_PAR_NR5	UINT	Reference value parameter number 5
u_WR_PAR_NR6	UINT	Reference value parameter number 6
u_WR_PAR_NR7	UINT	Reference value parameter number 7
u_RD_PAR_NR0	UINT	Actual value parameter number 0
u_RD_PAR_NR1	UINT	Actual value parameter number 1
u_RD_PAR_NR2	UINT	Actual value parameter number 2
u_RD_PAR_NR3	UINT	Actual value parameter number 3
u_RD_PAR_NR4	UINT	Actual value parameter number 4
u_RD_PAR_NR5	UINT	Actual value parameter number 5
u_RD_PAR_NR6	UINT	Actual value parameter number 6
u_RD_PAR_NR7	UINT	Actual value parameter number 7
t_TIME	TIME	Monitoring time in ms
x_EN	BOOL	Enable initialization
x_RESET	BOOL	Reset

Parameter output	Data type	Description
w_STATUS_REG	WORD	BAPS status register
x_BUSY	BOOL	BUSY bit
b_SL_QUIT	BYTE	V-controller acknowledgement
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB BAPS_INIT makes it possible to initialize and reinitialize the BAPS for cyclical communication.

FBs BAPS_PD_COMM8, BAPS_PD_COMM2 or BAPS_PD_COMM24 (also referred to below as BAPS_PD_COMMxx) are available for cyclical communication.

Cyclical communication can be initialized for

up to two reference and two actual values using FB BAPS_PD_COMM2,

up to eight reference and eight actual values using FB BAPS_PD_COMM8,

up to two reference and four actual values using FB BAPS_PD_COMM24 [a)]

Input us_HW_TYPE:

At input us_HW_TYPE you state with us_HW_TYPE = 0 that FB BAPS_INIT is used in the Ωmega Drive-Line II (us_HW_TYPE ≠ 0 is not implemented).

NOTE

If input us_HW_TYPE is **not** assigned, this yields the presetting us_HW_TYPE = 0 (FB BAPS_INIT in the Ωmega Drive-Line II).

The system can process the BAPS_PD_COMMxx FB of cyclical communication in every cycle in the main program of an event task (to any event) or in an event task to the „BAPS process data“ event. In the latter case, the „BAPS process data“ event is initialized by FB BAPS_INIT .

The „BAPS process data“ event can be initialized with an interrupt level of 13 (low priority) or 14 (high priority).

Input i_EVENT:

With i_EVENT = 4, the system initializes the „BAPS process data“ event with an interrupt level of 13 (low).
With i_EVENT = 8, the system initializes the „BAPS process data“ event with an interrupt level of 14 (high).

If input i_EVENT is not assigned or i_EVENT = 0, the system does not initialize any events. If i_EVENT is not 0, 4 or 8, the system sets error bit 3 at output b_ERR.

---

a) From V-controller software version 000309 onwards

Inputs u_WR_PAR_NR0 to u_WR_PAR_NR7:

You state at the inputs the reference value parameter numbers of the reference values that are to be cyclically transferred

u_WR_PAR_NR0 to u_WR_PAR_NR7 (when using BAPS_PD_COMM8),

u_WR_PAR_NR0 and u_WR_PAR_NR1 (when using BAPS_PD_COMM2 or BAPS_PD_COMM24)

Inputs u_RD_PAR_NR0 to u_RD_PAR_NR7:

You state at the inputs the actual value parameter numbers of the actual values that are to be cyclically transferred

u_RD_PAR_NR0 **to** u_RD_PAR_NR7 (when using BAPS_PD_COMM8),

u_RD_PAR_NR0 **and** u_RD_PAR_NR1 (when using BAPS_PD_COMM2),

u_RD_PAR_NR0 **to** u_RD_PAR_NR3 (when using BAPS_PD_COMM24),

Input w_COMMAND_REG:

The following take place at input w_COMMAND_REG:

- selection of reference and actual value transfer,
- selection of the method of calculating the communications cycle time of reference and actual value transfer as well as
- setting of the communications cycle time of reference and actual value transfer

Bit No.	Meaning		
0	Reserved		
1, 2	Bit 2 0 0 1 1	Bit 1 0 1 0 1	Selection of reference and actual value transfer ($\rightarrow t_{cyc\ P}$) Time slice procedure direct specification of reference/actual value no. Two reference and two (four) actual values in the same cycle Reserved
3	Bit 3 0 1		Selection of the method of calculating the communications cycle time of reference and actual value transfer ($\rightarrow t_{cyc\ K}$) Counter Time slice
4, 5, 6, 7	1…15: 1…15:		Bit 3 = 0: Value of the counter The system internally increments a counter every 500µs. If the reading of this counter matches the value that is formed from bits 4 to 7, the system carries out process data communication (assuming that an event from the V-controller is pending). Bit 3 = 1: Number of the time slice Process data communication takes place in each case every 500µs after the time slice whose number is entered in bits 4 to 7 (assuming that an event from the V-controller is pending).
8 ... 15	Reserved		

Time slice of the V-controller for cyclical communication: 500 µs

$t_{cyc\ K}$ - Time between two communications via the BAPS interface

$t_{cyc\ P}$ - Time with which the parameters (reference and actual values) at position x are updated

Determination of $t_{cyc\ K}$: → bits 3 and 4 to 7

Counter: → bit 3 = 0

$t_{cyc\ K}$ = 0.5 ms * "value from bits 4 to 7"

Minimum value:	0.5 ms * 1	=	0.5 ms
Example:	0.5 ms * 2	=	1.0 ms
Example:	0.5 ms * 3	=	1.5 ms
Example:	0.5 ms * 4	=	2.0 ms
	etc.		
Maximum value:	0.5 ms * 15	=	7.5 ms

Time slice: → bit 3 = 1

$t_{cyc\ K}$ = 0.5 ms * 2 "value from bits 4 to 7"

Minimum value:	0.5 ms * 2^1	=	1 ms
Example:	0.5 ms * 2^2	=	2 ms
Example:	0.5 ms * 2^3	=	4 ms
Example:	0.5 ms * 2^4	=	8 ms
Example:	0.5 ms * 2^5	=	16 ms
	etc.		
Maximum value:	0.5 ms * 2^{15}	=	16384 ms

NOTE

After every cycle time $t_{cyc\ K}$, the system must call the respective function block in the Ωmega Drive-Line II for BAPS process data communication (FB BAPS_PD_COMMxx). For example: in an event task to the BAPS process data communication event, or in an event task to the SYNC signal network (CANsync) event.

NOTE

If BAPS process data communication is triggered via a synchronization signal or you use the Synchronized position reference value specification mode, you must set parameter 167 (Sync.-Slot) to $t_{cyc\ K}$ in µs.

Determination of $t_{cyc\ P}$: bits 1 and 2:

Time slice procedure (up to 8 reference values and 8 actual values):

$t_{cyc\ P\ x} = (2 * t_{cyc\ K}) * 2^x$; x: parameter number at FB: 0 to 7

$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	1 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	2 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 2} = (2 * t_{cyc\ K}) * 2^2 =$	$8 * t_{cyc\ K}$	=	4 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 3} = (2 * t_{cyc\ K}) * 2^3 =$	$16 * t_{cyc\ K}$	=	8 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 4} = (2 * t_{cyc\ K}) * 2^4 =$	$32 * t_{cyc\ K}$	=	16 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 5} = (2 * t_{cyc\ K}) * 2^5 =$	$64 * t_{cyc\ K}$	=	32 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 6} = (2 * t_{cyc\ K}) * 2^6 =$	$128 * t_{cyc\ K}$	=	128 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 7} = (2 * t_{cyc\ K}) * 2^7 =$	$256 * t_{cyc\ K}$	=	128 ms	(where $t_{cyc\ K}$ = 0.5 ms)

$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	4 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	8 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 2} = (2 * t_{cyc\ K}) * 2^2 =$	$8 * t_{cyc\ K}$	=	16 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 3} = (2 * t_{cyc\ K}) * 2^3 =$	$16 * t_{cyc\ K}$	=	32 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 4} = (2 * t_{cyc\ K}) * 2^4 =$	$32 * t_{cyc\ K}$	=	64 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 5} = (2 * t_{cyc\ K}) * 2^5 =$	$64 * t_{cyc\ K}$	=	128 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 6} = (2 * t_{cyc\ K}) * 2^6 =$	$128 * t_{cyc\ K}$	=	256 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 7} = (2 * t_{cyc\ K}) * 2^7 =$	$256 * t_{cyc\ K}$	=	512 ms	(where $t_{cyc\ K}$ = 2 ms)

Two reference values and two (four) actual values in the same cycle:

$t_{cyc\ P\ 0} = t_{cyc\ P\ 1}\ (= t_{cyc\ P\ 2} = t_{cyc\ P\ 3}) = t_{cyc\ K}$

Input us_MODE:

You use us_MODE = 0 to state that FB BAPS_INIT is called once for process data configuration. This is necessary at first initialization of the BAPS interface after a reset, a warm boot or a cold boot.

You use us_MODE = 1 to state that FB BAPS_INIT is used to change the process data configuration when process data communication is blocked.

The following applies additionally from V-controller software version 000309 onwards:

Using us_MODE = 2, you can overwrite the reference value parameter numbers of reference values 0 and 1 that are to be cyclically transferred as well as the actual value parameter number of actual values 0, 1, 2 und 3 that are to be cyclically transferred. This is provided for reinitializing or reconfiguring the BAPS during ongoing process data communication.

NOTE

Start initialization of the BAPS using us_MODE = 2 is **not** possible. In us_MODE = 2, FB BAPS_INIT does not issue **any** OK or error messages!

Input t_TIME:

You set the monitoring time in seconds at input t_TIME. If input t_TIME is not assigned, this yields a presetting of 3 s.

Input x_EN:

Using x_EN = TRUE, you enable initialization or process data configuration of the BAPS. The default setting is x_EN = TRUE.

Input x_RESET:

You use x_RESET = TRUE to reset FB BAPS_INIT.

Output w_STATUS_REG:

With bit 0 set, output w_STATUS_REG indicates that the V-controller is synchronized to the Trigger Controller signal. (See “The Interrupt Sources and Trigger Signals” on page 52.)

Output x_BUSY:

With us_MODE = 1, output x_BUSY indicates by TRUE that initialization of the process data configuration is active; with us_MODE = 0 or 2, x_BUSY stays FALSE.

Output b_SL_QUIT:

The system reports at output b_SL_QUIT the V-controller acknowledgement after initialization has been completed.

Value b_SL_QUIT	**Meaning**
16#00	No meaning
16#01	Reference value has been read/actual value has been written
16#02	Configuration/initialization has been carried out correctly
16#03 – 16#7F	Reserved
16#80	An uninterpretable command has been received
16#81	Configuration/initialization has not been carried out
16#82	Actual value cannot be read
16#83	Reference value cannot be written
16#84 – 16#FE	Reserved
16#FF	No meaning

Output x_OK:

Output x_OK is set to TRUE if the BAPS and possibly the „BAPS process data“ event have been initialized correctly.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR (does NOT apply to us_MODE = 2).

Error byte b_ERR:

Bit number	Error
0	Output b_SL_QUIT ≠ 16#02
1	Reserved
2	Reserved
3	Input i_EVENT is not 4 or 8 (or 0 for no event)
4	Timeout
5	Error setting up the event
6 – 7	Reserved

6.2.2 BAPS_PAR_READ

Description

You can use this function block for BAPS to read a requirements data value (parameter) of the V-controller via the BAPS interface and FB BAPS_SD_CONTROL.

NOTE

FB BAPS_PAR_READ uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BAPS_SD_DATA	BAPS_BMSTRUCT	Communications data
u_PAR_NR	UINT	Parameter number
us_PAR_ELEMENT	USINT 7	Parameter element
x_EN	BOOL	Enable
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BAPS_SD_DATA	BAPS_BMSTRUCT	Communications data
ud_PAR_VALUE	UDINT	Read parameter value
x_PAR_FORMAT	BOOL	Format of the parameter value
x_BUSY	BOOL	Communication is active
i_ERR_DETAIL	INT	Operating system error
i_ERR_COMM	INT	Communications error
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB BAPS_PAR_READ transfers with the values of inputs u_PAR_NR and us_PAR_ELEMENT a read parameter job to FB BAPS_SD_CONTROL. FB BAPS_SD_CONTROL passes on the read parameter job to the V-controller and returns the data that the V-controller returns to FB BAPS_PAR_READ. The system displays at output ud_PAR_VALUE the parameter element that the V-controller requested, and displays the format at output x_PAR_FORMAT. If errors occur while the read parameter job is being carried out, the system displays them at the error outputs of FB BAPS_PAR_READ and specifies them in more detail.

You can use (instantiate) FB BAPS_PAR_READ several times. FB BAPS_SD_CONTROL is used only once and it processes in each case one read parameter job (or write parameter job, see FB BAPS_PAR_WRITE).

Input/output _BAPS_SD_DATA:

At _BAPS_SD_DATA, you must connect a global variable of data type BAPS_BMSTRUCT.

Example:

```
_BAPS_SD_DATEN : BAPS_BMSTRUCT;
```

Where:

_BAPS_SD_DATEN	is the variable name with the data type short designation "_" for STRUCT
BAPS_BMSTRUCT	is the data type

The system uses this variable to exchange data with FB BAPS_SD_CONTROL. This variable is connected with the FBs of requirements data communication of the BAPS – this also applies if you use FBs BAPS_PAR_READ and/or BAPS_PAR_WRITE several times.

Input u_PAR_NR:

At input u_PAR_NR, you state the parameter number of the parameter whose value you want to read.

Input us_PAR_ELEMENT:

At input us_PAR_ELEMENT, you state the element of the parameter to be read. If us_PAR_ELEMENT is not assigned, this yields a presetting of us_PAR_ELEMENT = 7 (≡ value of the parameter).

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY=FALSE, it is assumed that communication was cancelled deliberately. In this case, FB BAPS_PAR_WRITE and FB BAPS_SD_CONTROL must each be reset using x_RESET = TRUE.

Input x_RESET:

You use x_RESET = TRUE to reset the FB.

Output ud_PAR_VALUE:

The read parameter value is output at output ud_PAR_VALUE.

Output x_PAR_FORMAT:

The format of parameter value is made available at output x_PAR_FORMAT. x_PAR_FORMAT = FALSE means word format, x_PAR_FORMAT = TRUE means doubleword format.

Output x_BUSY:

Output x_BUSY indicates by TRUE the communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, i_ERR_DETAIL, i_ERR_COMM:

If an error occurs, the system sets error bit x_ERR to TRUE and specifies the error at outputs i_ERR_DETAIL und i_ERR_COMM.

Error number i_ERR_DETAIL:

i_ERR_DETAIL	**Error**
0	No error
-1	Data error that is not specified in more detail
-2	Value less than minimum value
-3	Value greater than maximum value
-4	Element must not be written to
-5	No element present
-6	Element not currently available due to calculation
-7	Wrong transfer data format
-8	Wrong number of elements at writing

Error number i_ERR_COMM:

i_ERR_COMM	**Error**
0	No error
-1	Communications error

6.2.3 BAPS_PAR_WRITE

Description

You can use this function block for BAPS to write a requirements data value (parameter) of the V-controller via the BAPS interface and FB BAPS_SD_CONTROL.

NOTE

FB BAPS_PAR_WRITE uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BAPS_SD_DATA	BAPS_BMSTRUCT	Communications data
u_PAR_NR	UINT	Parameter number
x_PAR_FORMAT	BOOL	Format of the parameter value
ud_PAR_VALUE	UDINT	Parameter value
x_EN_ERR_FORMAT	BOOL	Display format error ON/OFF
x_EN	BOOL	Enable
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BAPS_SD_DATA	BAPS_BMSTRUCT	Communications data
x_BUSY	BOOL	Communication is active
i_ERR_DETAIL	INT	Operating system error
i_ERR_COMM	INT	Communications error
x_ERR	BOOL	Group error bit
x_OK	BOOL	OK bit

FB BAPS_PAR_WRITE transfers with the values of inputs u_PAR_NR, x_PAR_FORMAT and ud_PAR_VALUE a write parameter job to FB BAPS_SD_CONTROL. FB BAPS_SD_CONTROL passes on the write parameter job to the V-controller and returns the result of communication that the V-controller returns to FB BAPS_PAR_WRITE. If errors occur while the write parameter job is being carried out, the system displays them at the error outputs of FB BAPS_PAR_WRITE and specifies them in more detail.

You can use (instantiate) FB BAPS_PAR_WRITE several times. FB BAPS_SD_CONTROL is used only once and it processes in each case one write parameter job (or write parameter job, see FB BAPS_PAR_WRITE).

Input/output _BAPS_SD_DATA:

At _BAPS_SD_DATA, you must connect a global variable of data type BAPS_BMSTRUCT.

Example:

```
_BAPS_SD_DATEN : BAPS_BMSTRUCT;
```

Where:

_BAPS_SD_DATEN	is the variable name with the data type short designation "_" for STRUCT
BAPS_BMSTRUCT	is the data type

The system uses this variable to exchange data with FB BAPS_SD_CONTROL. This variable is connected with the FBs of requirements data communication of the BAPS – this also applies if you use FBs BAPS_PAR_READ and/or BAPS_PAR_WRITE several times.

Input u_PAR_NR:

At input u_PAR_NR, you state the parameter number of the parameter whose value you want to write.

Input x_PAR_FORMAT:

At input x_PAR_FORMAT, you set the format of the value to be written. x_PAR_FORMAT = FALSE means word format, x_PAR_FORMAT = TRUE means doubleword format.

Input ud_PAR_VALUE:

You state the parameter value to be written at input ud_PAR_VALUE.

Input x_EN_ERR_FORMAT:

You can use input x_EN_ERR_FORMAT to set display of a format error i_ERR_DETAIL = -7 in error bit x_ERR. If you do not want this to be displayed, set x_EN_ERR_FORMAT to FALSE. If input x_EN_ERR_FORMAT is not assigned, this yields a presetting of x_EN_ERR_FORMAT = TRUE and i_ERR_DETAIL = -7 is displayed in x_ERR.

NOTE

If x_EN_ERR_FORMAT = FALSE, the system displays a format error in i_ERR_DETAIL (= -7); however, the format error is not displayed in error bit x_ERR!
In this case, the OK bit is set to TRUE despite the format error!

Input x_EN:

Communication is started by means of x_EN = TRUE.

If x_EN is set to FALSE before x_BUSY=FALSE, it is assumed that communication was cancelled deliberately. In this case, FB BAPS_PAR_WRITE and FB BAPS_SD_CONTROL must each be reset using x_RESET = TRUE.

Input x_RESET:

You use x_RESET = TRUE to reset the FB.

Output x_BUSY:

Output x_BUSY indicates by TRUE the communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, i_ERR_DETAIL, i_ERR_COMM:

If an error occurs, the system sets error bit x_ERR to TRUE and specifies the error at outputs i_ERR_DETAIL und i_ERR_COMM.

Error number i_ERR_DETAIL:

i_ERR_DETAIL	**Error**
0	No error
-1	Data error that is not specified in more detail
-2	Value less than minimum value
-3	Value greater than maximum value
-4	Element must not be written to
-5	No element present
-6	Element not currently available due to calculation!
-7	Wrong transfer data format
-8	Wrong number of elements at writing

Error number i_ERR_COMM:

i_ERR_COMM	**Error**
0	No error
-1	Communications error

6.2.4 BAPS_PD_COMM2

Description

You can use this function block for BAPS to initialize process data communication between the V-controller and the Ωmega Drive-Line II via the BAPS interface

Parameter input	Data type	Description
us_HW_TYPE	USINT 0	Ωmega Drive-Line II
w_CONTROLWORD	WORD	Control word
w_COMMAND_REG	WORD	Control register
us_MODE	USINT 0, 1	Mode
ud_WR_VALUE0	UDINT	Reference value 0
ud_WR_VALUE1	UDINT	Reference value 1
x_EN	BOOL	Enable

Parameter output	Data type	Description
w_STATUSWORD	WORD	Status word
w_STATUS_REG	WORD	Status register
ud_RD_VALUE0	UDINT	Actual value 0
ud_RD_VALUE1	UDINT	Actual value 1
b_SL_QUIT	BYTE	Controller acknowledgement
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

The system sends the reference values and the control word to the V-controller; the V-controller receives and outputs the actual values and the status word.

Input us_HW_TYPE:

At input us_HW_TYPE you state with us_HW_TYPE = 0 that FB BAPS_PD_COMM2 is used in the Ωmega Drive-Line II (us_HW_TYPE ≠ 0 is not implemented).

NOTE

If input us_HW_TYPE is **not** assigned, this yields the presetting us_HW_TYPE=0 (FB BAPS_PD_COMM2 in the Ωmega Drive-Line II).

Input w_CONTROLWORD:

At input w_CONTROLWORD, you state the control word that is to be sent to the V-controller.

Input w_COMMAND_REG:

The following take place at input w_COMMAND_REG:

- selection of reference and actual value transfer,
- selection of the method of calculating the communications cycle time of reference and actual value transfer as well as
- setting of the communications cycle time of reference and actual value transfer

Bit No.	Meaning		
0	Reserved		
1, 2	Bit 2 0 0 1 1	Bit 1 0 1 0 1	Selection of reference and actual value transfer (→ $t_{cyc\ P}$) Time slice procedure direct specification of reference/actual value no. Two reference and two actual values in the same cycle Reserved
3	Bit 3 0 1		Selection of the method of calculating the communications cycle time of reference and actual value transfer (→ $t_{cyc\ K}$) Counter Time slice
4, 5, 6, 7	1…15: 1…15:		Bit 3 = 0: Value of the counter The system internally increments a counter every 500µs. If the reading of this counter matches the value that is formed from bits 4 to 7, the system carries out process data communication (assuming that an event from the V-controller is pending). Bit 3 = 1: Number of the time slice Process data communication takes place in each case every 500µs after the time slice whose number is entered in bits 4 to 7 (assuming that an event from the V-controller is pending).
8 ... 15	Reserved		

Time slice of the V-controller for cyclical communication: 500 µs

$t_{cyc\ K}$ - Time between two communications via the BAPS

$t_{cyc\ P}$ - Time with which the system updates the parameters (reference and actual values) at position x (inputs ud_WR_VALUEx and outputs ud_RD_VALUEx)

Determination of $t_{cyc\ K}$: → bits 3 and 4 to 7

Counter: → bit 3 = 0

$t_{cyc\ K}$ = 0.5 ms * “value from bits 4 to 7”

Minimum value:	0.5 ms * 1	=	0.5 ms
Example:	0.5 ms * 2	=	1.0 ms
Example:	0.5 ms * 3	=	1.5 ms
Example:	0.5 ms * 4	=	2.0 ms
	etc.		
Maximum value:	0.5 ms * 15	=	7.5 ms

Time slice: → bit 3 = 1

$t_{cyc\ K}$ = 0.5 ms * 2 "value from bits 4...7"

Minimum value:	0.5 ms * 2^1	=	1 ms
Example:	0.5 ms * 2^2	=	2 ms
Example:	0.5 ms * 2^3	=	4 ms
Example:	0.5 ms * 2^4	=	8 ms
Example:	0.5 ms * 2^5	=	16 ms
	etc.		
Maximum value:	0.5 ms * 2^{15}	=	16384 ms

NOTE

After every cycle time $t_{cyc\ K}$, must call FB BAPS_PD_COMM2 in the Ωmega Drive-Line II (e.g. in an event task to the „BAPS process data" event or in an event task to the SYNC signal network (CANsync) event).

NOTE

If (BAPS) process data communication is triggered via a synchronization signal or you use the Synchronized position reference value specification mode, you must set parameter 167 (Sync.-Slot) to $t_{cyc\ K}$ in µs.

Determination of $t_{cyc\ P}$: bits 1 and 2:

Time slice procedure (up to 2 reference values and 2 actual values):

$t_{cyc\ P\ x} = (2 * t_{cyc\ K}) * 2^x$; x: referenced/actual value number at FB: 0 to 1 (inputs ud_WR_VALUEx and outputs ud_RD_VALUEx)

$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	1 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	2 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	4 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	8 ms	(where $t_{cyc\ K}$ = 2 ms)

Two reference values and two actual values in the same cycle:

$t_{cyc\ P\ 0} = t_{cyc\ P\ 1} = t_{cyc\ K}$

Input us_MODE:

Using us_MODE = 0, you state that FB BAPS_PD_COMM2 is called in a cyclical main program or in an event task to any event (e.g. to the SYNC signal network (CANsync) event).

Using us_MODE = 1 you state that FB BAPS_PD_COMM2 is called in an event task to the „BAPS process data“ event (see description of FB BAPS_INIT, input i_EVENT).

If us_MODE is not assigned, this yields the presetting us_MODE = 0 (use of FB BAPS_PD_COMM2 in the cyclical main program or in an event task to any event).

Inputs ud_WR_VALUE0, ud_WR_VALUE3:

The reference values are connected to inputs ud_WR_VALUE0 and ud_WR_VALUE1 whose parameter numbers were stated at FB BAPS_INIT (inputs u_WR_PAR_NR0 and u_WR_PAR_NR1) at initialization of BAPS process data communication.

Input x_EN:

Communication is enabled by means of x_EN = TRUE. The default setting is x_EN = TRUE, i.e. communication is enabled.

Output w_STATUSWORD:

At output w_STATUSWORD, the system outputs the V-controller's status word.

Output w_STATUS_REG:

With bit 0 set, output w_STATUS_REG (BAPS status register) indicates that the V-controller is synchronized to the Trigger Controller signal. (See “The Interrupt Sources and Trigger Signals” on page 52.)

Outputs ud_RD_VALUE0, ud_RD_VALUE3:

The actual values are output at outputs ud_WR_VALUE0 and ud_WR_VALUE1 whose parameter numbers were stated at FB BAPS_INIT (inputs u_RD_PAR_NR0 and u_RD_PAR_NR1) at initialization of BAPS process data communication.

Output b_SL_QUIT:

The system reports at output b_SL_QUIT the V-controller acknowledgement after communication has been completed.

Value b_SL_QUIT	Meaning
16#00	No meaning
16#01	Reference value has been read/actual value has been written
16#02	Configuration/initialization has been carried out correctly
16#03 – 16#7F	Reserved
16#80	An uninterpretable command has been received
16#81	Configuration/initialization has not been carried out
16#82	Actual value cannot be read
16#83	Reference value cannot be written
16#84 – 16#FE	Reserved
16#FF	No meaning

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit number	Error
0	Output b_SL_QUIT ≠ 16#01
1 -7	Reserved

6.2.5 BAPS_PD_COMM24

Description

You can use this function block for BAPS to initialize process data communication between the V-controller and the Ωmega Drive-Line II via the BAPS interface [a)]

Parameter input	Data type	Description
us_HW_TYPE	USINT 0	Ωmega Drive-Line II
w_CONTROLWORD	WORD	Control word
w_COMMAND_REG	WORD	Control register
us_MODE	USINT	Mode
ud_WR_VALUE0	UDINT	Reference value 0
ud_WR_VALUE1	UDINT	Reference value 1
t_RE_INIT_TIME	TIME	Monitoring time in ms for reinitializing
x_RE_INIT_START	BOOL	Start (edge) for reinitializing
x_EN	BOOL	Enable

Parameter input	Data type	Description
w_STATUSWORD	WORD	Status word
w_STATUS_REG	WORD	Status register
ud_RD_VALUE0	UDINT	Actual value 0
ud_RD_VALUE1	UDINT	Actual value 1
ud_RD_VALUE2	UDINT	Actual value 2
ud_RD_VALUE3	UDINT	Actual value 3
b_SL_QUIT	BYTE	Controller acknowledgement
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_RE_INIT_ERR	BOOL	Error bit of reinitializing
x_RE_INIT_OK	BOOL	OK bit of reinitializing

The system sends the reference values and the control word to the V-controller; the V-controller receives and outputs the actual values and the status word.

Input us_HW_TYPE:

At input us_HW_TYPE you state with us_HW_TYPE = 0 that FB BAPS_PD_COMM24 is used in the Ωmega Drive-Line II (us_HW_TYPE ≠ 0 is not implemented).

NOTE

If input us_HW_TYPE is **not** assigned, this yields the presetting us_HW_TYPE=0 (FB BAPS_PD_COMM24 in the Ωmega Drive-Line II).

a) From V-controller software version 000309 onwards

Input w_CONTROLWORD:

At input w_CONTROLWORD, you state the control word that is to be sent to the V-controller.

Input w_COMMAND_REG:

The following take place at input w_COMMAND_REG:

- selection of reference and actual value transfer,
- selection of the method of calculating the communications cycle time of reference and actual value transfer as well as
- setting of the communications cycle time of reference and actual value transfer

Bit No.	Meaning		
0	Reserved		
1, 2	Bit 2 0 0 1 1	Bit 1 0 1 0 1	Selection of reference and actual value transfer (→ $t_{cyc\ P}$) Time slice procedure direct specification of reference/actual value no. Two reference and two (four) actual values in the same cycle Reserved
3	Bit 3 0 1		Selection of the method of calculating the communications cycle time of reference and actual value transfer (→ $t_{cyc\ K}$) Counter Time slice
4, 5, 6, 7	1…15: 1…15:		Bit 3 = 0: Value of the counter The system internally increments a counter every 500µs. If the reading of this counter matches the value that is formed from bits 4 to 7, the system carries out process data communication (assuming that an event from the V-controller is pending). Bit 3 = 1: Number of the time slice Process data communication takes place in each case every 500µs after the time slice whose number is entered in bits 4 to 7 (assuming that an event from the V-controller is pending).
8 ... 15	Reserved		

Time slice of the V-controller for cyclical communication: 500 µs

$t_{cyc\ K}$ - Time between two communications via the BAPS

$t_{cyc\ P}$ - Time with which the system updates the parameters (reference and actual values) at position x (inputs ud_WR_VALUEx and outputs ud_RD_VALUEx)

Determination of $t_{cyc\ K}$: → bits 3 and 4 to 7

Counter: → bit 3 = 0

$t_{cyc\ K}$ = 0.5 ms * "value from bits 4 to 7"

Minimum value:	0.5 ms * 1	=	0.5 ms
Example:	0.5 ms * 2	=	1.0 ms
Example:	0.5 ms * 3	=	1.5 ms
Example:	0.5 ms * 4	=	2.0 ms
	etc.		
Minimum value:	0.5 ms * 15	=	7.5 ms

Time slice: → bit 3 = 1

$t_{cyc\ K}$ = 0.5 ms * 2 "value from bits 4...7"

Minimum value:	0.5 ms * 2^1	=	1 ms
Example:	0.5 ms * 2^2	=	2 ms
Example:	0.5 ms * 2^3	=	4 ms
Example:	0.5 ms * 2^4	=	8 ms
Example:	0.5 ms * 2^5	=	16 ms
	etc.		
Maximum value:	0.5 ms * 2^{15}	=	16384 ms

NOTE

After every cycle time $t_{cyc\ K}$, must call FB BAPS_PD_COMM24 in the Ωmega Drive-Line II (e.g. in an event task to the „BAPS process data" event or in an event task to the SYNC signal network (CANsync) event).

NOTE

If (BAPS) process data communication is triggered via a synchronization signal or you use the Synchronized position reference value specification mode, you must set parameter 167 (Sync.-Slot) to $t_{cyc\ K}$ in µs.

Determination of $t_{cyc\ P}$: bits 1 and 2:

Time slice procedure (up to 2 reference values and 4 actual values):

$t_{cyc\ P\ x} = (2 * t_{cyc\ K}) * 2^x$;	x:	Referenced/actual value number at FB: 0 to 1 or 3 (inputs ud_WR_VALUEx and outputs ud_RD_VALUEx)		
$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	1 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	2 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 2} = (2 * t_{cyc\ K}) * 2^2 =$	$8 * t_{cyc\ K}$	=	4 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 3} = (2 * t_{cyc\ K}) * 2^3 =$	$16 * t_{cyc\ K}$	=	8 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	4 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	8 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 2} = (2 * t_{cyc\ K}) * 2^2 =$	$8 * t_{cyc\ K}$	=	16 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 3} = (2 * t_{cyc\ K}) * 2^3 =$	$16 * t_{cyc\ K}$	=	32 ms	(where $t_{cyc\ K}$ = 2 ms)

Two reference values and four actual values in the same cycle:

$t_{cyc\ P\ 0} = t_{cyc\ P\ 1} = t_{cyc\ P\ 2} = t_{cyc\ P\ 3} = t_{cyc\ K}$

Input us_MODE:

Using us_MODE = 0, you state that FB BAPS_PD_COMM24 is called in a cyclical main program or in an event task to any event (e.g. to the SYNC signal network (CANsync) event).

Using us_MODE = 1 you state that FB BAPS_PD_COMM24 is called in an event task to the „BAPS process data“ event (see description of FB BAPS_INIT, input i_EVENT).

If us_MODE is not assigned, this yields the presetting us_MODE = 0 (use of FB BAPS_PD_COMM24 in the cyclical main program or in an event task to any event).

Inputs ud_WR_VALUE0, ud_WR_VALUE3:

The reference values are connected to inputs ud_WR_VALUE0 and ud_WR_VALUE1 whose parameter numbers were stated at FB BAPS_INIT (inputs u_WR_PAR_NR0 and u_WR_PAR_NR1) at initialization of BAPS process data communication. [a)]

Input x_EN:

Communication is enabled by means of x_EN = TRUE. The default setting is x_EN = TRUE, i.e. communication is enabled.

For inputs t_RE_INIT_TIME and x_RE_INIT_START, see below.

---

a) Function two reference values and four actual values in the same cycle is implemented from V-controller software version 000309 onwards.

Output w_STATUSWORD:

At output w_STATUSWORD, the system outputs the V-controller's status word.

Output w_STATUS_REG:

With bit 0 set, output w_STATUS_REG (BAPS status register) indicates that the V-controller is synchronized to the Trigger Controller signal. (See “The Interrupt Sources and Trigger Signals” on page 52.)

Outputs ud_RD_VALUE0 to ud_RD_VALUE3:

The actual values are output at outputs ud_RD_VALUE0 to ud_RD_VALUE3 whose parameter numbers were stated at FB BAPS_INIT (inputs u_RD_PAR_NR0 to u_RD_PAR_NR3) at initialization of BAPS process data communication. [a)]

Output b_SL_QUIT:

The system reports at output b_SL_QUIT the V-controller acknowledgement after communication has been completed.

Value b_SL_QUIT	**Meaning**
16#00	No meaning
16#01	Reference value has been read/actual value has been written
16#02	Configuration/initialization has been carried out correctly
16#03	Reinitialization is active
16#04 – 16#7F	Reserved
16#80	An uninterpretable command has been received
16#81	Configuration/initialization has not been carried out
16#82	Actual value cannot be read
16#83	Reference value cannot be written
16#84	Reserved
16#85	Reserved
16#86	Reference value cannot be written (reinitializing)
16#87	Actual value cannot be read (reinitializing)
16#88 – 16#FE	Reserved
16#FF	No meaning

For outputs x_RE_INIT_ERR und x_RE_INIT_OK, see below.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

a) Function two reference values and four actual values in the same cycle is implemented from V-controller software version 000309 onwards.

Error byte b_ERR:

Bit Number	Error
0	Output b_SL_QUIT ≠ BYTE#16#01
1	Timeout (reinitializing)
2 – 7	Reserved

From V-controller software version 000309 onwards, it is to possible to reinitialize or reparameterize the BAPS process data configuration during ongoing process data communication. For this, the system must run through FB BAPS_INIT in a cyclical task in us_MODE = 2. For more information on this topic, refer to the description of FB BAPS_INIT and see below.

Input t_RE_INIT_TIME:

At input t_RE_INIT_TIME, you state the monitoring time for reinitializing. The default setting is t_RE_INIT_TIME = 1 s. If reinitializing is not completed within this time, the system sets bit 1 in error byte b_ERR.

Input x_RE_INIT_START:

At input x_RE_INIT_START, reinitializing is started by means of x_RE_INIT_START = TRUE.

You state a timeout at reinitializing by means of b_ERR = 16#02 and signal it by the set error bit, x_ERR. If you state an invalid parameter number at reinitializing, the system does **not** set a bit in error byte b_ERR and error bit x_ERR stays FALSE. The system reports that an invalid parameter number has been stated by means of b_SL_QUIT = 16#86 or b_SL_QUIT = 16#87 and x_RE_INIT_ERR = TRUE.

Function of reinitializing:

From V-controller software version 000309 onwards, you can change (i.e. reinitialize) the parameter numbers of the reference and/or actual values during ongoing process data communication (see input w_COMMAND_REG, selection of reference and actual value transfer = 2 reference and 4 actual values in the same cycle).

During the reinitialization stage, the reference and actual values that are to be reinitialized are **not** defined. The reference and actual values that are not reinitialized stay valid.

Sequence of reinitializing:

For reinitializing BAPS process data communication, you must tell the V-controller which parameter numbers are to be changed and how they are to be changed. To do this, you call FB BAPS_INIT in us_MODE = 2.

NOTE

One instance of FB BAPS_INIT is available for this in a POU that is called cyclically and not in the event task in which FB BAPS_PD_COMM24 is called.

FB BAPS_INIT, inputs u_WR_PAR_NR0, u_WR_PAR_NR1:

At inputs u_WR_PAR_NR0 and u_WR_PAR_NR1 of FB BAPS_INIT, you state the new parameter numbers for reference value 0 and reference value 1.

FB BAPS_INIT, inputs u_RD_PAR_NR0 to u_RD_PAR_NR3:

At inputs u_RD_PAR_NR0 to u_RD_PAR_NR3 of FB BAPS_INIT, you state the new parameter numbers for actual value 0 to actual value 3.

If parameter numbers do not need to be changed, you state the previous parameter number.

If one of inputs u_WR_PAR_NR0, u_WR_PAR_NR1, u_RD_PAR_NR0, u_RD_PAR_NR1, u_RD_PAR_NR2, or u_RD_PAR_NR3 is not assigned, you must enter 0 as the parameter number.

FB BAPS_INIT, input x_EN:

If the inputs are interconnected appropriately, you use x_EN = TRUE (of FB BAPS_INIT) to start entry of the parameter numbers in the appropriate registers of the BAPS interface. It is only necessary to run through FB BAPS_INIT once for this.

NOTE

In us_MODE = 2, FB BAPS_INIT does not issue **any** OK or error messages!

Input x_RE_INIT_TRUE, output x_RE_INIT_OK:

After FB BAPS_INIT has been run through, the system starts reinitializing at FB BAPS_PD_COMM24 with x_RE_INIT_START = TRUE. From now on, the values of the reference and actual value parameters to be reinitialized are not defined until the end of the reinitialization stage is signalled by x_RE_INIT_OK = TRUE (or x_RE_INIT_ERR = TRUE).

In the case of an OK (x_RE_INIT_OK = TRUE), the system transfers the values of reference values 0 and 1 to the reinitialized reference value parameters in the V-controller and "fetches" the values of actual values 0 to 3 from the reinitialized actual value parameter number in the V-controller.All the values are defined again.

Output x_RE_INIT_ERR, b_ERR, b_SL_QUIT:

In the case of an error, the system signals a timeout after time x_RE_INIT_TIME expires by a set bit 1 in error byte b_ERR (of FB BAPS_PD_COMM24) and by x_ERR = TRUE.

If the case of one (or two) invalid reference value parameter number(s), the system outputs the value 16#86 at output b_SL_QUIT and sets x_RE_INIT_ERR = TRUE.

If the case of one (or more) invalid actual value parameter number(s), the system outputs the value 16#87 at output b_SL_QUIT and sets x_RE_INIT_ERR = TRUE.

If the case of one (or two) invalid reference value parameter number(s) **and** one (or more) invalid actual value parameter number(s), the system outputs the value 16#86 at output b_SL_QUIT and sets x_RE_INIT_ERR = TRUE.

6.2.6 BAPS_PD_COMM8

Description

You can use this function block for BAPS to initialize process data communication between the V-controller and the Ωmega Drive-Line II via the BAPS interface

Parameter input	Data type	Description
us_HW_TYPE	USINT 0	Ωmega Drive-Line II
w_CONTROLWORD	WORD	Control word
w_COMMAND_REG	WORD	Control register
us_MODE	USINT 0, 1	Mode
ud_WR_VALUE0	UDINT	Reference value 0
ud_WR_VALUE1	UDINT	Reference value 1
ud_WR_VALUE2	UDINT	Reference value 2
ud_WR_VALUE3	UDINT	Reference value 3
ud_WR_VALUE4	UDINT	Reference value 4
ud_WR_VALUE5	UDINT	Reference value 5
ud_WR_VALUE6	UDINT	Reference value 6
ud_WR_VALUE7	UDINT	Reference value 7
x_EN	BOOL	Enable

Parameter output	Data type	Description
w_STATUSWORD	WORD	Status word
w_STATUS_REG	WORD	Status register
ud_RD_VALUE0	UDINT	Actual value 0
ud_RD_VALUE1	UDINT	Actual value 1
ud_RD_VALUE2	UDINT	Actual value 2
ud_RD_VALUE3	UDINT	Actual value 3
ud_RD_VALUE4	UDINT	Actual value 4
ud_RD_VALUE5	UDINT	Actual value 5
ud_RD_VALUE6	UDINT	Actual value 6
ud_RD_VALUE7	UDINT	Actual value 7
b_SL_QUIT	BYTE	Controller acknowledgement
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

The system sends the reference values and the control word to the V-controller; the V-controller receives and outputs the actual values and the status word.

Input us_HW_TYPE:

At input us_HW_TYPE you state with us_HW_TYPE = 0 that FB BAPS_PD_COMM8 is used in the Ωmega Drive-Line II (us_HW_TYPE ≠ 0 is not implemented).

NOTE

If input us_HW_TYPE is **not** assigned, this yields the presetting us_HW_TYPE=0 (FB BAPS_PD_COMM8 in the Ωmega Drive-Line II).

Input w_CONTROLWORD:

At input w_CONTROLWORD, you state the control word that is to be sent to the V-controller.

Input w_COMMAND_REG:

The following take place at input w_COMMAND_REG:

- selection of reference and actual value transfer,
- selection of the method of calculating the communications cycle time of reference and actual value transfer as well as
- setting of the communications cycle time of reference and actual value transfer

Bit No.	Meaning		
0	Reserved		
1, 2	Bit 2 0 0 1 1	Bit 1 0 1 0 1	Selection of reference and actual value transfer (→ $t_{cyc\ P}$) Time slice procedure direct specification of reference/actual value no. Two reference and two (four) actual values in the same cycle Reserved
3	Bit 3 0 1		Selection of the method of calculating the communications cycle time of reference and actual value transfer (→ $t_{cyc\ K}$) Counter Time slice
4, 5, 6, 7	1…15:		Bit 3 = 0: Value of the counter The system internally increments a counter every 500µs. If the reading of this counter matches the value that is formed from bits 4 to 7, the system carries out process data communication (assuming that an event from the V-controller is pending).
	1…15:		Bit 3 = 1: Number of the time slice Process data communication takes place in each case every 500µs after the time slice whose number is entered in bits 4 to 7 (assuming that an event from the V-controller is pending).
8 ... 15	Reserved		

Time slice of the V-controller for cyclical communication: 500 µs

$t_{cyc\ K}$ - Time between two communications via the BAPS

$t_{cyc\ P}$ - Time with which the system updates the parameters (reference and actual values) at position x (inputs ud_WR_VALUEx and outputs ud_RD_VALUEx)

Determination of $t_{cyc\ K}$: → bits 3 and 4 to 7

Counter: → bit 3 = 0

$t_{cyc\ K}$ = 0.5 ms * "value from bits 4 to 7"

Minimum value:	0.5 ms * 1	=	0.5 ms
Example:	0.5 ms * 2	=	1.0 ms
Example:	0.5 ms * 3	=	1.5 ms
Example:	0.5 ms * 4	=	2.0 ms
	etc.		
Minimum value:	0.5 ms * 15	=	7.5 ms

Time slice: → bit 3 = 1

$t_{cyc\ K}$ = 0.5 ms * 2 "value from bits 4...7"

Minimum value:	0.5 ms * 2^1	=	1 ms
Example:	0.5 ms * 2^2	=	2 ms
Example:	0.5 ms * 2^3	=	4 ms
Example:	0.5 ms * 2^4	=	8 ms
Example:	0.5 ms * 2^5	=	16 ms
	etc.		
Maximum value:	0.5 ms * 2^{15}	=	16384 ms

NOTE

After every cycle time $t_{cyc\ K}$, must call FB BAPS_PD_COMM8 in the Ωmega Drive-Line II (e.g. in an event task to the „BAPS process data" event or in an event task to the SYNC signal network (CANsync) event).

NOTE

If (BAPS) process data communication is triggered via a synchronization signal or you use the Synchronized position reference value specification mode, you must set parameter 167 (Sync.-Slot) to $t_{cyc\ K}$ in µs.

Determination of $t_{cyc\ P}$: bits 1 and 2:

Time slice procedure (up to 8 reference values and 8 actual values):

$t_{cyc\ P\ x} = (2 * t_{cyc\ K}) * 2^x$; x: Referenced/actual value number at FB: 0 to 7 (inputs ud_WR_VALUEx and outputs ud_RD_VALUEx)

$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	1 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	2 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 2} = (2 * t_{cyc\ K}) * 2^2 =$	$8 * t_{cyc\ K}$	=	4 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 3} = (2 * t_{cyc\ K}) * 2^3 =$	$16 * t_{cyc\ K}$	=	8 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 4} = (2 * t_{cyc\ K}) * 2^4 =$	$32 * t_{cyc\ K}$	=	16 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 5} = (2 * t_{cyc\ K}) * 2^5 =$	$64 * t_{cyc\ K}$	=	32 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 6} = (2 * t_{cyc\ K}) * 2^6 =$	$128 * t_{cyc\ K}$	=	64 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 7} = (2 * t_{cyc\ K}) * 2^7 =$	$256 * t_{cyc\ K}$	=	128 ms	(where $t_{cyc\ K}$ = 0.5 ms)
$t_{cyc\ P\ 0} = (2 * t_{cyc\ K}) * 2^0 =$	$2 * t_{cyc\ K}$	=	4 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 1} = (2 * t_{cyc\ K}) * 2^1 =$	$4 * t_{cyc\ K}$	=	8 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 2} = (2 * t_{cyc\ K}) * 2^2 =$	$8 * t_{cyc\ K}$	=	16 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 3} = (2 * t_{cyc\ K}) * 2^3 =$	$16 * t_{cyc\ K}$	=	32 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 4} = (2 * t_{cyc\ K}) * 2^4 =$	$32 * t_{cyc\ K}$	=	64 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 5} = (2 * t_{cyc\ K}) * 2^5 =$	$64 * t_{cyc\ K}$	=	128 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 6} = (2 * t_{cyc\ K}) * 2^6 =$	$128 * t_{cyc\ K}$	=	256 ms	(where $t_{cyc\ K}$ = 2 ms)
$t_{cyc\ P\ 7} = (2 * t_{cyc\ K}) * 2^7 =$	$256 * t_{cyc\ K}$	=	512 ms	(where $t_{cyc\ K}$ = 2 ms)

Two reference values and four actual values in the same cycle:

$t_{cyc\ P\ 0} = t_{cyc\ P\ 1} = t_{cyc\ P\ 2} = t_{cyc\ P\ 3} = t_{cyc\ K}$

Input us_MODE:

Using us_MODE = 0, you state that FB BAPS_PD_COMM8 is called in a cyclical main program or in an event task to any event (e.g. to the SYNC signal network (CANsync) event).

Using us_MODE = 1 you state that FB BAPS_PD_COMM8 is called in an event task to the „BAPS process data“ event (see description of FB BAPS_INIT, input i_EVENT).

If us_MODE is not assigned, this yields the presetting us_MODE = 0 (use of FB BAPS_PD_COMM8 in the cyclical main program or in an event task to any event).

Inputs ud_WR_VALUE0 to ud_WR_VALUE7:

The reference values are connected to inputs ud_WR_VALUE0 to ud_WR_VALUE7 whose parameter numbers were stated at FB BAPS_INIT (inputs u_WR_PAR_NR0 to u_WR_PAR_NR7) at initialization of BAPS process data communication.

Input x_EN:

Communication is enabled by means of x_EN = TRUE. The default setting is x_EN = TRUE, i.e. communication is enabled.

Output w_STATUSWORD:

At output w_STATUSWORD, the system outputs the V-controller's status word.

Output w_STATUS_REG:

With bit 0 set, output w_STATUS_REG (BAPS status register) indicates that the V-controller is synchronized to the Trigger Controller signal (See “The Interrupt Sources and Trigger Signals” on page 52.)

Outputs ud_RD_VALUE0 to ud_RD_VALUE7:

The actual values are output at outputs ud_RD_VALUE0 to ud_RD_VALUE7 whose parameter numbers were stated at FB BAPS_INIT (inputs u_RD_PAR_NR0 to u_RD_PAR_NR7) at initialization of BAPS process data communication.

Output b_SL_QUIT:

The system reports at output b_SL_QUIT the V-controller acknowledgement after communication has been completed.

Value b_SL_QUIT	**Meaning**
16#00	No meaning
16#01	Reference value has been read/actual value has been written
16#02	Configuration/initialization has been carried out correctly
16#03 – 16#7F	Reserved
16#80	An uninterpretable command has been received
16#81	Configuration/initialization has not been carried out
16#82	Actual value cannot be read
16#83	Reference value cannot be written
16#84 – 16#FE	Reserved
16#FF	No meaning

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte: b_ERR

Bit number	**Error**
0	Output b_SL_QUIT ≠ 16#01
1 – 7	Reserved

6.2.7 BAPS_PD_CONTROL

Description

You can use this function block for BAPS to check the call of process data communication.

The system monitors the correct sequence of process data communication in FB BAPS_PD_COMM2, FB BAPS_PD_COMM24 or FB BAPS_PD_COMM8 (referred to from now on as BAPS_PD_COMMxx).

Parameter input	Data type	Description
x_RESET	BOOL	Reset
t_TIME	TIME	Monitoring time

Parameter output	Data type	Description
x_ERR	BOOL	Error bit

On the BAPS interface, FB BAPS_PD_COMMxx changes a specific (timeout) register at every call of BAPS process data communication. Monitoring the change in this register makes it possible to monitor the call of BAPS process data communication.

The change in this register is not dependent on the result of communication!

Input x_RESET:

You can use x_RESET = TRUE to reset BAPS_PD_CONTROL.

Input t_TIME:

At input t_TIME, you set the monitoring time. If input t_TIME is not assigned, this yields a presetting of 3 s.

Output x_ERR:

x_ERR = TRUE indicates that the timeout register of the BAPS interface that was mentioned above has not been changed again within the monitoring time (t_TIME). This can be due to FB BAPS_PD_COMMxx not having been called or to FB BAPS_PD_COMMxx not having triggered the event task.

NOTE

Output x_ERR is TRUE if no process data communication via the BAPS was carried out within t_TIME.
FB BAPS_PD_CONTROL is not used in an event task to the BAPS process data event (see description of FB BAPS_INIT, FB BAPS_PD_COMMxx).

6.2.8 BAPS_SD_CONTROL

Description

You can use this function block for BAPS to read or write a requirements data value (parameter) of the V-controller via the BAPS interface.

Data is exchanged and FB BAPS_SD_CONTROL is controlled via a structure that FBs BAPS_PAR_READ and/or BAPS_PAR_WRITE read and write to.

NOTE

FB BAPS_SD_CONTROL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BAPS_SD_DATA	BAPS_BMSTRUCT	Communications data
x_RESET	BOOL	Reset
t_TIME	TIME	Monitoring time

Parameter output	Data type	Description
_BAPS_SD_DATA	BAPS_BMSTRUCT	Communications data
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

FB BAPS_SD_CONTROL carries out requirements data communication via the BAPS interface. The jobs for communication are received by FB BAPS_SD_CONTROL from FBs BAPS_PAR_READ and/or BAPS_PAR_WRITE.

The system passes on FB BAPS_PAR_READ's read parameter job to the V-controller and returns the data that the V-controller returns to FB BAPS_PAR_READ.

The system passes on FB BAPS_PAR_WRITE's write parameter job to the V-controller and returns the result of communication that the V-controller returns to FB BAPS_PAR_WRITE.

The system displays at the error outputs errors in communication via the BAPS interface as well as in data exchange with FBs BAPS_PAR_READ and BAPS_PAR_WRITE.

Input/output _BAPS_SD_DATA:

At _BAPS_SD_DATA, you must connect a global variable of data type BAPS_BMSTRUCT.

Example:

```
_BAPS_SD_DATEN : BAPS_BMSTRUCT;
```

Where:

`_BAPS_SD_DATEN`	is the variable name with the data type short designation "_" for STRUCT
`BAPS_BMSTRUCT`	is the data type

Data is exchanged via this variable with FBs BAPS_PAR_READ and BAPS_PAR_WRITE. This variable is connected with the FBs of requirements data communication of the BAPS – this also applies if you use FBs BAPS_PAR_READ and/or BAPS_PAR_WRITE several times.

Input x_RESET:

You can use x_RESET = TRUE to reset BAPS_SD_CONTROL. This is necessary if there were faults in communication via the BAPS interface or data exchange with FBs BAPS_PAR_READ and BAPS_PAR_WRITE. FBs BAPS_PAR_READ and/or BAPS_PAR_WRITE must also be reset so that data exchange with these FBs can be restarted.

Input t_TIME:

At input t_TIME, you set the monitoring time. If input t_TIME is not assigned, this yields a presetting of 3 s.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit Number	Error
0, 1	Data exchange with FBs BAPS_PAR_READ / BAPS_PAR_WRITE is disturbed
2	BAPS interface: Timeout, master or slave semaphore is not equal to 0, communication start not possible
3	BAPS interface: Timeout, master or slave semaphore is not equal to 1, communication interrupted
4	BAPS interface: Timeout, master or slave semaphore is not equal to 0, communication not completed
5	BAPS interface: General operating system or communications error
6	BAPS interface: format error
7	Reserved

NOTE

The Ωmega Drive-Line II writes the master semaphore and the V-controller writes the slave semaphore.

7 CANSYNC

NOTE

The function blocks that are mentioned in this chapter are located in libraries SYSTEM1_DLII_20bd00 (or above), SYSTEM2_DLII_20bd00 (or above) and CANsync_DLII_20db00 (or above).
The data types that are mentioned in this chapter are defined in library BM_TYPES_20bd00 (or above) .
To program the CANsync under PROPROG wt II, you integrate these libraries into a project.

7.1 General

7.1.1 Overview

The CANsync field bus was developed by Baumüller Nürnberg GmbH. The aim of replacing mechanical line shafts by an electronic leading axle was achieved by making available to all the connected drives (⇒ CANsync slaves) the leading axle value at the same instant (time-synchronous transfer).

The CAN bus represents the physical basis. The bus was enhanced by adding a synchronization signal (SYNC signal). The SYNC signal is transferred on two additional wires in the CAN cable. The SYNC signal is for hardware synchronizing the CANsync master with all the CANsync slaves that are located on the CANsync bus. By contrast with the CAN bus, this makes it possible to send and receive message frames at defined instants. The system achieves a guaranteed, high data throughput rate, which, in addition, has a fixed time reference on the CANsync bus.

The CANsync bus is a master-slave bus with one CANsync master and up to 32 CANsync slaves. To differentiate the CANsync slaves each one is assigned a slave number. You specify the slave number by setting DIP switches (See “Setting the Slave Number” on page 20.).

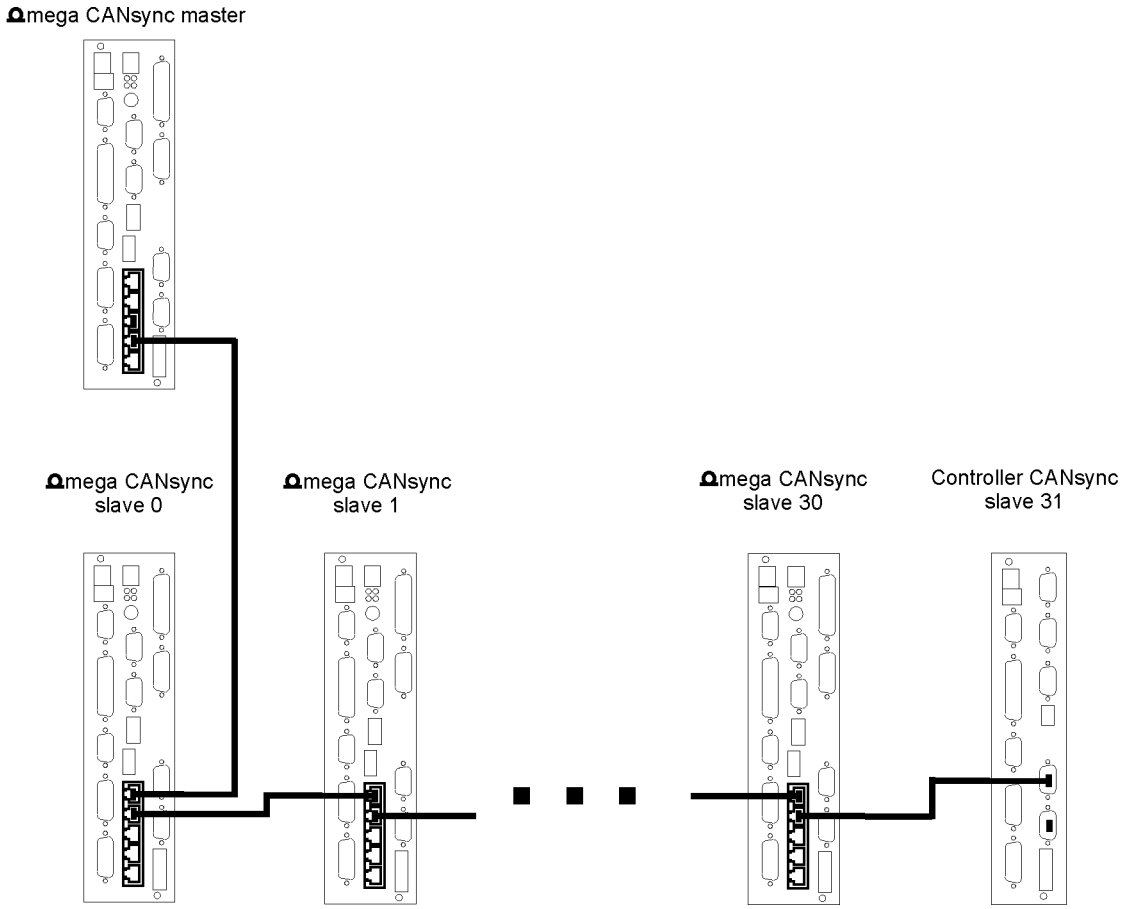


Figure 6-1: CANsync bus, CANsync master with CANsync slaves 0, 1, .., 30, 31

Clustering was implemented to extend the CANsync bus to more than 32 CANsync slaves. Clustering means that you can integrate new CANsync networks starting out from a CANsync slave on the CANsync bus. This represents the CANsync master for the cluster. This also makes it possible to implement following axles that are themselves used as the leading axle for the drives located in the cluster.

The SYNC signal is available for synchronization to every node of the CANsync bus, including the ones in the clusters; each node knows the instant of application of specific message frames!

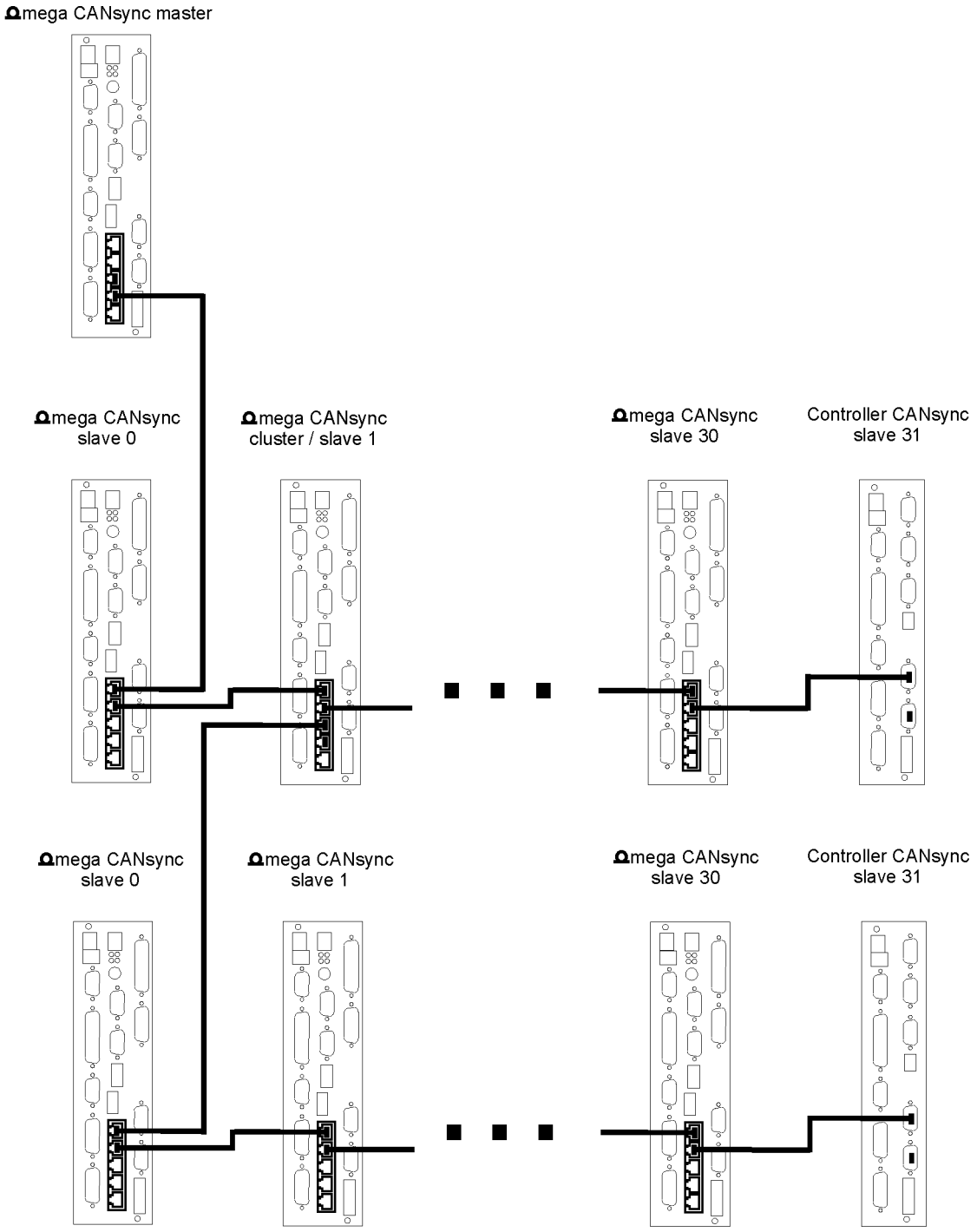


Figure 6-2: Example of the structure of a CANsync cluster.

The theoretical maximum number of nodes is 32^{16}. The number of nodes is, however, limited to 65535, since you must assign a CANsync number (a unique identification of the node for the CANsync bus) in addition to the slave number that you set using DIP switches (See “Setting the Slave Number” on page 20.).

The CANsync synchronization signal (SYNC signal)

- is a specified hardware signal,
- is generated on the CANsync master's CANsync interface module,

- is transferred via two additional lines on the CANsync bus,
- is also transferred to the clusters.

Depending on the operating mode and the transmission speed (baud rate) on the CANsync bus, the system generates the SYNC signal in a specific raster of the CANsync cycle time (the time between two falling edges of the SYNC signal).

Baud rate	CANsync cycle time
500 kbps	2 ms
250 kbps	4 ms
125 kbps	8 ms

The message frame traffic of the CANsync bus is carried out in a specified sequence such that individual message frames are always in defined time windows, which are also known as channels. The following channels are defined in CANsync and are sent by the CANsync master:

- Reference value message frames; WRC1 and WRC2 (**Wr**ite**C**hannel)
- Broadcast message frames; CC (**C**ommand**C**hannel)
- Parameter message frames, CC
- Upload/download message frames, CC

The CANsync slaves must adapt their responses to the CANsync master's time scheme; the following channels are available to them:

- Actual value message frames, RDC1 and RDC 2 (**R**ea**dC**hannel)
- parameter response message frames, RC (**R**esponse**C**hannel)
- Upload/download response message frames, RC

NOTE

We will also refer from now on to WRC1 and WRC2 as reference value channel 1 and reference value channel 2.
We will also refer from now on to RDC1 and RDC2 as actual value channel 1 and actual value channel 2.
We will also refer to CC from now on as command channel.
We will also refer to RC from now on as response channel.

The following example shows the time scheme of the respective message frames with a transmission speed of 500 kbps (CANsync interval with a CANsync cycle time of 2 ms):

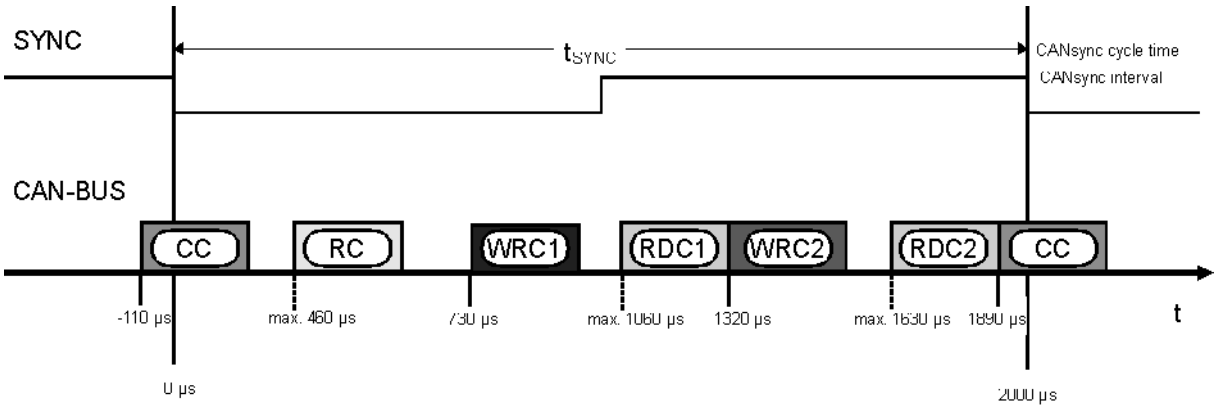


Figure 6-3: Time scheme of the CANsync interval with a transmission speed of 500 kbps

The instants for the channels are listed in the table below in dependence on the baud rate:

Table: Assignment of transmission speed (baud rate), CANsync cycle time t_{SYNC} in ms and maximum bus length in m. The quoted times are maximum values.

Baud rate	CANsync cycle time in µs t_{SYNC}	t_{WRC1}	t_{RDC1}	t_{WRC2}	t_{RDC2}	t_{CC}	t_{RC}	Maximum CANsync bus length
500 kbps	2000	730	1060	1320	1630	1890	460	134 m
250 kbps	4000	1430	2100	2620	3290	3810	900	300 m
125 kbps	8000	3020	4310	5350	6640	7680	1920	600 m

Sequence in principle of communication

Process data communication

Reference value channels WRC1 and WRC2 (WriteChannel), actual value channels RDC1 and RDC2 (ReadChannel) and command channel CC (CommandChannel) are used for process data communication. In every CANsync cycle, the system sends them in the specified sequence.

The master sends the reference value message frames on channels WRC1 and WRC2. In each case, they have a defined length of useful data of 64 bits ≙ 8 bytes ≙ 4 words!

All the slaves/nodes receive them with each slave deciding on its own initiative and on the basis of its set configuration (→ Mapping) the data that represents reference values for it. You make this setting for the V-controller slaves in the supplementary board parameters (see the user guide of the CANsync-Interface option board), for the Ωmega slaves in the mapping to be programmed.

The system adds to the reference value message frame an additional piece of information indicating which slave in the same CANsync cycle is to send its prepared actual value message frame back to the master. This means that it is possible to receive the actual value message frames of a maximum of two slaves per CANsync cycle, since only two actual value message frames are available (in channels RDC1 and RDC2). These message frames also each have useful data amounting to 64 bits ≙ 8 bytes ≙ 4 words.

You can also program automatic polling of all the slaves in sequence.

Usually, the system only polls one CANsync slave per CANsync cycle for its actual values, which means that the actual value message frames on channels RDC1 and RDC2 come from one slave.

NOTE

With a maximum bus configuration with 32 slaves (with no additional clusters!), you need at least 16 CANsync cycles to have available all the current actual values in the master, if two slaves per CANsync cycle return their actual values to the master!

With the V-controller, the process data includes the control word and the status word. Whereas the system enters the status word like any other ordinary actual value in actual value message frames 1 or 2 in dependence on the supplementary board parameter setting, it must treat the control word separately. It is defined as a broadcast command that is sent in the CC.

Mapping

In the following section, we will describe the principle of mapping using the CANsync master interface module as an example. Mapping is carried out in a similar way for the CANsync slave interface module (see also FBs CANsync_PD_CFG_SL or CANsync_PD_CFG_READ_SL). For V-controller slaves, mapping is carried out in the supplementary board parameters (see the user guide of the CANsync-Interface option board).

64 bits are available in each case as the useful data for reference value message frames 1 and 2. This yields the following options for reference value setting by the master:

4 word reference values	(16 bits each)
2 doubleword reference values	(32 bits each)
1 doubleword reference value	(32 bits) and 2 word reference values (16 bits each)

The system enters the reference values in a field (array) that is connected at FB CANsync_PD_COMM_MA at process data communication. It consists of eight 32 bit entries:

Reference values for reference value message frame 1	Reference value 0 highword	Reference value 0 lowword
	Reference value 1 highword	Reference value 1 lowword
	Reference value 2 highword	Reference value 2 lowword
	Reference value 3 highword	Reference value 3 lowword
Reference values for reference value message frame 2	Reference value 4 highword	Reference value 4 lowword
	Reference value 5 highword	Reference value 5 lowword
	Reference value 6 highword	Reference value 6 lowword
	Reference value 7 highword	Reference value 7 lowword

These reference values are buffered in the CANsync interface module's communication RAM as doublewords. This makes it necessary to tell the CANsync interface module which reference values and

which highword or lowword are to be used for the reference value message frame. You must make this setting at function block CANsync_PD_CFG_MA. Since this setting does not generally change any more, users must make it during initialization of the CANsync bus.

NOTE

Doubleword parameters in the V-controller must be transferred in the sequence:
First lowword, second highword (like the following examples).

Example:

You want to write a doubleword reference value as reference value 0 in reference value message frame 1 (in WRC1) and, in addition, two word reference values as reference value 2 and reference value 3:

Setting to make: a_WRC1 = [0, 0, 2, 3]
a_HL_WRC1 = [FALSE, TRUE, FALSE, FALSE]

Reference value channel WRC1 with reference value message frame 1	Reference value 0 highword	Reference value 0 lowword
	----	----
	----	Reference value 2 lowword
	----	Reference value 3 lowword

The arrays that are used are connected to the inputs of FB CANsync_PD_CFG_MA. Array a_WRC1 defines the positions of the reference values to be transferred, doubleword to reference value 0, word reference values to reference values 2 and 3. Array a_HL_WRC1 tells the system whether it is to take the highword (TRUE) or the lowword (FALSE) from communication RAM.

64 bits of useful data are also available for actual value message frames 1 and 2. This means that it is possible to transfer:

4 word actual values	(16 bits each)
2 doubleword actual values	(32 bits each)
1 doubleword actual value	(32 bits) and 2 word actual values (16 bits each)

The system also saves these actual values in communication RAM as doublewords, which means that, here too, you must make an assignment.

Actual values from actual value message frame 1	Actual value 0 highword	Actual value 0 lowword
	Actual value 1 highword	Actual value 1 lowword
	Actual value 2 highword	Actual value 2 lowword
	Actual value 3 highword	Actual value 3 lowword

Actual values from actual value message frame 2	Actual value 4 highword	Actual value 4 lowword
	Actual value 5 highword	Actual value 5 lowword
	Actual value 6 highword	Actual value 6 lowword
	Actual value 7 highword	Actual value 7 lowword

Example:

You want to read from actual value message frame 1 (in RDC1) a doubleword actual value as actual value 0 (position in the message frame words 0 and 1) and to read a second doubleword actual value (words 2 and 3) as actual value 2:

Setting to make:

```
a_RDC1    = [     0,     0,     2,     2]
a_HL_RDC1 = [FALSE,  TRUE, FALSE,  TRUE]
```

Actual value channel RDC1 with actual value message frame 1	Actual value 0 highword	Actual value 0 lowword
	----	----
	Actual value 2 highword	Actual value 2 lowword
	----	----

The arrays that are used are connected to the inputs of FB CANsync_PD_CFG_READ_MA. Array a_RDC1 defines the positions of the actual values to be transferred, doubleword to actual value 0, doubleword to actual value 2. Array a_HL_RDC1 tells the system whether it is to write the highword (TRUE) or the lowword (FALSE) to communication RAM.

In the CANsync master, the system writes the actual values of the individual CANsync slaves to different areas of communication RAM, with each CANsync slave being assigned its own area. This field comprises 32 (since this is the maximum number of nodes without clusters) • 8 (since there are 8 actual values) entries that each contain 32 bits (since they are doubleword actual values) of data.

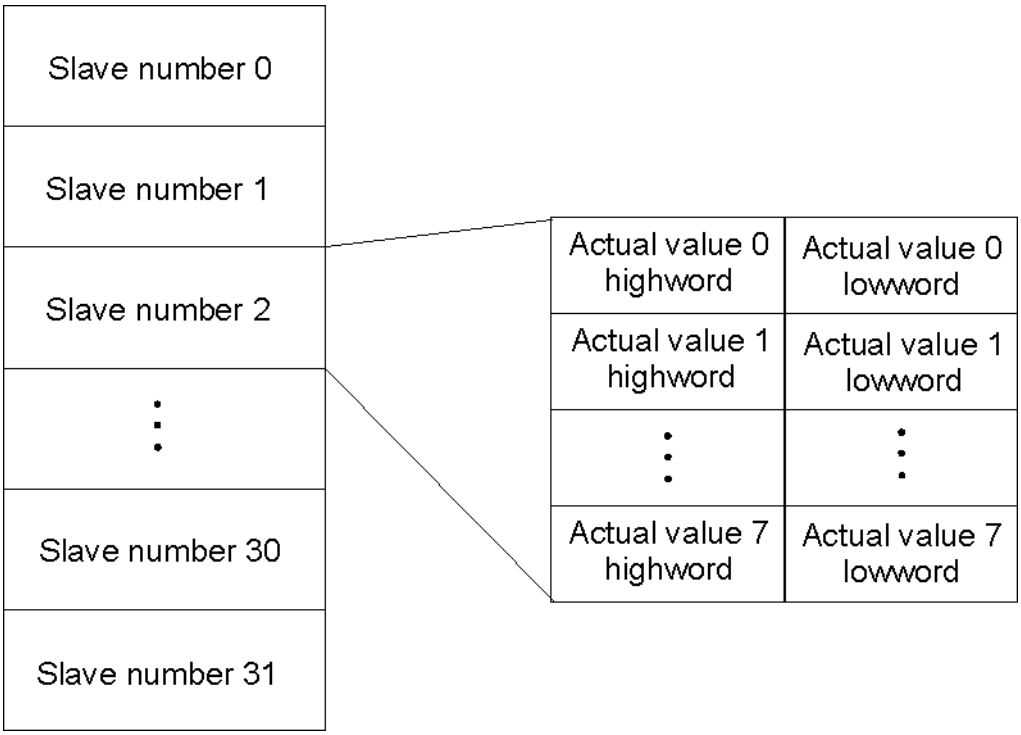


Figure 6-4: Entering actual values in the CANsync master's of the CANsync slaves communication RAM, example for Slave 2

Since only one CANsync slave sends its actual values and possibly its status word in every CANsync actual value message frame, all the other CANsync slaves can monitor these message frames. As a result, it is possible for these actual values to be used as reference values for one or more other CANsync slaves. If this is what you want, you only need to make the setting in the mapping.

Summary of process data communication

In every CANsync interval, the CANsync master sends reference value message frames 1 and 2 and (generally) uses the reference value message frames to request actual value message frames 1 and 2 from a CANsync slave.

All the CANsync slaves receive the CANsync master's reference value message frames and, after a corresponding request, send their actual value message frames to the CANsync master. You make the setting in mapping of which reference values in the reference value message frames are relevant to the CANsync slave. You also define there the combination of actual values that is to be entered in the actual value message frame.

All the CANsync slaves can evaluate the actual value message frames of the other CANsync slaves that the master requests.

You configure the mappings using the following FBs:

a) In the Ωmega CANsync master:

CANsync_PD_CFG_MA (FB is used once)	Mapping of reference value message frames 1 and 2 in the Ωmega CANsync master that are to be sent
CANsync_PD_CFG_READ_MA (FB is used per CANsync slave, a maximum of 32 times)	Mapping of the received actual value message frames 1 and 2 of a CANsync slave

b) In the Ωmega CANsync slave:

CANsync_PD_CFG_SL (FB is used once)	Mapping of the received reference value message frames 1 and 2 and of the actual value message frames 1 and 2 to be sent in the Ωmega CANsync slave
CANsync_PD_CFG_READ_SL (FB is used per further CANsync slave, a maximum of 31 times)	Mapping of the received actual value message frames 1 and 2 of a CANsync slave in the Ωmega CANsync slave

c) In the V-controller CANsync slave:

In the supplementary board parameters (see the user guide of the CANsync-Interface option board)

Requirements Data

In the CANsync, requirements data communication is carried out via the CommandCannel (CC) and the ResponseChannel (RC). In this connection, the CANsync master sends message frames on the command channel that trigger actions in one or more CANsync slaves.

Several message frames are available:

- broadcast message frames
- control word message frames (special case of a broadcast message frame)
- Parameter message frames
- Upload/download message frames

The CANsync slave sends its response on the response channel (RC); it can be

- parameter response message frames
- Upload/download response message frames.

NOTE

The CANsync master can send command message frames in every CANsync interval. The CANsync slaves only respond following a request by the CANsync master.

The system processes the various message frames on the command channel in a priority-based sequence:

Table: Priority specification of the message frames on the command channel (CC)

Message frame type	Priority
Broadcast message frame 0	Highest
Broadcast message frame 1	↑
Broadcast message frame 2	|
Control word message frames	|
Parameter message frames	↓
Upload/download message frames	Lowest

As a result of these priorities, you cannot send another message frame if a higher-priority one is being transmitted. If you send the control word message frame in every CANsync interval, for example, you can never transmit a parameter message frame or an upload/download message frame!

In each CANsync interval, it is only possible to send one of the following broadcast message frames:

a broadcast message frame to **all the** CANsync slaves or
a control word message frame to **one** CANsync slave or
a parameter message frame to **one** CANsync slave or
an upload/download message frame to **one** CANsync slave.

Assuming that there is no broadcast message frame to send, the system sends control word, parameter or upload/download message frames to one CANsync slave in every CANsync interval.

In this connection, you can set in the CANsync master whether a message frame is to be sent to one specific CANsync slave or if it is to be sent automatically to all the CANsync slaves in succession.

Since the maximum number of CANsync slaves (32 without a cluster) do not necessarily need to be present, it is possible to tell the CANsync master the maximum slave number for sending the control word, parameter, upload/download message frame and for requesting the actual value message frame. It is made at function block CANsync_COMM_CONTROL_MA or CANsync_PD_COMM_MA.

7.1.2 Information on Programming

There are two CANsync interface modules (CANsync nodes 1 and 2) on the Ωmega Drive-Line II. They are used to connect a CANsync bus to a CANsync bus of a sublevel. This forms a network with several levels.

Initialization:

If you use the Ωmega Drive-Line II as a CANsync master, you must initialize the CANsync interface module on CANsync node 2 (hardware address: %MB3.200000) as the CANsync master.

After this, active operation on the CANsync master interface module is enabled.

Necessary FBs and their sequences at initialization as a CANsync master:

CANsync_SL_TYP_INIT

CANsync_INIT

CANsync_PD_CFG_MA

CANsync_PD_CFG_READ_MA (one per CANsync slave)

OPT_INIT

INTR_SET

CANsync_MODE_MA

If you use the Ωmega Drive-Line II as a CANsync slave, you must initialize the CANsync interface module on CANsync node 1 (hardware address: %MB3.100000) as the CANsync slave.

After this, active operation on the CANsync slave interface module is enabled.

Necessary FBs and their sequences at initialization as a CANsync slave:

CANsync_INIT

CANsync_PD_CFG_SL

CANsync_PD_CFG_READ_SL (one per further CANsync slave)

OPT_INIT

INTR_SET

CANsync_MODE_SL

If you use the Ωmega Drive-Line II as a CANsync cluster, you must first initialize the CANsync interface module on CANsync node 1 (hardware address: %MB3.100000) as the CANsync slave.

After this, the CANsync interface module on CANsync node 2 (hardware address: %MB3.200000) is initialized as the CANsync master .

After this, active operation on the CANsync master interface module is enabled.

Necessary FBs and their sequences at initialization as a CANsync cluster:

CANsync_INIT

CANsync_PD_CFG_SL

CANsync_PD_CFG_READ_SL (one per further CANsync slave)

CANsync_SL_TYP_INIT

CANsync_INIT

CANsync_PD_CFG_MA

CANsync_PD_CFG_READ_MA (one per CANsync slave)

OPT_INIT

INTR_SET

CANsync_MODE_MA

Information on process data communication

The FBs of CANsync process data communication are placed in a POU that is assigned to the CANsync event task. The FBs of process data communication must be called immediately after calling of the event task so that reference values and/or actual values can be sent and/or received in the same call of the event task.

If the Ωmega Drive-Line II is used as a CANsync master, the following FB must be called:

CANsync_PD_COMM_MA

If the Ωmega Drive-Line II is used as a CANsync slave, the following FB must be called:

CANsync_PD_COMM_SL

If the Ωmega Drive-Line II is used as a CANsync cluster, the FBs must be called in this sequence:

CANsync_PD_COMM_SL

CANsync_PD_COMM_MA

7.2 Detailed Information on CANsync

NOTE

This chapter contains detailed information on CANsync. You do not need this information if you use the function blocks of library CANsync_DLII_20bd00 or above for programming.

t_{RSPTO} - Response Timeout: The time within which the CANsync slave must send a response during initialization.

Baud rate	t_{RSPTO}
500 kbps	600 µs
250 kbps	1100 µs
125 kbps	2100 µs

NOTE

All the following timings are relative to a baud rate of 500 kbps.

Start-up and initialization

Starting characteristics are divided into the following steps:

⇨ Initialization with FB CANsync_INIT:

After the initialization with FB CANsync_INIT is done, the CANsync master outputs the SYNC signal with a 2-ms timing code and the "SYNC-Modus“ (SYNC mode) action command containing the data: “SYNC-Betrieb einschalten” (Activate SYNC operation) to all the CANsync slaves. The CANsync slaves start to synchronize their control task to the SYNC signal.

The CANsync master uses the Parameter Lesen (read parameter) parameter command to request the status word from each CANsync slave that it expects on the CANsync bus.

The CANsync slave must respond with its parameter response within $t_{RSPTO.}$ The CANsync master monitors whether the CANsync slave responds within this time.

The CANsync master does not yet output any reference values during synchronization.

⇨ Change to activ mode

The function block CANsync_MODE_MA (input x_CANsync_RUN = TRUE) starts directly the activ mode. At this time the reference values and the actual values will be transferred. Think on that CANsync slaves need some time (some seconds) to synchronize to the SYNC-Signal. The

V-controller status word include the information about the state of synchronizing (status word Bit 15 = TRUE).

If you like to transfer position set values, the application in CANsync master must wait until all CANsync slaves are synchronized (all CANsync slaves have to be in "synchronized" status). Other reference values or other modes of operation are allowed before all CANsync slaves are in "synchronized" status.

Synchronized Status

In synchronized status, all the drives (Ωmega CANsync master, Ωmega CANsync slaves, V-controllers CANsync slaves) in the same CANsync interval, i.e. all the CANsync slaves carry out processing in their control tasks and apply the reference values at the same time.

The CANsync master sends its jobs in a defined sequence with assigned time windows. The CANsync slaves must adapt their responses to this scheme (See “Overview” on page 115. and Figure 6-3).

The reference value that was received in the previous CANsync interval becomes active in the control task of the V-controller 750 µs after the SYNC signal.

The CANsync master must send the next reference value message frame 1 by 730 µs after the SYNC signal at the latest.

Synchronization loss with V-controller CANsync slaves

If the SYNC signal fails in a CANsync interval, the system still runs through the next CANsync interval. In this case, the actual value message frame of the CANsync slaves can be omitted if it cannot be sent with the necessary time precision.

If the SYNC signal fails for a settable time in a row, the CANsync slave is in the non-synchronized status. The system reports a corresponding error in this case. Users can set the response (fast brake, controller inhibit,...) in the V-controller via communication monitoring.

Reference value loss with V-controller CANsync slaves

If no new reference value is received in a CANsync interval, the CANsync slave carries out extrapolation using the reference value that it received last.

In the case of a settable number of reference value dropouts in a row (ZK 26), the system reports a corresponding error (communication monitoring). Users can set the response (fast brake, controller inhibit,...) in the V-controller.

7.2.1 Structure of Message Frames

⇨ CANsync message frames

Message frame lengths are variable (0 to 8 data bytes). They result from the list above and in some cases are also dependent on the respective operating status.

⇨ Data format

The data is stored in the message frames in Intel format (low byte/high byte).

⇨ Status Word

A CANsync slave's status word indicates its drive status. The SYNC status must be displayed in the topmost bit (15). If the bit is set, then the CANsync slave is synchronized.

Reference value channels

Reference value channel 1

On reference value channel 1, the CANsync master sends reference value message frame 1.

With this, the CANsync master transfers to all the CANsync slaves one or more reference values (up to a maximum of four). Number NNNNNNN in the identifier indicates which CANsync slave must send its actual value message frame 1 after reference value message frame 1.

<IDENTIFIER><SOLLWERTE>			
<IDENTIFIER>	::=	0010NNNNNNN	Identifier of reference value channel 1
<SOLLWERTE>	::=	W_SOLL | DW_SOLL | W_DW_SOLL	
<W_SOLL>	::=	<W_SOLL_1> | <W_SOLL_1..2> | <W_SOLL_1..3> | <W_SOLL_1..4>	Word reference values only
<DW_SOLL>	::=	<DW_SOLL_1> | <DW_SOLL_1..2>	DWord reference values only
<W_DW_SOLL>	::=	<DW_SOLL_1><W_SOLL_3> | <DW_SOLL_1><W_SOLL_3><W_SOLL_4>	Word and DWord reference values
<W_SOLL_1>	::=	<Word>	Word reference value 1 = CAN-DB 0..1
<W_SOLL_2>	::=	<Word>	Word reference value 2 = CAN-DB 2..3
<W_SOLL_3>	::=	<Word>	Word reference value 3 = CAN-DB 4..5
<W_SOLL_4>	::=	<Word>	Word reference value 4 = CAN-DB 6..7
<DW_SOLL_1>	::=	<DWord>	DWord reference value 1 = CAN-DB 0..3
<DW_SOLL_2>	::=	<DWord>	DWord reference value 2 = CAN-DB 4..7
			CAN-DB: CAN data byte

Reference value channel 2

On reference value channel 2, the CANsync master sends reference value message frame 2.

With this, the CANsync master transfers up to four additional reference values on this reference value channel.

The structure and function correspond to reference value channel 1. Reference value message frame 2 has a different identifier and assignment of reference values to reference value message frame 1. Word reference values 5...8 and doubleword reference values 3...4 are assigned to reference value channel 2. Number NNNNNNN in the identifier indicates the slave number of the CANsync slaves that responds with actual value message frame 2.

<IDENTIFIER>	::=	0011NNNNNNN	Identifier of reference value channel 2

Actual value channels

Actual value channel 1

On actual value channel 1, the CANsync slave sends actual value message frame 1.

The identifier of reference value message frame 1 indicates which CANsync slave may send its actual value message frame 1 as a direct response to the reference value 1 message frame.

<IDENTIFIER><ISTWERTE>			
<IDENTIFIER>	::=	0110NNNNNNN	NNNNNNN = CANsync slave number
<ISTWERTE>	::=	W_IST | DW_IST | W_DW_IST	
<W_IST>	::=	<W_IST_1> | <W_IST_1..2> | <W_IST_1..3> | <W_IST_1..4>	Word actual values only
<DW_IST>	::=	<DW_IST_1> | <DW_IST_1..2>	DWord actual values only
<W_DW_IST>	::=	<DW_IST_1><W_IST_3> | <DW_IST_1><W_IST_3><W_IST_4>	Word and DWord actual values
<W_IST_1>	::=	<Word>	Word actual value 1 = CAN-DB 0..1
<W_IST_2>	::=	<Word>	Word actual value 2 = CAN-DB 2..3
<W_IST_3>	::=	<Word>	Word actual value 3 = CAN-DB 4..5
<W_IST_4>	::=	<Word>	Word actual value 4 = CAN-DB 6..7
<DW_IST_1>	::=	<DWord>	DWord actual value 1 = CAN-DB 0..3
<DW_IST_2>	::=	<DWord>	DWord actual value 2 = CAN-DB 4..7
			CAN-DB: CAN data byte

Actual value channel 2

On actual value channel 2, the CANsync slave sends actual value message frame 2.

The structure and the function of the message frame correspond to actual value message frame 1. A maximum of four further actual values can be transferred. Actual value message frame 2 has a different identifier and assignment of the actual values. Word actual values 5...8 and doubleword actual values 3...4 are assigned to actual value channel 2.

<IDENTIFIER>	::=	0111NNNNNNN	NNNNNNN = CANsync slave number

Command Channel & Response Channel

The command channel and the associated response channel consist functionally of three groups of message frames: in this connection, only one command/response can ever occur in any one CANsync interval.

- ⇨ Action commands are for initializing and controlling the CANsync slaves and are sent to one or more CANsync slaves without the CANsync master expecting a response.
- ⇨ Parameter commands are used for reading or writing a parameter and are always directed to one CANsync slave. The master always expects a response.
- ⇨ Upload/download commands are for transferring large volumes of data (program code, data records) and are always directed to one CANsync slave. The master always expects a response.

Parameter and upload/download commands are sent with the same identifier.

Action command

The CANsync master sends an action command to a single CANsync slave or to a group of CANsync slaves. The system makes the choice by means of a bit strip (SLAVE_GROUP) in which one bit is assigned to each CANsync slave. When this bit is set, the associated CANsync slave must carry out this command. In a broadcast command to all the CANsync slaves, all the bits in the bit strip are set.

The various commands are differentiated by the COMMAND data byte. Depending on the command, there follow different numbers of data bytes that contain data relating to the command.

<IDENTIFIER><SLAVE_GROUP><COMAND><DATA>			
<IDENTIFIER>	::=	00000010000	Identifier of action command
<SLAVE_GROUP>	::=	<DWord>	Slave bits 0..30 = CAN-DB 0..3
<COMMAND>	::=	<Byte> 1 = Write control word	= CAN-DB 4
<DATA>	::=	<DATA_1> | <DATA_2> | <DATA_3> | <DATA_4> | <DATA_5> | <DATA_6> | <DATA_7> | <DATA_8> | <DATA_9> | <DATA_10>	Data dependent on the command
<DATA_1>	::=	<res><Wert>	
<res>	::=	<Byte>	CAN-DB 5
<Wert>	::=	<Word>	Control word = CAN-DB 6..7
			CAN-DB: CAN data byte

Read parameter

The CANsync master uses the Parameter Lesen (read parameter) command to request a parameter of the CANsync slave for reading. The message frame length (of 4 data bytes) tells the CANsync slave that this is a read parameter command. The CANsync slave must not necessarily support the element selection: in this case it always responds with the current data value.

The CANsync slave must respond within the reference response time, $t_{RSPTO.}$ If it cannot finish the job by then, it responds with the parameter response in which the job's parameter number is entered and the BUSY bit is set. The next time that the CANsync master repeats the read parameter command to the parameter and the CANsync slave has processed the job in the meantime, it responds with the requested data and the BUSY bit is set to zero.

If the read job cannot be processed or an error occurs, the CANsync slave sets the ERR bit and states an error code in the data bytes.

Parameter numbers can be between 0 and 4095.

Job:

<IDENTIFIER><CONTROL><PARA_NUM_L><SUB-ADRESSE>

<IDENTIFIER>	::=	1010NNNNNNN	NNNNNNN = CANsync slave number
<CONTROL>	::=	<P><ELEMENT><PARA_NUM_H>	CAN-DB 0
<P>	::=	<Bit7>	0 = Identifier: Parameter command
<ELEMENT>	::=	<Bit6..4>	Element selection of the parameter
<PARA_NUM_H>	::=	<Bit3..0>	Bits 11..8 of the parameter number
<PARA_NUM_L>	::=	<Byte>	Bits 7..0 of parameter no. = CAN-DB 1
<SUB-ADRESSE>	::=	<Word>	Sub-slave address = CAN-DB 2..3

Response:

<IDENTIFIER><STATUS><PARA_NUM_L><DATA><SUB-ADRESSE>

<IDENTIFIER>	::=	1011NNNNNNN	NNNNNNN = CANsync slave number
<STATUS>	::=	<P><BUSY><ERR> <FREI><PARA_NUM_H>	CAN-DB 0
<P>	::=	<Bit7>	0 = Identifier: Parameter response
<BUSY>	::=	<Bit6>	0 = Response valid, 1 = Job being processed
<ERR>	::=	<Bit5>	0 = No error, 1 = Error
<FREI>	::=	<Bit4>	Free
<PARA_NUM_H>	::=	<Bit3..0>	Bits 11..8 of the parameter number
<PARA_NUM_L>	::=	<Byte>	Bits 7..0 of parameter no. = CAN-DB 1
<DATA>	::=	<2_BYTE> | <4_BYTE> | <ERR_CODE>	
<2_BYTE>	::=	<Word>	Word parameter = CAN-DB 2..3
<4_BYTE>	::=	<Dword>	DWord parameter = CAN-DB 2..5
<ERR_CODE>	::=	<Word>	2-byte error code = CAN-DB 2..3
<SUB-ADRESSE>	::=	<Word>	Sub-slave address = CAN-DB 4..5 / 6..7 CAN-DB: CAN data byte

Write parameter

The CANsync master uses the Parameter Schreiben (write parameter) command to write a parameter to a CANsync slave. The message frame length (6 or 8 data bytes) tells the CANsync slave whether the parameter in question is a word or doubleword parameter. Currently, when writing only element 7, the parameter value, is permissible.

The CANsync slave must respond within the reference response time, $t_{RSPTO.}$ If it cannot finish the job by then, it responds with the parameter response in which the job's parameter number is entered and the BUSY bit is set. The next time that the CANsync master repeats the write parameter command to the parameter and the CANsync slave has processed the job in the meantime, it responds with the parameter number and the BUSY bit is set to zero.

If the write job cannot be processed or an error occurs, the CANsync slave sets the ERR bit and states an error code in the data bytes.

Parameter numbers can be between 0 and 4095.

Job:

<IDENTIFIER><CONTROL><PARA_NUM_L><DATA><SUB-ADRESSE>

<IDENTIFIER>	::=	1010NNNNNNN	NNNNNNN = CANsync slave number
<CONTROL>	::=	<P><ELEMENT><PARA_NUM_H>	CAN-DB 0
<P>	::=	<Bit7>	0 = Identifier: Parameter command
<ELEMENT>	::=	<Bit6..4>	Element selection of the parameter
<PARA_NUM_H>	::=	<Bit3..0>	Bits 11..8 of the parameter number
<PARA_NUM_L>	::=	<Byte>	Bits 7..0 of parameter no. = CAN-DB 1
<DATA>	::=	<2_BYTE> | <4_BYTE>	
<2_BYTE>	::=	<Word>	Word parameter = CAN-DB 2..3
<4_BYTE>	::=	<Dword>	DWord parameter = CAN-DB 2..5
<SUB-ADRESSE>	::=	<Word>	Sub-slave address = CAN-DB 4..5 / 6..7

Response:

<IDENTIFIER><STATUS><PARA_NUM_L><DATA><SUB-ADRESSE>

<IDENTIFIER>	::=	1011NNNNNNN	NNNNNNN = CANsync slave number
<STATUS>	::=	<P><BUSY><ERR> <FREI> <PARA_NUM_H>	CAN-DB 0
<P>	::=	<Bit7>	0 = Identifier: Parameter response
<BUSY>	::=	<Bit6>	0 = Job finished, 1 = Job being processed
<ERR>	::=	<Bit5>	0 = No error, 1 = Error
<FREI>	::=	<Bit4>	Not yet assigned
<PARA_NUM_H>	::=	<Bit3..0>	Bits 11..8 of the parameter number
<PARA_NUM_L>	::=	<Byte>	Bits 7..0 of parameter no. = CAN-DB 1
<DATA>	::=	<0_BYTE> | <ERR_CODE>	
<0_BYTE>	::=	No data bytes if error-free	
<ERR_CODE>	::=	<Word>	2-byte error code = CAN-DB 2..3
<SUB-ADRESSE>	::=	<Word>	Sub-slave address = CAN-DB 2..3/4..5 CAN-DB: CAN data byte

Start of an upload or download

Using uploading or downloading, you can transfer relatively large contiguous data ranges from the CANsync master to the CANsync slave or vice versa.

You configure the transfer using an initialization message frame.

The CANsync slave must respond within the reference response time, $t_{RSPTO.}$ If it cannot finish the job by then, it responds with the upload/download response in which the BUSY bit is set. The next time that the CANsync master repeats the upload/download message frame and the CANsync slave has processed the job in the meantime, it responds with the response in which the BUSY bit is set to zero.

If the upload or download job cannot be processed or an error occurs, the CANsync slave sets the ERR bit and states an error code in the data bytes.

The start address is a doubleword address. The maximum length of an upload or download is 4096 bytes. You must transfer larger data ranges by means of several upload/download initializations. As an option, you can also state a sub-slave address. This address indicates that the subsequent upload/download does not refer directly to the addressed CANsync slave, but rather that the upload/download message frames are passed on to a sub-slave. This sub-slave address remains valid until the end of the upload/download. The address must be stated again for the next upload/download job. If the sub-address is equal to zero, the system addresses the CANsync slave directly and not a sub-slave.

Job:

<IDENTIFIER><CONTROL><OFFSET_L><ADRESSE><SUB-ADRESSE>

<IDENTIFIER>	::=	1010NNNNNNN	NNNNNNN = CANsync slave number
<CONTROL>	::=	<L><U/D><MODE><OFFSET_H>	CAN-DB 0
<L>	::=	<Bit7>	1 = Identifier: Upload/download job
<U/D>	::=	<Bit6>	0 = Upload, 1 = Download
<MODE>	::=	<Bit5..4>	01 = Initialization
<OFFSET_H>	::=	<Bit3..0>	Block length in bytes bits 11..8 a)
<OFFSET_L>	::=	<Byte>	Block length in bytes bits 7..0 = CAN-DB 1 a)
<ADRESSE>	::=	<Dword>	Absolute start address = CAN-DB 2..5
<SUB-ADRESSE>	::=	<Word>	Sub-slave address = CAN-DB 6..7

Response:

<IDENTIFIER><STATUS><OFFSET_L><DATA>

<IDENTIFIER>	::=	1011NNNNNNN	NNNNNNN = CANsync slave number
<STATUS>	::=	<L><BUSY><ERR><FREI><OFFSET_H>	CAN-DB 0
<L>	::=	<Bit7>	1 = Identifier: Upload/download response
<BUSY>	::=	<Bit6>	0 = Job finished, 1 = Job being processed
<ERR>	::=	<Bit5>	0 = No error, 1 = Error
<FREI>	::=	<Bit4>	Not yet assigned
<OFFSET_H>	::=	<Bit3..0>	Block length in bytes bits 11..8 a)
<OFFSET_L>	::=	<Byte>	Block length in bytes bits 7..0 = CAN-DB 1 a)
<DATA>	::=	<0_BYTE> | <ERR_CODE>	
<0_BYTE>	::=	No data bytes if error-free	
<ERR_CODE>	::=	<Word>	Bits 15..0 of error code = CAN-DB 2..3 CAN-DB: CAN data byte

a) In the case of an upload job, the CANsync slave can determine the length instead of the CANsync master. In this case, the length is stated in the response in <OFFSET_H> and <OFFSET_L>. Otherwise, these values in the response are zero.

Ongoing Upload and End of an Upload

The upload procedure consists of successive upload message frames in which the CANsync master requests successive data blocks starting from the start address that was set at initialization. The offset address increases consecutively as the byte address. The system transfers 6 bytes of useful data with each message frame. This means that the first message frame starts with offset address 0, the second one requests the data with offset address 6, etc.

The CANsync slave must respond within the reference response time, $t_{RSPTO.}$ If it cannot finish the job by then, it responds with the upload response in which the job's offset address is entered and the BUSY bit is set. The next time that the CANsync master repeats the upload message frame and the CANsync slave has processed the job in the meantime, it responds with the requested data block and the BUSY bit is set to zero.

In the last message frame, MODE is set to 11. The CANsync slave checks whether it has reached the end of the set data range and always sends the last data block with six data bytes. If the memory area to be loaded does not contain as much data, the system pads it with irrelevant data. If the CANsync slave has not reached the end, it responds with a set ERR bit and an error code.

The CANsync slave sets the ERR bit and indicates an error code in the data bytes even if the upload job cannot be processed or if the CANsync slave determines a gap in the requested offset addresses. If necessary, the CANsync master can repeat the failed message frame, or it cancels the upload by setting MODE to 01 and entering 0 as the base address and the block length.

Job:

<IDENTIFIER><CONTROL><OFFSET_L>

<IDENTIFIER>	::=	1010NNNNNNN	NNNNNNN = CANsync slave number
<CONTROL>	::=	<UD><U/D><MODE><OFFSET_H>	CAN-DB 0
<UD>	::=	<Bit7>	1 = Identifier: Upload/download job
<U/D>	::=	<Bit6>	0 = Upload
<MODE>	::=	<Bit5..4>	10 = Body
			11 = Last block
<OFFSET_H>	::=	<Bit3..0>	Bits 11..8 of the offset address
<OFFSET_L>	::=	<Byte>	Bits 7..0 of the offset address = CAN-DB 1

Response:

<IDENTIFIER><STATUS><OFFSET_L><DATA>

<IDENTIFIER>	::=	1011NNNNNNN	NNNNNNN = CANsync slave number
<STATUS>	::=	<UD><BUSY><ERR><FREI><OFFSET_H>	CAN-DB 0
<UD>	::=	<Bit7>	1 = Identifier: Upload/download response
<BUSY>	::=	<Bit6>	0 = Job finished,
			1 = Job being processed
<ERR>	::=	<Bit5>	0 = No error, 1 = Error
<FREI>	::=	<Bit4>	Not yet assigned
<OFFSET_H>	::=	<Bit3..0>	Bits 11..8 of the offset address
<OFFSET_L>	::=	<Byte>	Bits 7..0 of the offset address = CAN-DB 1
<DATA>	::=	<DATEN> | <ERR_CODE>	
<DATEN>	::=	<Word><Word><Word>	6 bytes of data = CAN-DB 2..7
<ERR_CODE>	::=	<Word>	Error code = CAN-DB 2..3
			CAN-DB: CAN data byte

Ongoing Download and End of a Download

The download procedure consists of successive download message frames in which the CANsync master sends successive data blocks starting from the start address that was set at initialization. The offset address increases consecutively as the byte address. This means that the first message frame starts with offset address 0, the second one sends the data with offset address 6, etc.

The CANsync slave must respond within the reference response time, $t_{RSPTO.}$ If it cannot finish the job by then, it responds with the download response in which the job's offset address is entered and the BUSY bit is set. The next time that the CANsync master repeats the download message frame and the CANsync slave has processed the job in the meantime, it responds with the response message frame in which the BUSY bit is set to zero.

In the last message frame, MODE is set to 11 and it also contains six data bytes. However, the CANsync slave may only take the data bytes that correspond to the previously set download length. If the CANsync slave has not yet reached the end, it responds with a set ERR bit and an error code.

The CANsync slave sets the ERR bit and indicates an error code in the data bytes even if the download job cannot be processed or if the CANsync slave determines a gap in the sent offset addresses. If necessary, the CANsync master can repeat the failed message frame, or it cancels the download by setting MODE to 01 and entering 0 as the base address and the block length.

Job:

<IDENTIFIER><CONTROL><OFFSET_L><DATA>

<IDENTIFIER>	::=	1010NNNNNNN	NNNNNNN = CANsync slave number
<CONTROL>	::=	<UD><U/D><MODE><OFFSET_H>	CAN-DB 0
<UD>	::=	<Bit7>	1 = Identifier: Upload/download job
<U/D>	::=	<Bit6>	1 = Download
<MODE>	::=	<Bit5..4>	10 = Body
			11 = Last block
<OFFSET_H>	::=	<Bit3..0>	Bits 11..8 of the offset address
<OFFSET_L>	::=	<Byte>	Bits 7..0 of the offset address = CAN-DB 1
<DATA>	::=	<Word><Word><Word>	6 bytes of useful data = CAN-DB 2..7

Response:

<IDENTIFIER><STATUS><OFFSET_L><DATA>

<IDENTIFIER>	::=	1011NNNNNNN	NNNNNNN = CANsync slave number
<STATUS>	::=	<UD><BUSY><ERR><FREI><ERR_CODE_H>	CAN-DB 0
<UD>	::=	<Bit7>	1 = Identifier: Upload/download
<BUSY>	::=	<Bit6>	0 = Job finished,
			1 = Job being processed
<ERR>	::=	<Bit5>	0 = No error, 1 = Error
<FREI>	::=	<Bit4>	Not yet assigned
<OFFSET_H>	::=	<Bit3..0>	Bits 11..8 of the offset address
<OFFSET_L>	::=	<Byte>	Bits 7..0 of the offset address = CAN-DB 1
<DATA>	::=	<0_BYTE> | <ERR_CODE>	
<0_BYTE>	::=	No data bytes if error-free	
<ERR_CODE>	::=	<Word>	Bits 15..0 of error code = CAN-DB 2..3
			CAN-DB: CAN data byte

7.2.2 Register Structure and Function of the Ωmega CANsync Master

In the following sections, we will explain the register structure of communication RAM in the Omega CANsync master.

To allow you to access registers of the communication RAM in the PROPROG wt II project, data types are defined that map the register structure. The system uses these data types to declare variables that are assigned to the CANsync interface module's base address.

After this, it is possible to access the registers of communication RAM via the structure elements of the declared variables.

At initialization of the CANsync master interface module, the registers in communication RAM have a different meaning than after initialization in cyclical operation.

This means that, for initialization there is the

CANsync_INIT_BMSTRUCT

and for cyclical operation, the

CANsync_MA_CTRL_BMSTRUCT

These structures are defined from library BM_TYPES_20bd00 onwards. After you have integrated library BM_TYPES_20bd00 in the project, the data types are available.

These structures contain

- 8-bit elements,
- 16-bit elements,
- 32-bit elements,
- Structures from the elements mentioned above
- Fields (ARRAY) and structures from the elements and structures mentioned above

Short designations have been prepended to the data types (8-, 16-, 32-bit elements, structures and fields) that are used in a structure. This is for the sake of clarity when using the structures in programming.

Data type	Short designation	Number of bits
BYTE	b	8
WORD	W	16
DWORD (double word)	d	32
SINT (short integer)	Si	8
DINT (double integer)	di	32
USINT (unsigned short integer)	us	8
UINT (unsigned integer)	u	16
UDINT (unsigned double integer)	ud	32
STRUCT	_ (underline)	-
ARRAY	a	-

Other data types that are not used in the structures include:

Data type	Short designation	Number of bits
BOOL (bit)	X	1
TIME	t	-

Explanation of declaring the global variables

For initialization, you create a global variable of data type CANsync_INIT_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) in Ωmega Drive-Line II

```
_CANsync_INIT_MA          AT      %MB3.200000 : CANsync_INIT_BMSTRUCT;
```

Where:

`CANsync_INIT_ MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_INIT_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II.

For cyclical operation, you create a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) in Ωmega Drive-Line II

```
_CANsync_CTRL_MA          AT      %MB3.200000 : CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II.

NOTE

In the following tables, the variable name is replaced by an asterisk (*).

This means that you access register `*.w_CPU_CONTROL` via

`_CANsync_INIT_MA.w_CPU_CONTROL,`

you access `*.w_OPTION_STATUS` via

`_CANsync_INIT_MA.w_OPTION_STATUS.`

Where:

CANsync_INIT_ MA	is the variable name with the data type short designation "_" for STRUCT
w_CPU_CONTROL	is the control register of the CANsync interface module with data type short designation "w" for WORD

Registers *.w_CPU_CONTROL and *.w_OPTION_STATUS can also be triggered via the structure for cyclical operation. This makes possible access via

_CANsync_CTRL_MA.w_CPU_CONTROL and

_CANsync_CTRL_MA.w_OPTION_STATUS.

Where

CANsync_CTRL_MA	is the variable name with the data type short designation "_" for STRUCT
w_CPU_CONTROL	is the status register of the CANsync interface module with data type short designation "w" for WORD

Example of accessing an element of a field that is used in the structure:

According to the table: *.a_WR_VALUE[3]

Access: _CANsync_CTRL_MA.a_WR_VALUE[3]

Where

CANsync_CTRL_MA	is the variable name with the data type short designation "_" for STRUCT
a_WR_VALUE[3]	is the register for reference value 3 with the data type short designation "a" for ARRAY. The data type of the elements of the field (of the reference values) is taken from the corresponding table and the description.

Example of accessing an element of a two-dimensional field that is used in the structure:

According to the table: *.a_RD_VALUE[5][7]

Access: _CANsync_CTRL_MA.a_RD_VALUE[5][7]

Where

CANsync_CTRL_MA	is the variable name with the data type short designation "_" for STRUCT
a_RD_VALUE[5][7]	is the register for actual value 7 of CANsync slave 5 with the data type short designation "a" for ARRAY. The data type of the elements of the field (of the reference values) is taken from the corresponding table and the description.

Example of accessing an element a (sub) structure, which is itself a field that is used in the structure:

According to the table:`*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3`

Access:`_CANsync_CTRL_MA.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3`

Where

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`a_CFG_RDC_WORD[31]`	is the field containing the configuration data for mapping the words of CANsync slave 31's actual value message frames with the data type short designation "a" for ARRAY
`b_CFG_RDC2_WORD3`	is the register for the configuration data for mapping the third word of actual value message frame 2 (of CANsync slave 31) with the data type short designation "b" for BYTE

Example of accessing an element of a (sub) structure that is used in the structure:

According to the table:`*._CFG_WRC_WORD.b_CFG_WRC1_WORD0`

Access: `_CANsync_CTRL_MA._CFG_WRC_WORD.b_CFG_WRC1_WORD0`

Where

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`_CFG_WRC_WORD`	is the structure containing the configuration data for mapping the words of the reference value message frames with the data type short designation "_" for STRUCT
`b_CFG_WRC1_WORD0`	is the register for the configuration data for mapping the 0th word of reference value message frame 1 with the data type short designation "b" for BYTE

General Registers of the CANsync Interface Module

Register	Contents
*.w_CANsync_STATUS	CANsync-Status
*.w_OMEGA_NR	The Ωmega number set using a DIP switch
*.i_SW1_NR	Card software number
*.i_SW1_RELEASE	Software revision incompatible and compatible

CANsync status

Each time the CANsync processor cycle is run through, the system outputs the CANsync status to *.w_CAN_STATUS.

Meaning:

Bit No.	Meaning (bit = TRUE)
0	Reserved
1	Overrun: A CANsync message could not be received
2	CANsync send buffer is free
3	CANsync send job executed successfully
4	CANsync message currently being received
5	CANsync message currently being sent
6	Error present (warning)
7	CANsync node is deactivated (BUS off)
8-15	Reserved

Ωmega number

In register *.w_OMEGA_NR, the system displays the Ωmega number that was set using the DIP switch (S33). In the case of a CANsync master interface module, this number is meaningless.

Software number and software version

In register *.i_SW1_NR, the system displays the number of the CANsync software on the Ωmega Drive-Line II.

In register *.i_SW1_RELEASE, the system displays the compatible and the incompatible revision of the CANsync software on the Ωmega Drive-Line II.

Initialization:

For initialization, you create a global variable of data type CANsync_INIT_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) in Ωmega Drive-Line II

```
_CANsync_INIT_MA        AT     %MB3.200000 : CANsync_INIT_BMSTRUCT;
```

Where:

`CANsync_INIT_ MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_INIT_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II.

In addition, you carry out configuration for synchronous operation:

Meaning		**Register**	**Value**
Baud rate	For example: 500 kbps	*.b_BT_0 *.b_BT_1	16#00 16#1C
CANsync interval	For example: 2 ms	*.b_TIME_PATTERN	16#02
Acceptance Code	All message frames	*.b_AC	16#FF
Acceptance Mask		*.b_AM	16#FF
Output Control	16#FA	*.b_OUTPUT_CONTROL	16#FA
Clock Divider	16#07	*.b_CLOCK_DIVIDER	16#07
Slave/Master	Master	*.b_MA_SL_MODE	16#00
Slave types	CANsync slave with slave number x not available / available	*.a_SL_TYP[x]	16#00 / 16#01

You state the CANsync slave type for each CANsync slave interface module on the CANsync bus (slave number setting using DIP switches). Currently, there is only CANsync slave type 16#01. If the value is set to 16#00, this means that no CANsync slave is present with this slave number or that the CANsync master does not expect one.

You set the operating mode via register *.w_CPU_CONTROL. The system displays the currently active operating mode in register *.w_OPTION_STATUS. You can change the operating mode even after it has been started successfully.

Register	Contents
*.w_CPU_CONTROL	Control register of CANsync interface module
*.w_OPTION_STATUS	Status register of CANsync interface module

(* at initialization, corresponds to _CANsync_INIT_MA, for example; after initialization, corresponds to _CANsync_CTRL_MA) for example

Control register of CANsync interface module	Meaning
16#0000	Cold restart
16#0001	Handshake
16#0002	Take over initialization data
16#0012	Reserved
16#0013	Reserved
16#0020	Start synchronous operating mode
16#0040	Enable active operation
16#0080	(Bit 7 = TRUE) Reset CAN controller

Status register of CANsync interface module	Meaning
16#0001	Start up
16#0002	Take over waiting for initialization data
16#0003	Waiting for start
16#0011	Reserved
16#0012	Reserved
16#0013	*Setting up synchronous operation of CANsync slave*
16#0020	*Synchronous operation of CANsync slave is active*
16#0041	Reserved
16#0042	Reserved
16#0043	Setting up synchronous operation of CANsync master
16#0080	Synchronous operation of CANsync master is active

Initialization is carried out using commands 16#0000, 16#0001 and 16#0002 to *.w_CPU_CONTROL. This starts set-up mode. Next, the system reports the slave status of the initialized CANsync slaves (see “Command and Response Channel” on page 155). When all the CANsync slaves have reported and have the synchronized status, active operation must be enabled. This is done by setting bit 6 in *.w_CPU_CONTROL (*.w_CPU_CONTROL = 16#0040).

Enabling may be carried out even if not all the CANsync slaves have reported but if the application can administer this.

If you set bit 7 of *.w_CPU_CONTROL (*.w_CPU_CONTROL = 16#0080), the CAN controller is reset and the bit is cleared. This makes it possible to reset the CAN controller's BUS-OFF status and to send and receive CANsync message frames again. The system displays the BUS-OFF status in *.w_CANsync_STATUS (See “General Registers of the CANsync Interface Module” on page 141.).

The following table lists the registers that can be operated at initialization.

Register	Contents
*.b_MA_SL_MODE	Operating mode: Master/slave (SYNC-OUT/SYNC-IN)
*.b_AC	Acceptance Code of the CANsync controller
*.b_AM	Acceptance Mask of the CANsync controller
*.b_BT_0	Bit timing register 0 of the CANsync controller
*.b_BT_1	Bit timing register 1 of the CANsync controller
*.b_OUTPUT_CONTROL	Output control register of the CANsync controller
*.b_CLOCK_DIVIDER	Clock divider of the CANsync controller
*.b_TIME_PATTERN	CANsync interval in ms
*.a_SL_TYP[0]	CANsync slave type 0
*.a_SL_TYP[1]	CANsync slave type 1
...	...
*.a_SL_TYP[31]	CANsync slave type 31

(* At initialization, corresponds to structure _CANsync_INIT_MA, for example)

Reference Values

Reference values are sent in the CANsync event task.

Register	Contents
*.b_CTRLREG_WRC1	Control register of reference value channel 1
*.b_CTRLREG_WRC2	Control register of reference value channel 2
*.b_CTRLREG_WRC3	Reserved
*.b_CTRLREG_WRC4	Reserved
*.b_CTRLREG_WRC5	Reserved
*.b_CTRLREG_WRC6	Reserved
*.b_CTRLREG_WRC7	Reserved
*.b_CTRLREG_WRC8	Reserved

(* After initialization, corresponds to variable _CANsync_CTRL_MA, for example)

In the control register, the system marks with 16#05 the fact that new reference values for the respective reference value message frame (as well as the reference value channel [SWK or WRC]) were entered and that the message frame will be sent. The CANsync interface module acknowledges the command with 16#04.

Reference value message frames are configured in the following registers.

Register	Contents
*._CFG_WRC_WORD.b_CFG_WRC1_WORD0	Configuration of reference value message frame 1 word 0
*._CFG_WRC_WORD.b_CFG_WRC1_WORD1	Configuration of reference value message frame 1 word 1
*._CFG_WRC_WORD.b_CFG_WRC1_WORD2	Configuration of reference value message frame 1 word 2
*._CFG_WRC_WORD.b_CFG_WRC1_WORD3	Configuration of reference value message frame 1 word 3
*._CFG_WRC_WORD.b_CFG_WRC2_WORD0	Configuration of reference value message frame 2 word 0
*._CFG_WRC_WORD.b_CFG_WRC2_WORD1	Configuration of reference value message frame 2 word 1
*._CFG_WRC_WORD.b_CFG_WRC2_WORD2	Configuration of reference value message frame 2 word 2
*._CFG_WRC_WORD.b_CFG_WRC2_WORD3	Configuration of reference value message frame 2 word 3
*._CFG_WRC_WORD.b_CFG_WRC3_WORD0	Reserved
*._CFG_WRC_WORD.b_CFG_WRC3_WORD1	Reserved
*._CFG_WRC_WORD.b_CFG_WRC3_WORD2	Reserved
*._CFG_WRC_WORD.b_CFG_WRC3_WORD3	Reserved
...	...
*._CFG_WRC_WORD.b_CFG_WRC8_WORD0	Reserved
*._CFG_WRC_WORD.b_CFG_WRC8_WORD1	Reserved
*._CFG_WRC_WORD.b_CFG_WRC8_WORD2	Reserved
*._CFG_WRC_WORD.b_CFG_WRC8_WORD3	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

With the configuration, you state which reference value is to be entered at what location (..._WORD0, ..., ..._WORD3) in a reference value message frame (..._WRC1_..., ..._WRC2_...)

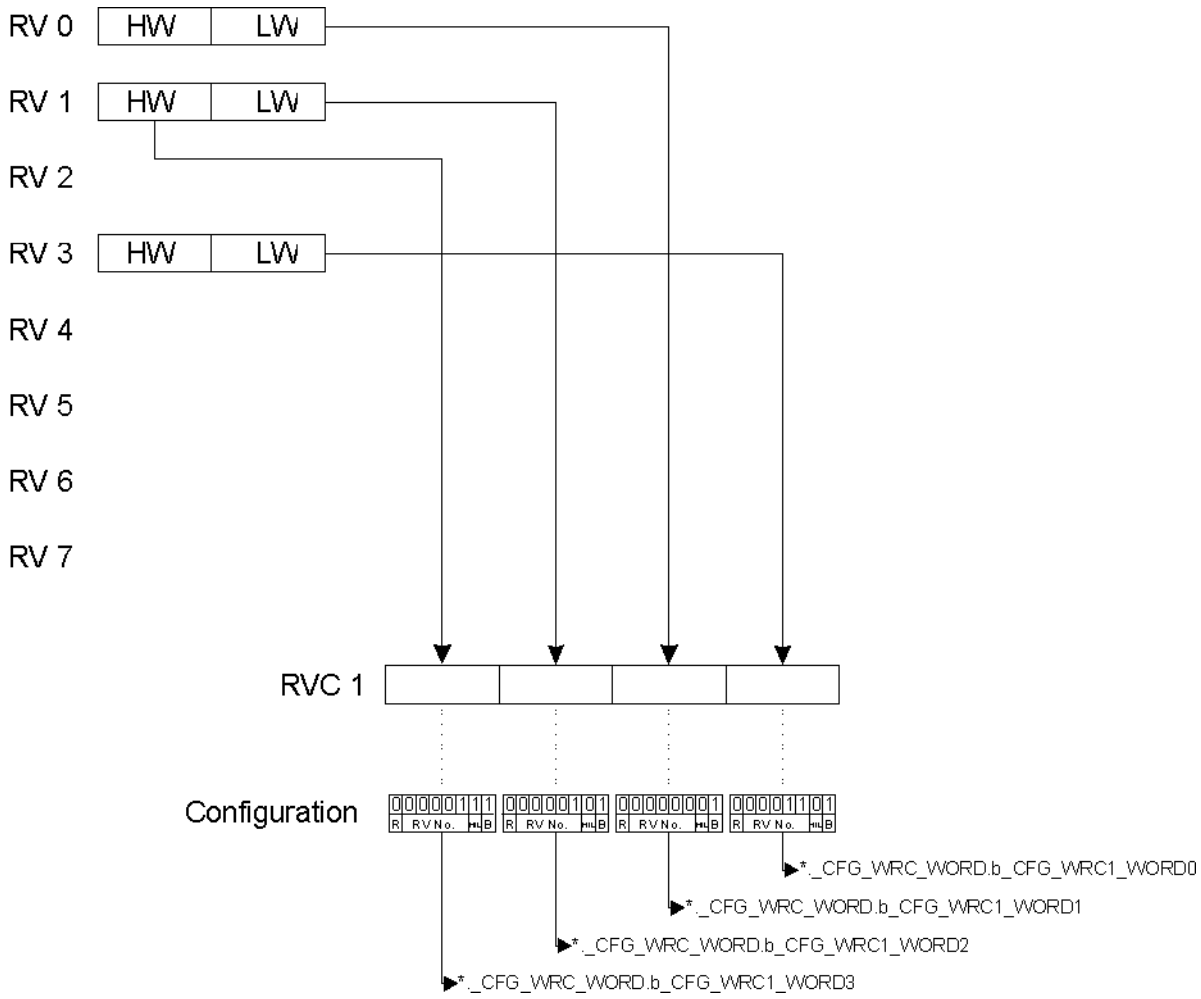
Meaning

Bit 0	**Assigned** If = 1, the system uses this word of the message frame
Bit 1	**Highword/lowword** If = 1, the system enters the highword of the reference value
Bit 2	**Reference value number** Number of reference value 0 to 31
Bit 3	
Bit 4	
Bit 5	
Bit 6	
Bit 7	Reserved, must be set to zero

NOTE

One word of the message is only not used if the following settings are made: reference value number = 0, highword/lowword = 0 and assigned = 0.

Example:



*._CFG_WRC_WORD.b_CFG_WRC1_WORD0 = 16#0D

0	0	0	0	1	1	0	1
R	RV No.					H/L	b

(* Corresponds, for example, to _CANsync_CTRL_MA).

This setting enters the lowword of reference value 3 in reference value message frame 1 in word 0.

Register	Contents
*.a_WR_VALUE[0]	Reference value 0
*.a_WR_VALUE[1]	Reference value 1
*.a_WR_VALUE[2]	Reference value 2
*.a_WR_VALUE[3]	Reference value 3
*.a_WR_VALUE[4]	Reference value 4
*.a_WR_VALUE[5]	Reference value 5
*.a_WR_VALUE[6]	Reference value 6
*.a_WR_VALUE[7]	Reference value 7
*.a_WR_VALUE[8]	Reserved
...	...
*.a_WR_VALUE[31]	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

The reference values can be word or doubleword reference values.

Explanation of Using the Reference Value Channels

In synchronous operation, two reference value channels (channels 1 and 2; also WRC1 and WRC2) are available. Reference value message frames 1 and 2 are sent on reference value channels 1 and 2 (RVC).

Both reference value message frames consist of four words each (W0 to W3). After CAN initialization, you must state at least once the assignment of these words. This can be carried out in the initialization program. To do this, enter in areas *._CFG_WRC_WORD.b_CFG_WRC1_WORD0 to *._CFG_WRC_WORD.b_CFG_WRC2_WORD3 for each word of the reference value message frames the reference value that is to be transferred at this location. Valid reference value numbers are in the range 0 to 7. For doubleword reference values, you must use two words in the message frame.

You can change the configuration even during active operation. The system applies the change in the next CANsync interval at the latest. At the start of the CANsync interval, the CANsync interface module reads the configuration data.

The system generates reference values in synchronous operation in the CANsync event task. The reference values for reference value message frame 1 must be entered as a reference value (*.a_WR_VALUE[0] to *.a_WR_VALUE[7]) by 490 µs after the start of the CANsync event task in communication RAM. This is because the CANsync interface module starts generating the reference value message frame then. To identify the fact that new reference values have been entered, the system must enter 16#05 in the appropriate control register (*.b_CTRLREG_WRC1 or *.b_CTRLREG_WRC2). This is the enable telling the CANsync interface module that the reference values can be read and that the reference value message frame is being generated. If this enable is not issued by 490 µs after the start

of the CANsync event task, reference value message frame 1 is omitted in this CANsync interval. To acknowledge generation of the reference value message frame, the CANsync interface module enters 16#04 in the control register. This allows users to check whether the system finished generation of the reference value in good time or not. If generation of the reference value takes a relatively long time – from the start of the CANsync event task to execution of the application program approximately 80 µs expire – the system must always generate the reference value for the next start of the CANsync event task and it only needs to copy the precalculated reference values to the corresponding locations in communication RAM at the start of the new CANsync event task.

The time by which the reference values for reference value message frame 2 must be entered is delayed by the execution time for generating reference value message frame 1. The duration depends on the number of reference value words that have to be entered (at least 15 µs to a maximum of 60 µs). Signalling in the control register is the same as for reference value message frame 1.

The reference value message frame is used to request the actual value message frame of a CANsync slave. The number of the CANsync slave interface module is entered in control register actual value request (*.b_CTRLREG_RD_ORDER_RDC1 or *.b_CTRLREG_RD_ORDER_RDC2) (for a further description, see “Actual Values” on page 148). Reference value message frame 1 requests actual value message frame 1, etc.

Actual Values

Actual value message frames are requested by a specific piece of information in the identifiers of reference value message frames. (See “Reference value channel 1” on page 130.)

Register	Contents
*.b_MAX_SL_NR	Maximum slave number (bus address of the CANsync slave

(* Corresponds, for example, to _CANsync_CTRL_MA).

This states the highest slave number of the CANsync slave for automatic actual value request (see *.b_CTRLREG_RD_ORDER_RDC1 ff.).

Register	Contents
*.b_CTRLREG_RD_ORDER_RDC1	Control register actual value request of actual value channel 1
*.b_CTRLREG_RD_ORDER_RDC2	Control register actual value request of actual value channel 2
*.b_CTRLREG_RD_ORDER_RDC3	Reserved
*.b_CTRLREG_RD_ORDER_RDC4	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

In the control register, you can state the slave number of the CANsync slave that is to report its actual values in the corresponding actual value message frame. If 16#80 is entered in the control register, the system increments the slave number by 1 in every cycle until the maximum slave number (*.b_MAX_SL_NR) is reached. The system then starts again with slave number 0.

Register	Contents
*.b_STATREG_RDC1	Status register of actual value channel 1
*.b_STATREG_RDC2	Status register of actual value channel 2
*.b_STATREG_RDC3	Reserved
*.b_STATREG_RDC4	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

The system enters in the status register the slave number of the CANsync slave from which an actual value message frame was received.

Register	Contents
*.a_STATREG_RDC[0].b_STATREG_RDC1	Actual value acknowledgement of actual value channel 1 of slave 0
*.a_STATREG_RDC[0].b_STATREG_RDC2	Actual value acknowledgement of actual value channel 2 of slave 0
*.a_STATREG_RDC[0].b_STATREG_RDC3	Reserved
*.a_STATREG_RDC[0].b_STATREG_RDC4	Reserved
*.a_STATREG_RDC[1].b_STATREG_RDC1	Actual value acknowledgement of actual value channel 1 of slave 1
*.a_STATREG_RDC[1].b_STATREG_RDC2	Actual value acknowledgement of actual value channel 2 of slave 1
*.a_STATREG_RDC[1].b_STATREG_RDC3	Reserved
*.a_STATREG_RDC[1].b_STATREG_RDC4	Reserved
*.a_STATREG_RDC[2].b_STATREG_RDC1	Actual value acknowledgement of actual value channel 1 of slave 2
*.a_STATREG_RDC[2].b_STATREG_RDC2	Actual value acknowledgement of actual value channel 2 of slave 2
*.a_STATREG_RDC[2].b_STATREG_RDC3	Reserved
*.a_STATREG_RDC[2].b_STATREG_RDC4	Reserved
...	...
*.a_STATREG_RDC[31].b_STATREG_RDC1	Actual value acknowledgement of actual value channel 1 of slave 31
*.a_STATREG_RDC[31].b_STATREG_RDC2	Actual value acknowledgement of actual value channel 2 of slave 31
*.a_STATREG_RDC[31].b_STATREG_RDC3	Reserved
*.a_STATREG_RDC[31].b_STATREG_RDC4	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

When the corresponding actual value message frame for a CANsync slave has arrived, the system enters 16#02 in the actual value acknowledgement register.

Actual value message frames are configured in the following registers.

Register	Contents
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD0	Configuration of actual value message frame 1 word 0 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD1	Configuration of actual value message frame 1 word 1 slave 0

Register	Contents
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD2	Configuration of actual value message frame 1 word 2 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD3	Configuration of actual value message frame 1 word 3 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD0	Configuration of actual value message frame 2 word 0 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD1	Configuration of actual value message frame 2 word 1 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD2	Configuration of actual value message frame 2 word 2 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD3	Configuration of actual value message frame 2 word 3 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD0	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD1	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD2	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD3	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD0	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD1	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD2	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD3	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD0	Configuration of actual value message frame 1 word 0 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD1	Configuration of actual value message frame 1 word 1 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD2	Configuration of actual value message frame 1 word 2 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD3	Configuration of actual value message frame 1 word 3 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD0	Configuration of actual value message frame 2 word 0 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD1	Configuration of actual value message frame 2 word 1 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD2	Configuration of actual value message frame 2 word 2 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD3	Configuration of actual value message frame 2 word 3 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC3_WORD0	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC3_WORD1	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC3_WORD2	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC3_WORD3	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC4_WORD0	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC4_WORD1	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC4_WORD2	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC4_WORD3	Reserved
...	...
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD0	Configuration of actual value message frame 1 word 0 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD1	Configuration of actual value message frame 1 word 1 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD2	Configuration of actual value message frame 1 word 2 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD3	Configuration of actual value message frame 1 word 3 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD0	Configuration of actual value message frame 2 word 0 slave 31

Register	Contents
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD1	Configuration of actual value message frame 2 word 1 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD2	Configuration of actual value message frame 2 word 2 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3	Configuration of actual value message frame 2 word 3 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD0	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD1	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD2	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD3	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD0	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD1	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD2	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD3	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

Your configuration tells the system which actual value it is to transfer at which location in the actual value message frame.

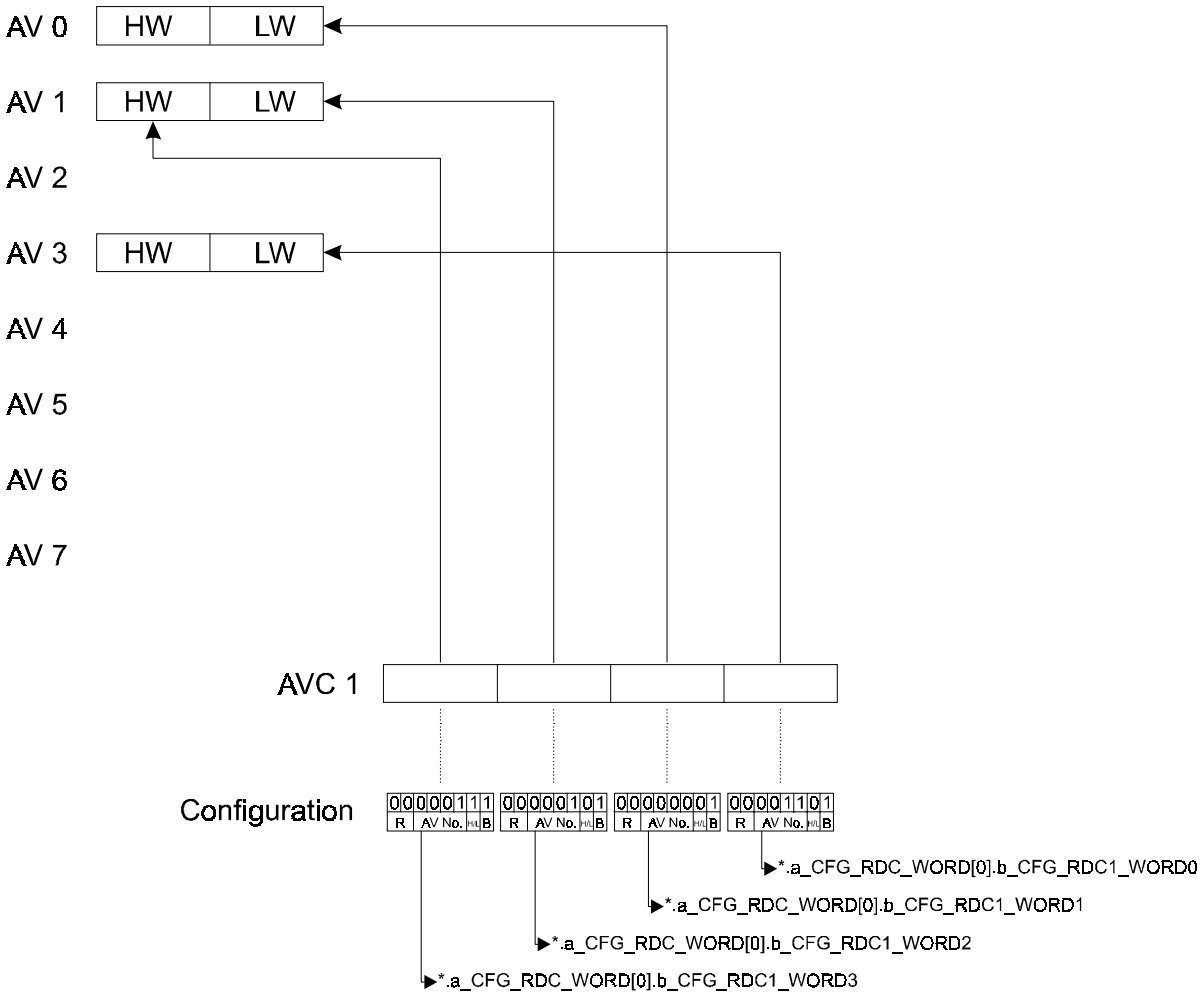
Meaning

Bit 0	**Assigned** If = 1, the system uses this word of the message frame
Bit 1	**Highword/lowword** If = 1, the system reads the highword of the actual value
Bit 2 Bit 3 Bit 4 Bit 5	**Actual value number** Number of actual value 0 to 15
Bit 6 Bit 7	Reserved

NOTE

One word of the message frame is only not used if the following settings are made: actual value number = 0, highword/lowword = 0 and assigned = 0.

Example:



*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD0 = 16#0D

0	0	0	0	1	1	0	1
R		AV No.				H/L	B

(* Corresponds, for example, to _CANsync_CTRL_MA).

This means that the system reads word 0 from actual value message frame 1 of CANsync slave 0 and writes it to the lowword of actual value 3.

Register	Contents
*.a_RD_BMARRAY[0][0]	Actual value 0 of slave 0
*.a_RD_BMARRAY[0][1]	Actual value 1 of slave 0
*.a_RD_BMARRAY[0][2]	Actual value 2 of slave 0
*.a_RD_BMARRAY[0][3]	Actual value 3 of slave 0
*.a_RD_BMARRAY[0][4]	Actual value 4 of slave 0
*.a_RD_BMARRAY[0][5]	Actual value 5 of slave 0
*.a_RD_BMARRAY[0][6]	Actual value 6 of slave 0
*.a_RD_BMARRAY[0][7]	Actual value 7 of slave 0
*.a_RD_BMARRAY[0][8]	Reserved

Register	Contents
...	...
*.a_RD_BMARRAY[0][15]	Reserved
*.a_RD_BMARRAY[1][0]	Actual value 0 of slave 1
*.a_RD_BMARRAY[1][1]	Actual value 1 of slave 1
*.a_RD_BMARRAY[1][2]	Actual value 2 of slave 1
*.a_RD_BMARRAY[1][3]	Actual value 3 of slave 1
*.a_RD_BMARRAY[1][4]	Actual value 4 of slave 1
*.a_RD_BMARRAY[1][5]	Actual value 5 of slave 1
*.a_RD_BMARRAY[1][6]	Actual value 6 of slave 1
*.a_RD_BMARRAY[1][7]	Actual value 7 of slave 1
*.a_RD_BMARRAY[1][8]	Reserved
...	...
*.a_RD_BMARRAY[1][15]	Reserved
...	...
*.a_RD_BMARRAY[31][0]	Actual value 0 of slave 31
*.a_RD_BMARRAY[31][1]	Actual value 1 of slave 31
*.a_RD_BMARRAY[31][2]	Actual value 2 of slave 31
*.a_RD_BMARRAY[31][3]	Actual value 3 of slave 31
*.a_RD_BMARRAY[31][4]	Actual value 4 of slave 31
*.a_RD_BMARRAY[31][5]	Actual value 5 of slave 31
*.a_RD_BMARRAY[31][6]	Actual value 6 of slave 31
*.a_RD_BMARRAY[31][7]	Actual value 7 of slave 31
*.a_RD_BMARRAY[31][8]	Reserved
...	...
*.a_RD_BMARRAY[31][15]	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

The actual values can be word or doubleword actual values.

NOTE

When displaying _CANsync_CTRL_MA in the PROPROG wt II watch window, a period may be between the square brackets.

Explanation of Using the Actual Value Channels

In synchronous operation, two actual value channels (channels 1 and 2; also RDC1 and RDC2) are available. Actual value message frames 1 and 2 are sent on actual value channels 1 and 2 (AVC).

Both actual value message frames consist of four words each (W0 to W3). After CANsync initialization, you must state at least once the assignment of these words. This can be carried out in the initialization program.
To do this, you must enter in area *.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD0 to *.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3 for each assigned word of the actual value message frames (of each CANsync slave) the actual value that is transferred at this position. Valid actual value numbers are in the range 0 to 7. For doubleword actual values, you must use two words in the message frame.

You can change the configuration even during active operation. The system applies the change in the next CANsync interval at the latest. (The CANsync interface module reads the configuration data when the corresponding actual value message frame is received).

The system carries out actual value message frame evaluation at the start of the CANsync event task. There are two evaluation methods: With the first one, you poll the status registers of the actual value channels (*.b_STATREG_RDC1 or *.b_STATREG_RDC2). The system enters in these registers the slave number of the CANsync slave from which the corresponding actual value message frame was received. You can then read out the actual values from registers (*.a_RD_BMARRAY[0][0] to *.a_RD_BMARRAY[31][7]) of this CANsync slave and set the status register to 16#FF, for example, to be able to detect correctly the next entry.

With the second method, you poll directly actual value acknowledgement (*.a_STATREG_RDC[0].b_STATREG_RDC1 to *.a_STATREG_RDC[31].b_STATREG_RDC2) for one CANsync slave. In this acknowledgement register, the system marks with 16#02 reception of an actual value message frame. You can then read out the actual values from registers (*.a_RD_BMARRAY[0][0] to *.a_RD_BMARRAY[31][7]) of this CANsync slave. In this case too, the system must then write another value to the status register to detect reentry of the acknowledgement.

You must assign in accordance with the actual value configuration in the application program the actual values that must be read when an actual value message frame has been received.

The system uses the reference value message frame to request a CANsync slave to send its actual value message frame.

The number of the CANsync slave interface module is entered in control register actual value request (*.b_CTRLREG_RD_ORDER_RDC1 and *.b_CTRLREG_RD_ORDER_RDC2). Reference value message frame 1 requests actual value message frame 1, etc.

The request number for actual value message frame 1 and actual value message frame 2, etc. can be the same or different, i.e. CANsync slave x can request actual value message frame 1 and CANsync slave y can request actual value message frame 2.

There are two options for the request number of the CANsync slave: The first one is to state directly the slave number in the control register. While this register is not changed, the system addresses the same CANsync slave in each CANsync interval.

If you enter 16#80 as the request number, the CANsync interface module automatically increments the slave number by 1 in each CANsync interval until the maximum slave number (*.b_MAX_SL_NR) is reached. Polling then starts again at slave number 0. This makes possible automatic requesting of the actual value message frames of all the CANsync slaves that are present. If you assign noncontiguous slave numbers to CANsync slaves, the system generates an actual value request for the CANsync slaves that are not present but this does not lead to any functional problems.

Command and Response Channel

Configuring the Command Channel

The command channel is configured in the following registers.

Register	Contents
*.b_SL_NR_CONTROLWORD	Slave number of the CANsync slave for control word command
*.b_SL_NR_ PARAMETER	Slave number of the CANsync slave for parameter word command
*.b_SL_NR_UPDOWN	Slave number of the CANsync slave for the upload/download command

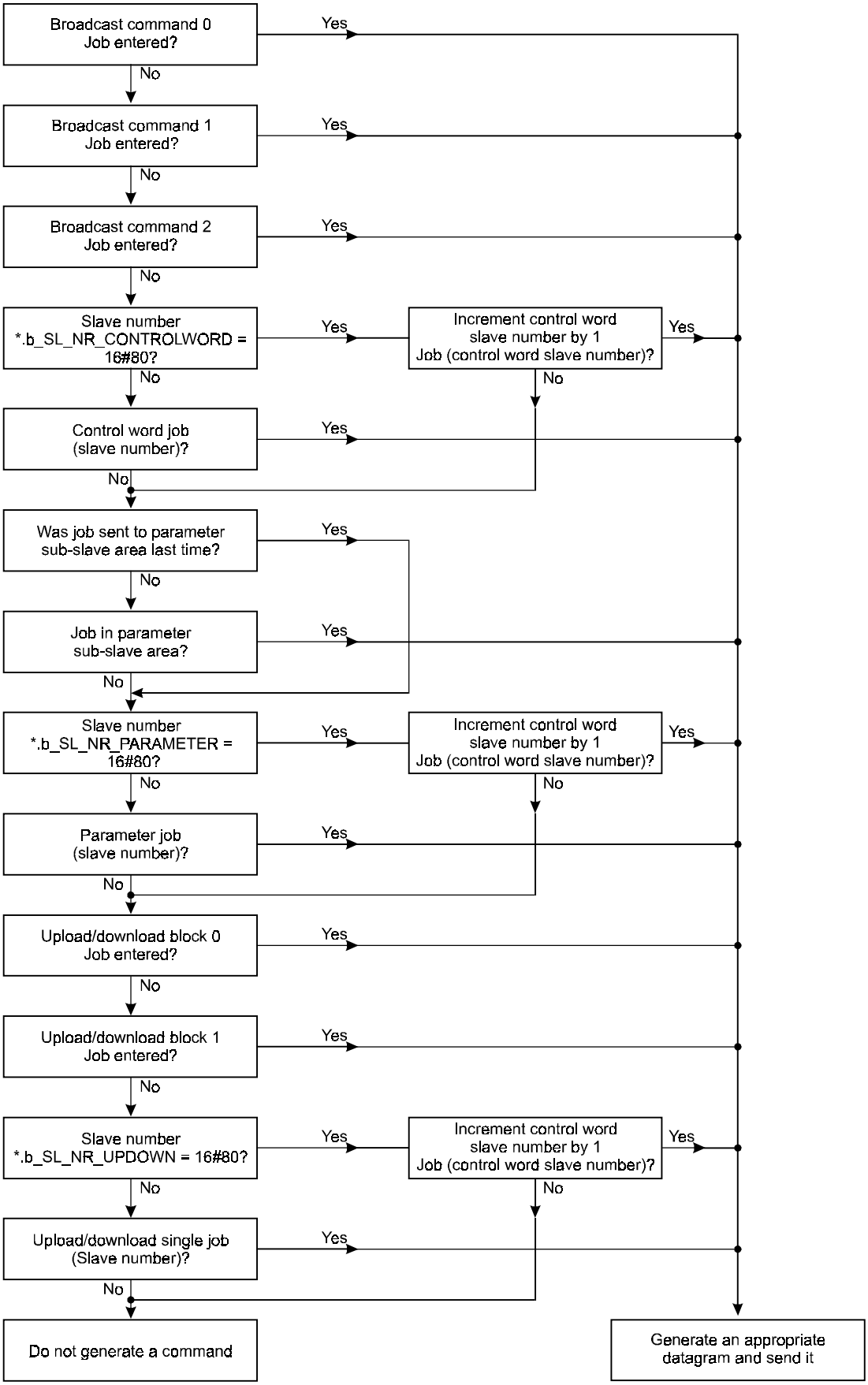
(* Corresponds, for example, to _CANsync_CTRL_MA).

Explanation of configuration of the command channel

Using configuration registers *.b_SL_NR_CONTROLWORD, *.b_SL_NR_PARAMETER and *.b_SL_NR_UPDOWN, you can set the use of the command channel. The normal setting is that all three registers are set to 16#80. In this case, the system cyclically increments the slave numbers of the CANsync slaves for which command message frames may be generated. The maximum slave number is the same as for actual value request (*.b_MAX_SL_NR).

As an alternative, you can explicitly specify the slave number. In this case, the system only checks whether the corresponding command should be generated for this CANsync slave.

In any one CANsync interval, it is only possible to send one command. The following scheme indicates the sequence in which the system polls particular areas in synchronous operation and whether an order for generating a commands is entered. The various commands are explained in the next few sections.



(* Corresponds, for example, to _CANsync_CTRL_MA).

The scheme shows you which jobs are treated with higher priority (e.g. broadcast commands), which jobs can block other jobs and the number of CANsync intervals at the latest after which the system processes a job. Broadcast command 0 has the highest priority: it is sent immediately in this CANsync interval. The system only checks the next area if no job is entered in this area. The system ignores the pre-event history. This means that if a job is entered in broadcast area 0 in every CANsync interval, no other commands are sent.

Broadcast Command

Broadcast commands can be sent using the following registers:

Register	Contents
*.b_CTRL_REG_BC0	Control register of broadcast command 0
*.d_SL_MASK_BC0	Bit strip of broadcast command 0
*.b_CMD_BC0	Command byte of broadcast command 0
*.b_DATA_BYTE_BC0	Data byte 0 of broadcast command 0
*.w_DATA_WORD_BC0	Data word 1 of broadcast command 0 (DW)

(* Corresponds, for example, to _CANsync_CTRL_MA).

Register	Contents
*.b_CTRL_REG_BC1	Control register of broadcast command 1
*.d_SL_MASK_BC1	Bit strip of broadcast command 1
*.b_CMD_BC1	Command byte of broadcast command 1
*.b_DATA_BYTE_BC1	Data byte 0 of broadcast command 1
*.w_DATA_WORD_BC1	Data word 1 of broadcast command 1 (DW)

(* Corresponds, for example, to _CANsync_CTRL_MA).

Register	Contents
*.b_CTRL_REG_BC2	Control register of broadcast command 2
*.d_SL_MASK_BC2	Bit strip of broadcast command 2
*.b_CMD_BC2	Command byte of broadcast command 2
*.b_DATA_BYTE_BC2	Data byte 0 of broadcast command 2 (DB)
*.w_DATA_WORD_BC2	Data word 1 of broadcast command 2 (DW)

(* Corresponds, for example, to _CANsync_CTRL_MA).

In bit strip *.d_SL_MASK_BC0, *.d_SL_MASK_BC1, *.d_SL_MASK_BC2, you state the CANsync slaves for which the action command is specified. Each CANsync slave has a bit assigned to it (bit 0 = slave 0; bit 1 = slave 1; ... bit 31 = slave 31).

When the bit = TRUE, the associated CANsync slave carries out the command.

In addition to transferring a command, it is also possible to transfer a data byte (*.b_DATA_BYTE...) and a data word (*.w_DATA_WORD...).

To be able to send a command, 16#05 must be entered in the control register. The CANsync interface module then returns 16#04 to confirm sending.

List of Commands

Command byte	Command
16#01	Control word DB = Not used DW = Control word
16#02 - 16#FF	Reserved

Control word command

The control word command is sent using the following registers.

Register	Contents
*.a_STEUREG_CONTROLWORD[0]	Control register of control word for slave 0
*.a_STEUREG_CONTROLWORD[1]	Control register of control word for slave 1
*.a_STEUREG_CONTROLWORD[2]	Control register of control word for slave 2
*.a_STEUREG_CONTROLWORD[3]	Control register of control word for slave 3
...	...
*.a_STEUREG_CONTROLWORD[31]	Control register of control word for slave 31

(* Corresponds, for example, to _CANsync_CTRL_MA).

To be able to send a control word to a CANsync slave, the system must enter the control word in the area *.a_CONTROLWORD_SL[0] onwards and then enter 16#05 to the corresponding control register from *.a_STEUREG_CONTROLWORD[0] onwards. When the control word has been sent, the system returns 16#04 as the acknowledgement.

Example: Sending a control word to CANsync slave 3:

1. Enter control word in *.a_CONTROLWORD_SL[3].
2. Enter 16#05 in control register *.a_STEUREG_CONTROLWORD[3].

   Acknowledgement after sending: 16#04 to *.a_STEUREG_CONTROLWORD[3].

Register	Contents
*.a_CONTROLWORD_SL[0]	Control word for slave 0
*.a_CONTROLWORD_SL[1]	Control word for slave 1
*.a_CONTROLWORD_SL[2]	Control word for slave 2
*.a_CONTROLWORD_SL[3]	Control word for slave 3
...	...
*.a_CONTROLWORD_SL[31]	Control word for slave 31

(* Corresponds, for example, to _CANsync_CTRL_MA).

On the CANsync bus, the control word command is mapped to an action command in which, in the bit mask (bit strip), only the bit for one CANsync slave is set, the command byte is set to 16#01 and the control word is entered as the data word (DW).

Parameter command

You can read and write parameters using the following registers:

Register	Contents
*.a_CTRLREG_PAR_CMD_SL[0].b_STEUREG_PAR_CMD	Control register of parameter command 0
*.a_CTRLREG_PAR_CMD_SL[0].b_STATREG_PAR_CMD	Status register of parameter command 0
*.a_CTRLREG_PAR_CMD_SL[1].b_STEUREG_PAR_CMD	Control register of parameter command 1
*.a_CTRLREG_PAR_CMD_SL[1].b_STATREG_PAR_CMD	Status register of subslave command 1
...	...
*.a_CTRLREG_PAR_CMD_SL[31].b_STEUREG_PAR_CMD	Control register of parameter command 31
*.a_CTRLREG_PAR_CMD_SL[31].b_STATREG_PAR_CMD	Status register of parameter command 31

(* Corresponds, for example, to _CANsync_CTRL_MA).

Meanings of the control and status register:

Bit 0	**Active** Must be set to 1 to start the command Is set to 0 when a response was received
Bit 1	**Read** =1: Read parameter
Bit 2	**Write** =1: Write parameter
Bit 3	**send** =1: Indication that the command has been sent
Bit 4	**Format** 0: Parameter is of word format 1: Parameter is of doubleword format
Bit 5	**Error display** 1: Error occurred 0: No error
Bit 6	**Busy** = 1 CANsync slave is processing the job, the data is not yet valid
Bit 7	Reserved

Register	Contents
*.a_DATA_PARAMETER[0].d_PAR_VALUE *.a_DATA_PARAMETER[0].w_PAR_NR *.a_DATA_PARAMETER[0].w_SUBSL_ADR	Data of CANsync slave 0 Parameter number 0 Reserved
*.a_DATA_PARAMETER[1].d_PAR_VALUE *.a_DATA_PARAMETER[1].w_PAR_NR *.a_DATA_PARAMETER[1].w_SUBSL_ADR	Data of CANsync slave 1 Parameter number 1 Reserved
...	...
*.a_DATA_PARAMETER[31].d_PAR_VALUE *.a_DATA_PARAMETER[31].w_PAR_NR *.a_DATA_PARAMETER[31].w_SUBSL_ADR	Data of CANsync slave 31 Parameter number 31 Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

The system enters or receives the parameter value under Data of CANsync slave x. This value can be a word or a doubleword.

The parameter number selects a parameter in the addressed CANsync slave. Refer to the description of the CANsync slave for the meaning of the parameter.

Sequence of a parameter access

The parameter commands can be used in the CANsync event task as well as in the rest of the program. To ensure data consistency, the system allocates jobs and evaluates the response via the control register **and** the status register. To guarantee that the job is carried out without conflicts, first the control register and then the status register must always be read from the application program and if a new value is written, the program must always first set the status register and then the control register. (The control register is the crucial one for the CANsync interface module).

In the following explanation, the addresses refer to CANsync slave 0.

For a write parameter job, you enter
the **data value** in *.a_DATA_PARAMETER[0].d_PAR_VALUE (doubleword) and
the **parameter number** in *.a_DATA_PARAMETER[0].w_PAR_NR

After this, you enter the value 16#05 in status register
*.a_CTRLREG_PAR_CMD_SL[0].b_STATREG_PAR_CMD and in control register
*.a_CTRLREG_PAR_CMD_SL[0].b_STEUREG_PAR_CMD if the system is to write a word parameter or 16#15 if it is to write a doubleword parameter.

When the CAN interface module has taken over the job (in accordance with “Configuring the Command Channel” on page 155) the system confirms by setting bit 3 (send) in the control register and in the status register.

If the CANsync slave has accepted the job but has not yet completed it, the system sets bit 6 (busy).

If an error occurs in the CANsync slave while it is carrying out the job, the system sets bit 5 (error display), clears bit 6 (busy) and bit 0 (active); and the error number can be read in data range *.a_DATA_PARAMETER[0].d_PAR_VALUE.

When the CANsync slave has completed the job the system sets bit 6 (busy) and clears bit 0 (active); in the byte, this results in 16x04 for write word or 04x14 for write doubleword.

The evaluation in the application program can be limited to polling bit 0; while the bit is set, the system continues to process the job and when it is reset, the job is completed.

It may well be that bit 6 (busy) is not set if the CANsync slave can respond to the job immediately.

The read parameter job runs in a similar way with the only differences being that the command is 16#03 for read word and 16#13 for read doubleword and that the system does not enter the data before the job, but rather that it is available for reading out after the system cleared bit 0 (active). In the byte the acknowledgement then reads 16#02 for word access and 16#12 fo doubleword access. The system takes the format from the CANsync slave's response.

Error number of the CANsync slave

Value	Meaning
16#0000	No error occurred
16#FFFF	Error occurred
16#FFFE	Value less than minimum value
16#FFFD	Value greater than maximum value
16#FFFC	Element cannot be changed
16#FFFB	Element not present
16#FFFA	Data is not available (e.g. being processed)
16#FFF9	Error in data format

Upload/Download Block Area

Upload and download jobs can be controlled using the following registers:

Register	Contents
*.b_STEUREG_UPDOWNBLK0	Reserved
*.b_STATREG_UPDOWNBLK0	Reserved
*.b_SL_NR_UPDOWNBLK0	Reserved
*.w_ERR_NR_UPDOWNBLK0	Reserved
*.w_SUBSL_NR_UPDOWNBLK0	Reserved
*.d_BASE_ADR_UPDOWNBLK0	Reserved
*.w_LENGTH_UPDOWNBLK0	Reserved
*.w_COUNTER_UPDOWNBLK0	Reserved
*.a_DATA_UPDOWNBLK0[0 to 74]	Reserved
*.b_STEUREG_UPDOWNBLK1	Control register of up/down block 1
*.b_STATREG_UPDOWNBLK1	Status register of up/down block 1
*.b_SL_NR_UPDOWNBLK1	CANsync slave number of up/down block 1
*.w_ERR_NR_UPDOWNBLK1	Error number of up/down block 1
*.w_SUBSL_NR_UPDOWNBLK1	Sub-slave number of up/down block 1
*.d_BASE_ADR_UPDOWNBLK1	Base address of up/down block 1
*.w_LENGTH_UPDOWNBLK1	Length in bytes of up/down block 1
*.w_COUNTER_UPDOWNBLK1	Counter of up/down block 1
*.a_DATA_UPDOWNBLK1[0 to 74]	Data block of up/down block 1 (75 doublewords)

(* Corresponds, for example, to _CANsync_CTRL_MA).

NOTE

Block 0 is reserved for the OmegaOS .

Meanings of the control and status register

Bit 0	**Active** Must be set to 1 to start the command Is set to 0 when the command is being processed
Bit 1	**send** =1: Indication that the command has been sent
Bit 2 Bit 3	**Mode** Bit3 bit2 0 0: reserved 0 1: initialization 1 0: ongoing upload/download 1 1: End of block
Bit 4	**Upload/download** 0: Upload 1: Download

Bit 5	**Error display** 1: Error occurred 0: No error
Bit 6	**busy** = 1 CANsync slave is processing the job, the data is not yet valid
Bit 7	**Reset** Cancellation of a job

Sequence of an Upload/Download Job in Block 1

The upload/download commands can be used in the CANsync event task as well as in the rest of the program. To ensure data consistency, the system allocates jobs and evaluates the response via the control register and the status register. To guarantee that the job is carried out without conflicts, first the control register and then the status register must always be read from the application program and if a new value is written, the program must always first set the status register and then the control register. (The control register is the crucial one for the CANsync interface module).

For a download, the system first fills data block area *.a_DATA_UPDOWNBLK1[0] to *.a_DATA_UPDOWNBLK1[74] with the data to be transferred.The maximum block length that can be sent with one download job is 300 bytes (75 doublewords).

The number of the CANsync slaves that is to receive the download is entered in *.b_SL_NR_UPDOWNBLK1 and the base address is entered in *.d_BASE_ADR_UPDOWNBLK1.

Base addresses 16#0000_0000 to 16#0000_00FF are reserved for operating system jobs.

After this, you enter the value 16#15 in status register *.b_STATREG_UPDOWNBLK1 and in control register *.b_STEUREG_UPDOWNBLK1.

The CANsync interface module will now try to transfer the entire block to the CANsync slave. The system automatically generates the sequential modes. The progress can be read off the byte counter in *.w_COUNTER_UPDOWNBLK1.

When the CANsync interface module has taken over the job (See “Configuring the Command Channel” on page 155.) the system confirms by setting bit (send) in the control register and in the status register.

If the CANsync slave has accepted the job but has not yet completed it, the system sets bit 6 (busy).

If an error occurs in the CANsync slave while it is carrying out the job, the system sets bit 5 (error display), clears bit 6 (busy) and bit 0 (active); and the error number can be read in *.b_SL_NR_UPDOWNBLK1.

When the CANsync slave has completed the job the system sets bit 6 (busy) and clears bit 0 (active); in the control and status register, this results in 16#1C.

The evaluation in the application program can be limited to polling bit 0 (active). While the bit is set, the system continues to process the job and when it is reset, the job is completed.

It may well be that bit 6 (busy) is not set if the CANsync slave can respond to the individual modes of the job immediately.

The upload job runs in a similar way with the only differences being that the command is 16#05 and that the data is available for reading out after bit 0 (active) has been cleared rather than before the job has been entered. The acknowledgement in the control and status register then reads 16#0C.

You can cancel an upload/download job by setting bit 7 (reset) to TRUE. The system then sends to the CANsync slave an upload initialization message frame with address = 0 and length = 0.

Upload/Download Error Numbers

Error number	Meaning
16#0001	CANsync slave acknowledges wrong block number
16#0002	Entered length greater than 300 bytes
...	
16#0100	CANsync slave expects block with the number that is entered in the counter
16#0101	CANsync slave expects block end
16#0102	CANsync slave does not yet expect block end
16#0103	CANsync slave cancels upload/download
16#0104	Upload/download not possible
16#0105	Base address not allowed
16#0106	Reserved
16#0107	Block length > CANsync slave's maximum block length
16#0108	Message frame mode error (mode not allowed at this stage)

CANsync Slave Status

Register	Contents
*.a_STAT_SL[0]	CANsync slave status 0
*.a_STAT_SL[1]	CANsync slave status 1
*.a_STAT_SL[2]	CANsync slave status 2
...	...
*.a_STAT_SL[31]	CANsync slave status 31

(* Corresponds, for example, to _CANsync_CTRL_MA).

Meaning of CANsync slave status:

Bit 0	**Response** = 1: CANsync slave responds
Bit 1	**SYNCHRONIZED** = 1: CANsync slave is synchronized
Bit 2	Reserved
Bit 3	
Bit 4	
Bit 5	
Bit 6	
Bit 7	

In set-up mode (see “Initialization:” on page 142), the system polls the slave status of all the initialized CANsync slaves and enters it here. Bit 0 (response) indicates that the CANsync slave has logged on to the CANsync bus. Bit 1 (synchronized) indicates that the CANsync slave is synchronized and that synchronous operation can be started.

7.2.3 Register Structure and Function of the Ωmega CANsync Slave

In the following sections, we will explain the register structure of communication RAM in the Ωmega CANsync slave.

To allow you to access registers of the communication RAM in the PROPROG wt II project, data types are defined that map the register structure. The system uses these data types to declare variables that are assigned to the CANsync interface module's base address.

After this, it is possible to access the registers of communication RAM via the structure elements of the declared variables.

At initialization of the CANsync slave interface module, the registers in communication RAM have a different meaning than after initialization in cyclical operation.

This means that, for initialization there is the

CANsync_INIT_BMSTRUCT

and for cyclical operation, the

CANsync_SL_CTRL_BMSTRUCT

These structures are defined from library BM_TYPES_20bd00 onwards. After you have integrated library BM_TYPES_20bd00 in the project, the data types are available.

These structures contain

- 8-bit elements,
- 16-bit elements,
- 32-bit elements,
- Structures from the elements mentioned above
- Fields (ARRAY) and structures from the elements and structures mentioned above

Short designations have been prepended to the data types (8-, 16-, 32-bit elements, structures and fields) that are used in a structure. This is for the sake of clarity when using the structures in programming.

Data type	Short designation	Number of bits
BYTE	b	8
WORD	W	16
DWORD (double word)	d	32
SINT (short integer)	Si	8
DINT (double integer)	di	32
USINT (unsigned short integer)	us	8
UINT (unsigned integer)	u	16
UDINT (unsigned double integer)	ud	32
STRUCT	_ (underline)	-
ARRAY	a	-

Other data types that are not used in the structures include:

Data type	Short designation	Number of bits
BOOL (bit)	X	1
TIME	t	-

Explanation of declaring the global variables

For initialization, you create a global variable of data type CANsync_INIT_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) in Ωmega Drive-Line II

```
_CANsync_INIT_SL          AT     %MB3.100000 : CANsync_INIT_BMSTRUCT;
```

Where:

`CANsync_INIT_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_INIT_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II.

For cyclical operation, you create a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) in Ωmega Drive-Line II

```
_CANsync_CTRL_SL          AT     %MB3.100000 : CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II.

NOTE

In the following tables, the variable name is replaced by an asterisk (*).

This means that you access register `*.w_CPU_CONTROL` via

`_CANsync_INIT_SL.w_CPU_CONTROL`,

you access `*.w_OPTION_STATUS` via

`_CANsync_INIT_SL.w_OPTION_STATUS` .

Where:

`CANsync_INIT_ SL`	is the variable name with the data type short designation "_" for STRUCT
`w_CPU_CONTROL`	is the control register of the CANsync interface module with data type short designation "w" for WORD

Registers *.w_CPU_CONTROL and *.w_OPTION_STATUS can also be triggered via the structure for cyclical operation. This makes possible access via

_CANsync_CTRL_SL.w_CPU_CONTROL and

_CANsync_CTRL_SL.w_OPTION_STATUS .

Where

CANsync_CTRL_ SL	is the variable name with the data type short designation "_" for STRUCT
w_CPU_CONTROL	is the status register of the CANsync interface module with data type short designation "w" for WORD

Example of accessing an element of a field that is used in the structure:

According to the table: *.a_WR_VALUE_RECEIVE[3]

Access: _CANsync_CTRL_SL.a_WR_VALUE_RECEIVE[3]

Where

CANsync_CTRL_ SL	is the variable name with the data type short designation "_" for STRUCT
a_WR_VALUE_RECEIVE[3]	is the register for reference value 3 with the data type short designation "a" for ARRAY. The data type of the elements of the field (of the reference values) is taken from the corresponding table and the description.

Example of accessing an element of a two-dimensional field that is used in the structure:

According to the table: *.a_RD_VALUE[5][7]

Access: _CANsync_CTRL_SL.a_RD_VALUE[5][7]

Where

CANsync_CTRL_ SL	is the variable name with the data type short designation "_" for STRUCT
a_RD_VALUE[5][7]	is the register for actual value 7 of CANsync slave 5 with the data type short designation "a" for ARRAY. The data type of the elements of the field (of the reference values) is taken from the corresponding table and the description.

Example of accessing an element a (sub) structure, which is itself a field that is used in the structure:

According to the table: *.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3

Access: _CANsync_CTRL_SL.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3

Where

CANsync_CTRL_ SL	is the variable name with the data type short designation "_" for STRUCT

a_CFG_RDC_WORD[31]	is the field containing the configuration data for mapping the words of CANsync slave 31's actual value message frames with the data type short designation "a" for ARRAY
b_CFG_RDC2_WORD3	is the register for the configuration data for mapping the third word of actual value message frame 2 (of CANsync slave 31) with the data type short designation "b" for BYTE

Example of accessing an element of a (sub) structure that is used in the structure:

According to the table: *._CFG_WRC_WORD.b_CFG_WRC1_WORD0

Access: _CANsync_CTRL_SL._CFG_WRC_WORD.b_CFG_WRC1_WORD0

Where

CANsync_CTRL_ SL	is the variable name with the data type short designation "_" for STRUCT
_CFG_WRC_WORD	is the structure containing the configuration data for mapping the words of the reference value message frames with the data type short designation "_" for STRUCT
b_CFG_WRC1_WORD0	is the register for the configuration data for mapping the 0th word of reference value message frame 1 with the data type short designation "b" for BYTE

General Registers of the CANsync Interface Module

Register	Contents
*.w_CANsync_STATUS	CANsync-Status
*.w_OMEGA_NR	The Ωmega number set using a DIP switch
*.i_SW1_NR	Card software number
*.i_SW1_RELEASE	Software revision incompatible and compatible

CANsync status

Each time the CANsync processor cycle is run through, the system outputs the CANsync status to *.w_CAN_STATUS.

Meaning:

Bit No.	Meaning (bit = TRUE)
0	Reserved
1	Overrun: A CANsync message could not be received
2	CANsync send buffer is free
3	CANsync send job executed successfully
4	CANsync message currently being received

Bit No.	Meaning (bit = TRUE)
5	CANsync message currently being sent
6	Error present (warning)
7	CANsync node is deactivated (BUS off)
8-15	Reserved

Ωmega number

In register *.w_OMEGA_NR, the system displays the Ωmega number that was set using the DIP switch (S33). In the case of a CANsync slave interface module, this number is the slave number.

Software number and software version

In register *.i_SW1_NR, the system displays the number of the CANsync software on the Ωmega Drive-Line II.

In register *.i_SW1_RELEASE, the system displays the compatible and the incompatible revision of the CANsync software on the Ωmega Drive-Line II.

Initialization

For initialization, you create a global variable of data type CANsync_INIT_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) in Ωmega Drive-Line II

```
_CANsync_INIT_SL          AT     %MB3.100000 : CANsync_INIT_BMSTRUCT;
```

Where:

`CANsync_INIT_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_INIT_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II.

In addition, you carry out configuration for synchronous operation:

Meaning		Register	Value
Baud rate	For example: 500 kbps	*.b_BT_0 *.b_BT_1	16#00 16#1C
CANsync interval	For example: 2 ms	*.b_TIME_PATTERN	16#02
Acceptance Code	All message frames	*.b_AC	16#FF
Acceptance Mask		*.b_AM	16#FF
Output Control	16#FA	*.b_OUTPUT_CONTROL	16#FA
Clock Divider	16#07	*.b_CLOCK_DIVIDER	16#07
Slave/Master	Slave	*.b_MA_SL_MODE	16#01

You set the operating mode via register *.w_CPU_CONTROL. The system displays the currently active operating mode in register *.w_OPTION_STATUS. You can change the operating mode even after it has been started successfully.

Register	Contents
*.w_CPU_CONTROL	Control register of CANsync interface module
*.w_OPTION_STATUS	Status register of CANsync interface module

(* at initialization, corresponds to _CANsync_INIT_SL, for example; after initialization, corresponds to _CANsync_CTRL_SL) for example

Control register of CANsync interface module	Meaning
16#0000	Cold restart
16#0001	Test Handshake
16#0002	Take over initialization data
16#0012	Reserved
16#0013	Reserved
16#0020	Start synchronous operating mode
16#0040	Enable active operation
16#0080	(Bit 7 = TRUE) Reset CAN controller

Status register of CANsync interface module	Meaning
16#0001	Start up
16#0002	Take over waiting for initialization data
16#0003	Waiting for start
16#0011	Reserved
16#0012	Reserved
16#0013	Setting up synchronous operation of CANsync slave
16#0020	Synchronous operation of CANsync slave is active
16#0041	Reserved
16#0042	Reserved
16#0043	*Setting up synchronous operation of CANsync master*
16#0080	*Synchronous operation of CANsync master is active*

Initialization is carried out using commands 16#0000, 16#0001 and 16#0002 to *.w_CPU_CONTROL. This starts set-up mode. After this, you can enable active operation. This is done by setting bit 6 in *.w_CPU_CONTROL (*.w_CPU_CONTROL = 16#0040).

If you set bit 7 of *.w_CPU_CONTROL, the CANsync controller is reset and the bit is cleared. This makes it possible to reset the CANsync controller's BUS-OFF status and to send and receive CANsync message frames again. The system displays the BUS-OFF status in *.w_CANsync_STATUS (See “General Registers of the CANsync Interface Module” on page 167.).

The following table lists the registers that can be operated at initialization.

Register	Contents
*.b_MA_SL_MODE	Operating mode: Master/slave (SYNC-OUT/SYNC-IN)
*.b_AC	Acceptance Code of the CANsync controller
*.b_AM	Acceptance Mask of the CANsync controller
*.b_BT_0	Bit timing register 0 of the CANsync controller
*.b_BT_1	Bit timing register 1 of the CANsync controller
*.b_OUTPUT_CONTROL	Output control register of the CANsync controller
*.b_CLOCK_DIVIDER	Clock divider of the CANsync controller
*.b_TIME_PATTERN	CANsync interval in ms

(* At initialization, corresponds to structure _CANsync_INIT_SL, for example)

Reference Values

Reference values are received in the CANsync event task.

Register	Contents
*.b_CTRLREG_WRC1	Status register of reference value channel 1
*.b_CTRLREG_WRC2	Status register of reference value channel 2
*.b_CTRLREG_WRC3	Reserved
*.b_CTRLREG_WRC4	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

In the status register of the reference value channels, 16#02 is used to report that the respective reference value message frame was received.

Reference value message frames are configured using the following registers.

Register	Contents
*._CFG_WRC_WORD.b_CFG_WRC1_WORD0	Configuration of reference value message frame 1 word 0
*._CFG_WRC_WORD.b_CFG_WRC1_WORD1	Configuration of reference value message frame 1 word 1
*._CFG_WRC_WORD.b_CFG_WRC1_WORD2	Configuration of reference value message frame 1 word 2
*._CFG_WRC_WORD.b_CFG_WRC1_WORD3	Configuration of reference value message frame 1 word 3
*._CFG_WRC_WORD.b_CFG_WRC2_WORD0	Configuration of reference value message frame 2 word 0
*._CFG_WRC_WORD.b_CFG_WRC2_WORD1	Configuration of reference value message frame 2 word 1
*._CFG_WRC_WORD.b_CFG_WRC2_WORD2	Configuration of reference value message frame 2 word 2
*._CFG_WRC_WORD.b_CFG_WRC2_WORD3	Configuration of reference value message frame 2 word 3
*._CFG_WRC_WORD.b_CFG_WRC3_WORD0	Reserved
*._CFG_WRC_WORD.b_CFG_WRC3_WORD1	Reserved
*._CFG_WRC_WORD.b_CFG_WRC3_WORD2	Reserved
*._CFG_WRC_WORD.b_CFG_WRC3_WORD3	Reserved
*._CFG_WRC_WORD.b_CFG_WRC4_WORD0	Reserved
*._CFG_WRC_WORD.b_CFG_WRC4_WORD1	Reserved
*._CFG_WRC_WORD.b_CFG_WRC4_WORD2	Reserved
*._CFG_WRC_WORD.b_CFG_WRC4_WORD3	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

With the configuration, you state which reference value is to be read at what location (..._WORD0, ..., ..._WORD3) in the reference value message frame (..._WRC1_..., ..._WRC2_...)

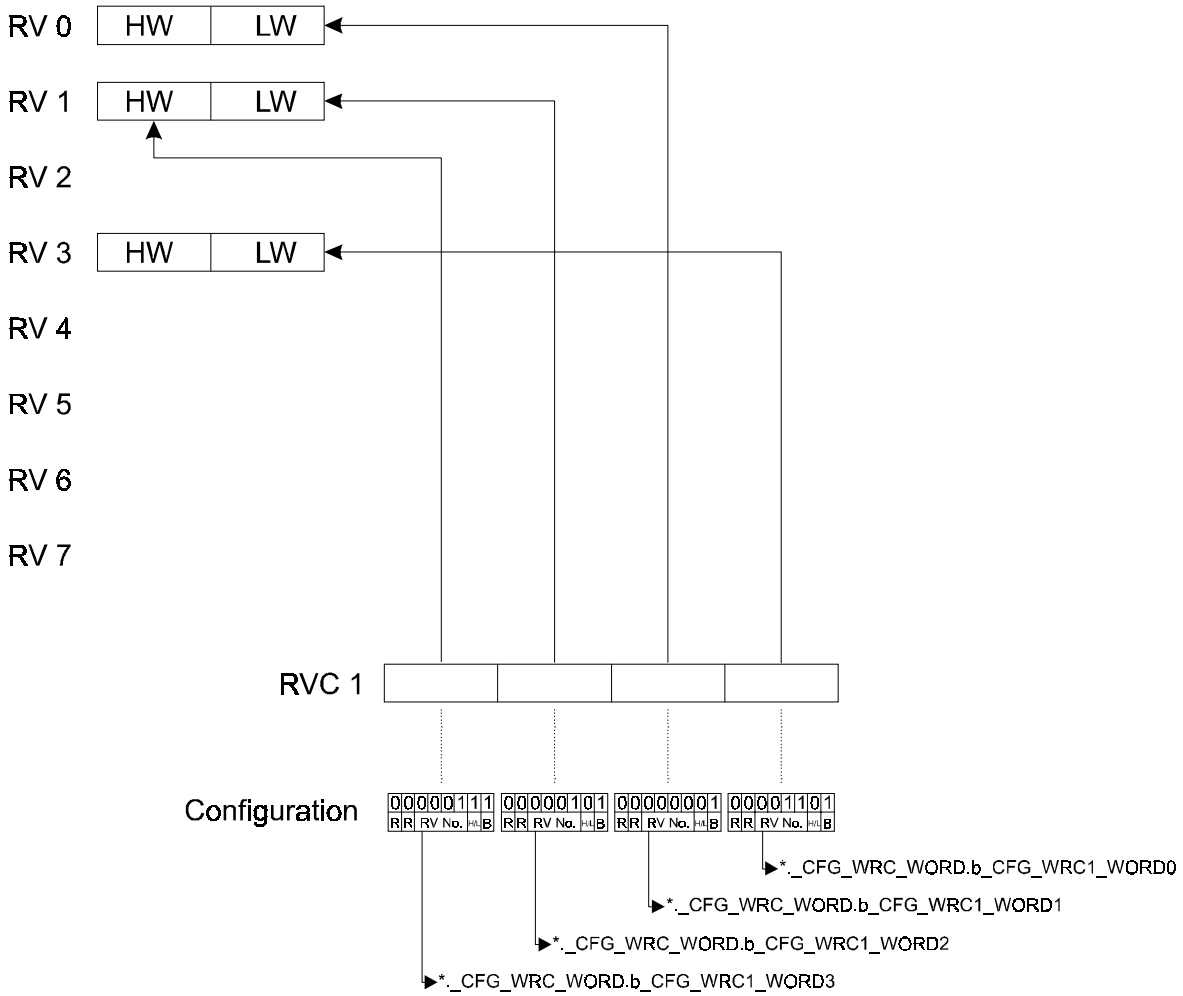
Meaning

Bit 0	**Assigned** If = 1, the system uses this word of the message frame
Bit 1	**Highword/lowword** If = 1, the system enters the highword of the reference value
Bit 2	**Reference value number** Number of reference value 0 to 15
Bit 3	
Bit 4	
Bit 5	
Bit 6	Reserved
Bit 7	

NOTE

One word of the message frame is only not used if the following settings are made: reference value number = 0, highword/lowword = 0 and assigned = 0.

Example:



*._CFG_WRC_WORD.b_CFG_WRC1_WORD0 = 16#0D

0	0	0	0	1	1	0	1
R	R	RV No.				H/L	B

(* Corresponds, for example, to _CANsync_CTRL_SL).

This setting enters word 0 of reference value message frame 1 as the lowword of reference value 3.

Register	Contents
*.a_WR_VALUE_RECEIVE[0]	Reference value 0
*.a_WR_VALUE_RECEIVE[1]	Reference value 1
*.a_WR_VALUE_RECEIVE[2]	Reference value 2
*.a_WR_VALUE_RECEIVE[3]	Reference value 3
*.a_WR_VALUE_RECEIVE[4]	Reference value 4
*.a_WR_VALUE_RECEIVE[5]	Reference value 5
*.a_WR_VALUE_RECEIVE[6]	Reference value 6
*.a_WR_VALUE_RECEIVE[7]	Reference value 7
*.a_WR_VALUE_RECEIVE[8]	Reserved
...	...
*.d_WR_VALUE_RECEIVE[15]	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

The reference values can be word or doubleword reference values.

Explanation of the use of reference value channels

In synchronous operation, two reference value channels (channels 1 and 2; also WRC1 and WRC2) are available.

Both reference value message frames consist of four words each (W0 to W3). After CANsync initialization, you must state at least once the assignment of these words.

This can be carried out in the initialization program. To do this, enter in areas *._CFG_WRC_WORD.b_CFG_WRC1_WORD0 to *._CFG_WRC_WORD.b_CFG_WRC2_WORD3 for each word of the reference value message frames the reference value that is to be received at this location. Valid reference value numbers are in the range 0 to 7. For doubleword reference values, you must use two words in the message frame.

You can change the configuration even during active operation. The system applies the change in the next CANsync interval at the latest. At the start of the CANsync interval, the CANsync interface module reads the configuration data.

The system carries out reference value message frame evaluation at the start of the CANsync interval. For this, you poll the status registers (control registers) of the reference value channels (*.b_CTRLREG_WRC1 to *.b_CTRLREG_WRC2). The system enters 16#02 in these registers if the corresponding reference value message frame was received. Then, you can read out the reference values from registers *.a_WR_VALUE_RECEIVE[0] to *.a_WR_VALUE_RECEIVE[7] and set the status register to 16#00, for example, to be able to detect correctly the next entry.

You must assign in accordance with the reference value configuration in the application program the reference values that must be read when a reference value message frame has been received.

If a reference value message frame fails that contains a position reference value from the virtual leading axle, for example, the application program must carry out extrapolation starting from the last position reference value. If the reference value fails several times in succession, a fault response must be triggered.

Actual values of the CANsync slave

Actual values are sent in the CANsync event task.

Register	Contents
*.b_CTRLREG_RD_RDC1	Control register of actual value channel 1
*.b_CTRLREG_RD_RDC2	Control register of actual value channel 2
*.b_CTRLREG_RD_RDC3	Reserved
*.b_CTRLREG_RD_RDC4	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

In the control register, the system marks with 16#05 the fact that new actual values for the respective actual value message frame were entered and that the message frame will be sent. The CANsync interface module acknowledges the command with 16#04.

Actual value message frames are configured using the following registers.

Register	Contents
*._CFG_RDC_WORD.b_CFG_RDC1_WORD0	Configuration of actual value message frame 1 word 0
*._CFG_RDC_WORD.b_CFG_RDC1_WORD1	Configuration of actual value message frame 1 word 1
*._CFG_RDC_WORD.b_CFG_RDC1_WORD2	Configuration of actual value message frame 1 word 2
*._CFG_RDC_WORD.b_CFG_RDC1_WORD3	Configuration of actual value message frame 1 word 3
*._CFG_RDC_WORD.b_CFG_RDC2_WORD0	Configuration of actual value message frame 2 word 0
*._CFG_RDC_WORD.b_CFG_RDC2_WORD1	Configuration of actual value message frame 2 word 1
*._CFG_RDC_WORD.b_CFG_RDC2_WORD2	Configuration of actual value message frame 2 word 2
*._CFG_RDC_WORD.b_CFG_RDC2_WORD3	Configuration of actual value message frame 2 word 3
*._CFG_RDC_WORD.b_CFG_RDC3_WORD0	Reserved
*._CFG_RDC_WORD.b_CFG_RDC3_WORD1	Reserved
*._CFG_RDC_WORD.b_CFG_RDC3_WORD2	Reserved
*._CFG_RDC_WORD.b_CFG_RDC3_WORD3	Reserved
*._CFG_RDC_WORD.b_CFG_RDC4_WORD0	Reserved
*._CFG_RDC_WORD.b_CFG_RDC4_WORD1	Reserved
*._CFG_RDC_WORD.b_CFG_RDC4_WORD2	Reserved
*._CFG_RDC_WORD.b_CFG_RDC4_WORD3	Reserved

(* Corresponds, for example, to _CANsync_CTRL_MA).

Your configuration tells the system which actual value is to be entered at which location in the actual value message frame.

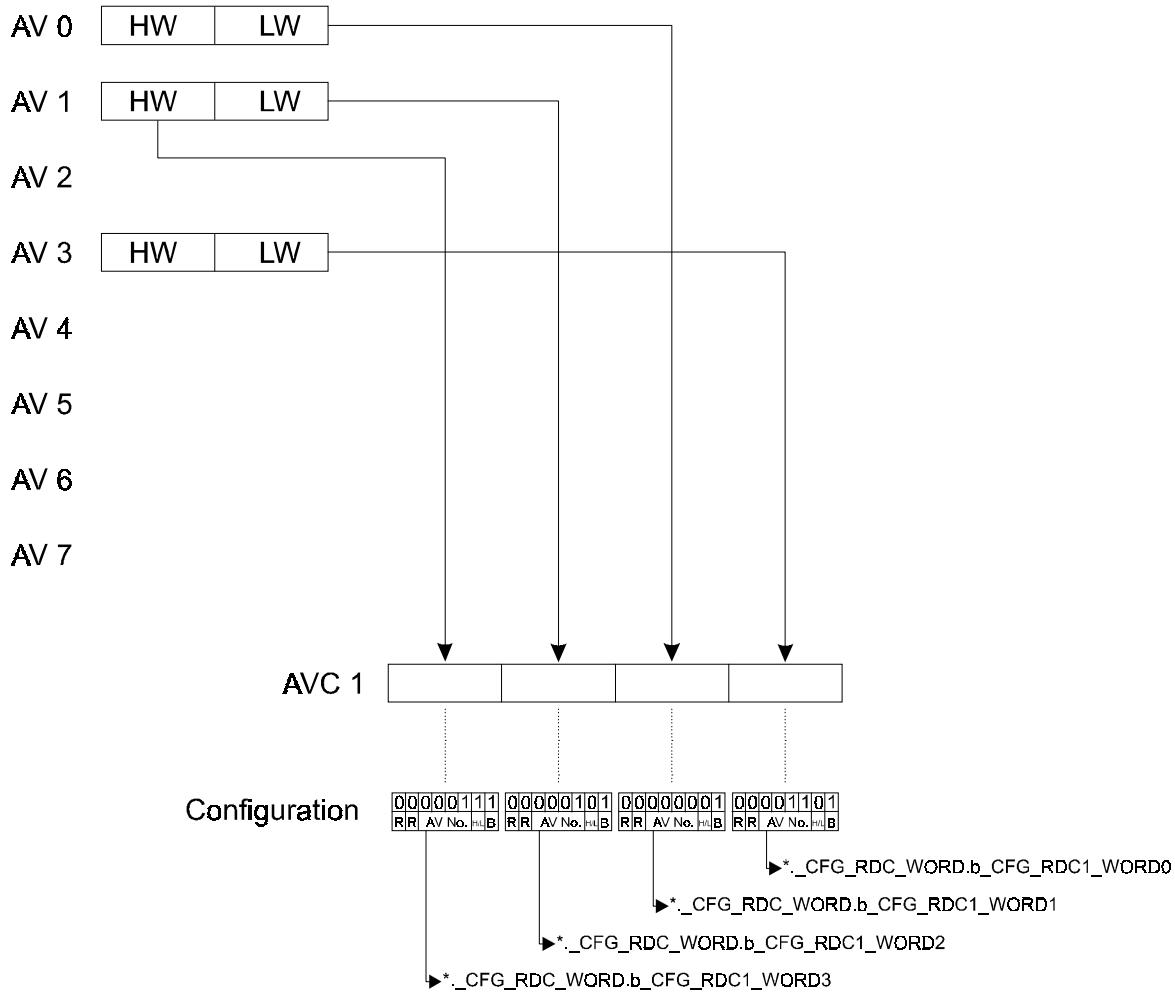
Meaning

Bit 0	**Assigned** If = 1, the system uses this word of the message frame
Bit 1	**Highword/lowword** If = 1, the system enters the highword of the actual value
Bit 2	**Actual value number** Number of actual value 0 to 15
Bit 3	
Bit 4	
Bit 5	
Bit 6	Reserved
Bit 7	

NOTE

One word of the message frame is only not used if the following settings are made: actual value number = 0, highword/lowword = 0 and assigned = 0.

Example:



*._CFG_RDC_WORD.b_CFG_RDC1_WORD0 = 16#0D

0	0	0	0	1	1	0	1
R	R	AV No.				H/L	B

(* Corresponds, for example, to _CANsync_CTRL_SL).

This setting enters the lowword of actual value 3 in actual value message frame 1 in word 0.

Register	Contents
*.a_RD_VALUE_SEND[0]	Actual value 0
*.a_RD_VALUE_SEND[1]	Actual value 1
*.a_RD_VALUE_SEND[2]	Actual value 2
*.a_RD_VALUE_SEND[3]	Actual value 3
*.a_RD_VALUE_SEND[4]	Actual value 4
*.a_RD_VALUE_SEND[5]	Actual value 5
*.a_RD_VALUE_SEND[6]	Actual value 6
*.a_RD_VALUE_SEND[7]	Actual value 7
*.a_RD_VALUE_SEND[8]	Reserved
...	...

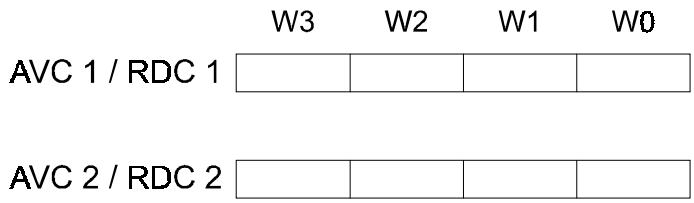
Register	Contents
*.a_RD_VALUE_SEND[15]	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

The actual values can be word or doubleword actual values.

Explanation of the use of actual value channels

In synchronous operation, two actual value channels (channels 1 and 2; also RDC1 and RDC2) are available. Actual value message frames 1 and 2 of this CANsync slave are sent to the CANsync master on actual value channels 1 and 2.



Both actual value message frames consist of four words each (W0 to W3). After CANsync initialization, you must state at least once the assignment of these words.

This can be carried out in the initialization program. To do this, enter in areas *._CFG_RDC_WORD.b_CFG_RDC1_WORD0 to *._CFG_RDC_WORD.b_CFG_RDC2_WORD3 for each word of the actual value message frames the actual value that is to be transferred at this location. Valid actual value numbers are in the range 0 to 7. For doubleword actual values, you must use two words in the message frame.

You can change the configuration even during active operation. The system applies the change in the next CANsync interval at the latest. The system reads the CANsync interface module's configuration data when the corresponding actual value message frame is received.

CANsync slaves cannot themselves trigger sending of their actual values, but rather, the CANsync master uses the identifier of the reference value message frame to request a CANsync slave's actual values.

To ensure that the current actual values are always sent, the actual value entry must be made at the start of the CANsync event task. The actual values for actual value message frame 1 must be entered in registers *.a_RD_VALUE_SEND[0] to *.a_RD_VALUE_SEND[7] by approximately 490 µs after the start of the CANsync event task. This is because the CANsync interface module starts preparing the actual value message frames then. To identify the fact that new actual values have been entered, the system must enter 16#05 in the appropriate control register of the actual value channel *.b_CTRLREG_RD_RDC1 to *.b_CTRLREG_RD_RDC2. This is the enable telling the CANsync interface module that the actual values can be read and that the actual value message frame is being generated. If this enable is not issued by 490 µs after the start of the CANsync event task, actual value message frame 1 is omitted in this CANsync interval. To acknowledge generation of the actual value message frame, the CANsync interface module enters 16#04 in the control register. This allows users to check whether the system finished entering the actual value in good time or not. If generation of the actual value takes a relatively long time – from the start of the CANsync event task to execution of the application program approximately 80 µs expire – the system must always generate the actual value for the next CANsync interval and it only needs to copy the precalculated actual values to the corresponding registers at the start of the new CANsync interval.

The time by which the actual values for actual value message frame 2 must be entered results from the time of reception of reference value message frame 1 plus the processing time of reference value message frame 1. The duration depends on the number of reference value words that have to be entered (a total of between 800 µs and 1000 µs after the start of the CANsync event task). Signalling in the control register is the same as for actual value message frame 1.

Actual values of other CANsync slaves

The actual values of the other CANsync slaves are received in the CANsync event task.

Register	Contents
*.b_STATREG_RDC1	Status register of actual value channel 1
*.b_STATREG_RDC2	Status register of actual value channel 2
*.b_STATREG_RDC3	Reserved
*.b_STATREG_RDC4	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

The system enters in the status register the slave numbers of the CANsync slaves from which an actual value message frame was received.

Register	Contents
*.a_STATREG_RDC[0].b_STATREG_RDC1	Actual value acknowledgement of actual value channel 1 of slave 0
*.a_STATREG_RDC[0].b_STATREG_RDC2	Actual value acknowledgement of actual value channel 2 of slave 0
*.a_STATREG_RDC[0].b_STATREG_RDC3	Reserved
*.a_STATREG_RDC[0].b_STATREG_RDC4	Reserved
*.a_STATREG_RDC[1].b_STATREG_RDC1	Actual value acknowledgement of actual value channel 1 of slave 1
*.a_STATREG_RDC[1].b_STATREG_RDC2	Actual value acknowledgement of actual value channel 2 of slave 1
*.a_STATREG_RDC[1].b_STATREG_RDC3	Reserved
*.a_STATREG_RDC[1].b_STATREG_RDC4	Reserved
...	...
*.a_STATREG_RDC[31].b_STATREG_RDC1	Actual value acknowledgement of actual value channel 1 of slave 31
*.a_STATREG_RDC[31].b_STATREG_RDC2	Actual value acknowledgement of actual value channel 2 of slave 31
*.a_STATREG_RDC[31].b_STATREG_RDC3	Reserved
*.a_STATREG_RDC[31].b_STATREG_RDC4	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

When the corresponding actual value message frame for a CANsync slave has arrived, the system enters 16#02 in the actual value acknowledgement register.

You use the following registers to configure the actual value message frames of the other CANsync slaves.

Register	Contents
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD0	Configuration of actual value message frame 1 word 0 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD1	Configuration of actual value message frame 1 word 1 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD2	Configuration of actual value message frame 1 word 2 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD3	Configuration of actual value message frame 1 word 3 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD0	Configuration of actual value message frame 2 word 0 slave 0

Register	Contents
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD1	Configuration of actual value message frame 2 word 1 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD2	Configuration of actual value message frame 2 word 2 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC2_WORD3	Configuration of actual value message frame 2 word 3 slave 0
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD0	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD1	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD2	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC3_WORD3	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD0	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD1	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD2	Reserved
*.a_CFG_RDC_WORD[0].b_CFG_RDC4_WORD3	Reserved
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD0	Configuration of actual value message frame 1 word 0 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD1	Configuration of actual value message frame 1 word 1 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD2	Configuration of actual value message frame 1 word 2 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC1_WORD3	Configuration of actual value message frame 1 word 3 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD0	Configuration of actual value message frame 2 word 0 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD1	Configuration of actual value message frame 2 word 1 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD2	Configuration of actual value message frame 2 word 2 slave 1
*.a_CFG_RDC_WORD[1].b_CFG_RDC2_WORD3	Configuration of actual value message frame 2 word 3 slave 1
...	...
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD0	Configuration of actual value message frame 1 word 0 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD1	Configuration of actual value message frame 1 word 1 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD2	Configuration of actual value message frame 1 word 2 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC1_WORD3	Configuration of actual value message frame 1 word 3 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD0	Configuration of actual value message frame 2 word 0 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD1	Configuration of actual value message frame 2 word 1 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD2	Configuration of actual value message frame 2 word 2 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3	Configuration of actual value message frame 2 word 3 slave 31
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD0	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD1	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD2	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC3_WORD3	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD0	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD1	Reserved

Register	Contents
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD2	Reserved
*.a_CFG_RDC_WORD[31].b_CFG_RDC4_WORD3	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

Your configuration tells the system which actual value it to be transferred at which position in the actual value message frame (of the other CANsync slave).

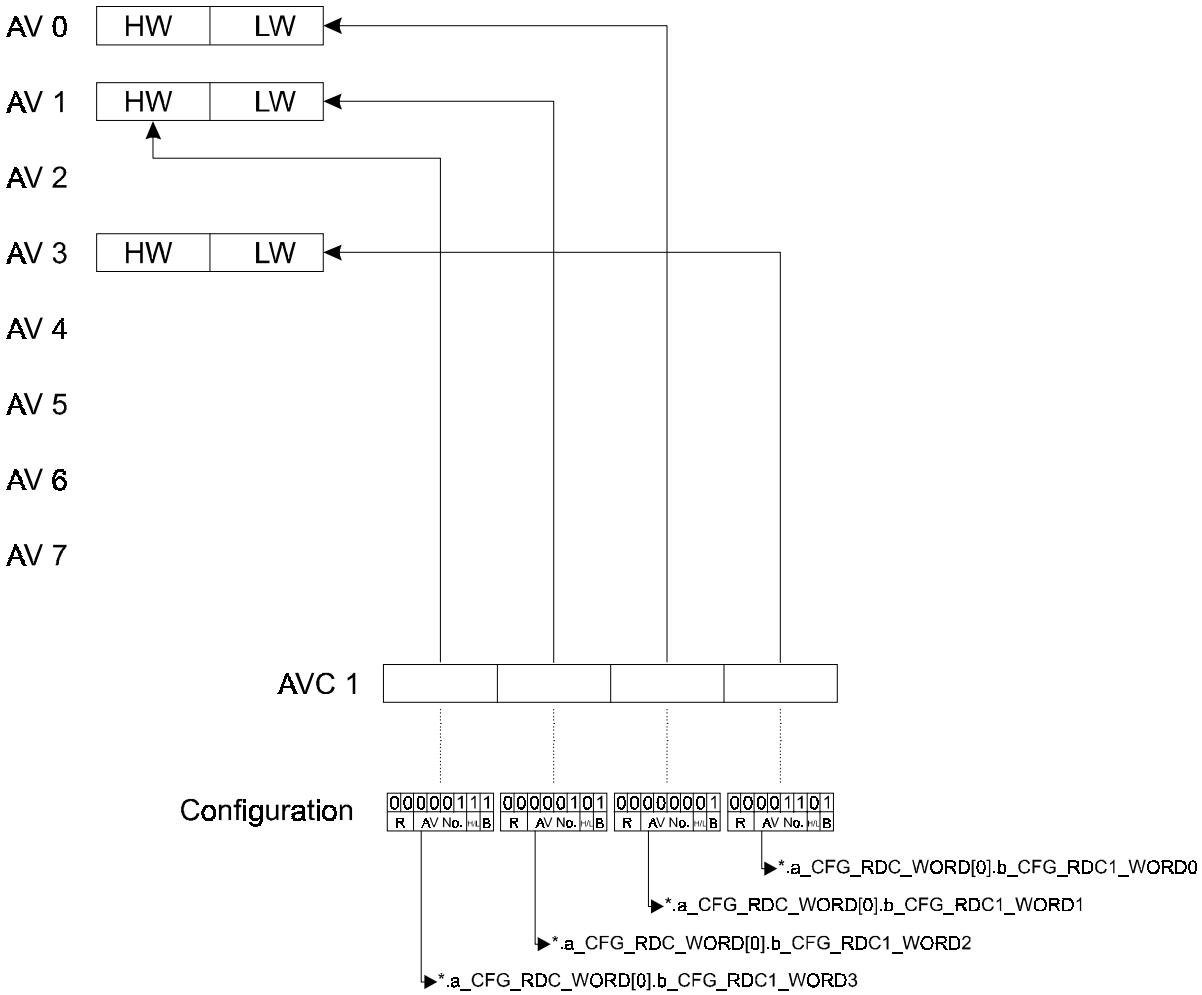
Meaning

Bit 0	**Assigned** If = 1, the system uses this word of the message frame
Bit 1	**Highword/lowword** If = 1, the system enters the highword of the actual value
Bit 2	**Actual value number** Number of actual value 0 to 7
Bit 3	
Bit 4	
Bit 5	
Bit 6	Reserved
Bit 7	

NOTE

One word of the message frame is only not used if the following settings are made: actual value number = 0, highword/lowword = 0 and assigned = 0.

Example:



*.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD0 = 16#0D

0	0	0	0	1	1	0	1
R		AV No.				H/L	B

(* Corresponds, for example, to _CANsync_CTRL_SL).

This means that the system reads word 0 from actual value message frame 1 of CANsync slave 0 and writes it to the lowword of actual value 3.

Register	Contents
*.a_RD_BMARRAY[0][0]	Actual value 0 of slave 0
*.a_RD_BMARRAY[0][1]	Actual value 1 of slave 0
*.a_RD_BMARRAY[0][2]	Actual value 2 of slave 0
*.a_RD_BMARRAY[0][3]	Actual value 3 of slave 0
*.a_RD_BMARRAY[0][4]	Actual value 4 of slave 0
*.a_RD_BMARRAY[0][5]	Actual value 5 of slave 0
*.a_RD_BMARRAY[0][6]	Actual value 6 of slave 0
*.a_RD_BMARRAY[0][7]	Actual value 7 of slave 0
*.a_RD_BMARRAY[0][8]	Reserved
...	...

Register	Contents
*.a_RD_BMARRAY[0][15]	Reserved
*.a_RD_BMARRAY[1][0]	Actual value 0 of slave 1
*.a_RD_BMARRAY[1][1]	Actual value 1 of slave 1
*.a_RD_BMARRAY[1][2]	Actual value 2 of slave 1
*.a_RD_BMARRAY[1][3]	Actual value 3 of slave 1
*.a_RD_BMARRAY[1][4]	Actual value 4 of slave 1
*.a_RD_BMARRAY[1][5]	Actual value 5 of slave 1
*.a_RD_BMARRAY[1][6]	Actual value 6 of slave 1
*.a_RD_BMARRAY[1][7]	Actual value 7 of slave 1
*.a_RD_BMARRAY[1][8]	Reserved
...	...
*.a_RD_BMARRAY[31][0]	Actual value 0 of slave 31
*.a_RD_BMARRAY[31][1]	Actual value 1 of slave 31
*.a_RD_BMARRAY[31][2]	Actual value 2 of slave 31
*.a_RD_BMARRAY[31][3]	Actual value 3 of slave 31
*.a_RD_BMARRAY[31][4]	Actual value 4 of slave 31
*.a_RD_BMARRAY[31][5]	Actual value 5 of slave 31
*.a_RD_BMARRAY[31][6]	Actual value 6 of slave 31
*.a_RD_BMARRAY[31][7]	Actual value 7 of slave 31
*.a_RD_BMARRAY[31][8]	Reserved
...	...
*.a_RD_BMARRAY[31][15]	Reserved

(* Corresponds, for example, to _CANsync_CTRL_SL).

The actual values can be word or doubleword actual values.

NOTE

When displaying _CANsync_CTRL_SL in the PROPROG wt II watch window, a period may be between the square brackets.

Explanation of the use of actual value channels

A CANsync slave can also evaluate all the actual value message frames of the other CANsync slaves. However, it is only possible to request them from the CANsync master.

In synchronous operation, two actual value channels (channels 1 and 2; also RDC1 and RDC2) are available. Actual value message frames 1 and 2 are sent on actual value channels 1 and 2.

Both actual value message frames consist of four words each (W0 to W3). After CANsync initialization, you must state at least once the assignment of these words. This can be carried out in the initialization program.

To do this, you must enter in area *.a_CFG_RDC_WORD[0].b_CFG_RDC1_WORD0 to *.a_CFG_RDC_WORD[31].b_CFG_RDC2_WORD3 for each assigned word of the actual value message frames (of each CANsync slave) the actual value of the other CANsync slaves that is transferred at this position. Valid actual value numbers are in the range 0 to 7. For doubleword actual values, you must use two words in the message frame.

You can change the configuration even during active operation. The system applies the change in the next CANsync interval at the latest. The CANsync interface module reads the configuration data when the corresponding actual value message frame is received.

The system carries out actual value message frame evaluation at the start of the CANsync event task. There are two evaluation methods: With the first one, you poll the status registers of the actual value channels (*.b_STATREG_RDC1 or *.b_STATREG_RDC2). The system enters in these registers the slave number of the CANsync slave from which the corresponding actual value message frame was received. You can then read out the actual values from registers (*.a_RD_BMARRAY[0][0] to *.a_RD_BMARRAY[31][7]) of this CANsync slave and set the status register to 16#FF, for example, to be able to detect correctly the next entry.

With the second method, you poll directly actual value acknowledgement (*.a_STATREG_RDC[0].b_STATREG_RDC1 to *.a_STATREG_RDC[31].b_STATREG_RDC2) for one CANsync slave (whose actual value message frames are to be monitored). In this acknowledgement register, the system marks with 16#02 reception of an actual value message frame. You can then read out the actual values from registers (*.a_RD_BMARRAY[0][0] to *.a_RD_BMARRAY[31][7]) of this CANsync slave. In this case too, the system must then write another value to the status register to detect reentry of the acknowledgement.

You must assign in accordance with the actual value configuration in the application program the actual values that must be read when an actual value message frame has been received.

The system uses the CANsync master's reference value message frame to request a CANsync slave to send its actual value message frame. You can only make the setting on the CANsync master.

Command and Response Channel

Action command

Action commands are reported in the following registers.

Register	Contents
*.b_STATREG_AKT_CMD	Status register of action command
*.b_AKT_CMD	Action command
*.b_DATA_BYTE_AKT_CMD	Data byte 0 (DB)
*.w_DATAWORD_AKT_CMD	Data word 1 (DW)
*.b_STATREG_CONTROLWORD	Status register of control word
*.w_CONTROLWORD	Control word
*.b_STATREG_SYNC_MODUS	Status register of SYNC mode
*.b_SYNC_MODUS	SYNC mode

(* Corresponds, for example, to _CANsync_CTRL_SL).

The action command that the CANsync master transmits as a broadcast command is only reported if the CANsync slave is activated, i.e. if the corresponding bit is set in the bit strip (See “Register Structure and Function of the Omega CANsync Master” on page 138.).

If the action command is a control word command (command byte = 16#01), the system enters the command's data word in *.w_CONTROLWORD as the control word and enters 16#02 in *.b_STATREG_CONTROLWORD as the reception indicator.

With all the other action commands, the system enters the data in *.b_DATA_BYTE_AKT_CMD and *.w_DATAWORD_AKT_CMD. In this case, the acknowledgement (reception indicator 16#02) is in *.b_STATREG_AKT_CMD. Since a new action command can be transferred in every CANsync interval, you should evaluate the action command's status register in every CANsync event task.

Parameter command

Parameter commands are reported in the following registers.

Register	Contents
*.b_STEUREG_PAR_CMD	Control register of parameter command
*.b_STATREG_PAR_CMD	Status register of parameter command
*.w_ERR_NR_PAR_CMD	Error number of parameter command
*.d_DATA_PAR_CMD	Data of parameter command
*.w_PAR_NR	Parameter number

(* Corresponds, for example, to _CANsync_CTRL_SL).

Meanings of the control and status registers

Bit 0	**Active** Is set to 1 when the job has been received Must be set to 0 when the job has been processed
Bit 1	**Read** =1: Write parameter
Bit 2	**Write** =1: Read parameter
Bit 3	**Change** = 1: Indication that the CANsync master has changed the parameter command; must be acknowledged by reset
Bit 4	**Format** 0: word parameter 1: doubleword parameter
Bit 5	**Error display.** Must be set to 1 if the job generates an error
Bit 6	**Busy** = 1: response has not yet been sent = 0: response has been sent to the CANsync master
Bit 7	**Reset** = 1 confirms that the command change was seen at the change bit and that the old command will not be further-processed.

Sequence of a parameter access

Parameter commands can be processed both in the CANsync event task and in the rest of the program. To ensure data consistency, the system allocates jobs and evaluates the response via the control register and the status register. To guarantee that the job is carried out without conflicts, first the control register and then the status register must always be read from the application program and if a new value is written, the program must always first set the status register and then the control register. (The control register is the crucial one for the CANsync interface module).

When a write parameter job has been received, the value to be written is located in *.d_DATA_PAR_CMD and the parameter number is located in *.w_PAR_NR. In the control register (*.b_STEUREG_PAR_CMD) and in the status register (*.b_STATREG_PAR_CMD), the system reports a write word command with 16#45 and a write doubleword command with 16#55.

If the application program has written the requested parameter error-free, this must be indicated in the status register and the control register by clearing bit 0 (active).

As soon as the response message frame has been sent to the CANsync master, the CANsync interface module clears bit 6 (busy).

If the application program cannot carry out the write access, an error number must be entered in *.w_ERR_NR_PAR_CMD and the system must set in the registers bit 5 (error display) and clear bit 0 (active). Then, a response message frame containing the entered error number is sent to the CANsync master.

As soon as the response message frame has been sent to the CANsync master, the CANsync interface module clears bit 6 (busy).

The read parameter job runs in a similar way with the only differences being that, regardless of the format, the command is 16#43 and that as the response the data must be entered in *.d_DATA_PAR_CMD. Before acknowledging by deleting bit 0 (active), the system must enter the parameter's actual format in bit 4 (format) so that the response to the CANsync master can be generated correctly.

If the CANsync master changes the parameter command (parameter number), the system indicates this by setting bit 3 (change). In this case, the old command must not be responded to. The application program must detect the change and acknowledge this by setting bit 7 (reset) . Then, the current parameter command is entered.

Error code

Value	Meaning
16#0000	No error occurred
16#FFFF	Error occurred
16#FFFE	Value less than minimum value
16#FFFD	Value greater than maximum value
16#FFFC	Element cannot be changed
16#FFFB	Element not present
16#FFFA	Data is not available (e.g. being processed)
16#FFF9	Error in data format

Upload/download command

Upload and download commands are displayed in the following registers.

Register	Contents
*.b_STEUREG_UPDOWNBLK0	Reserved
*.b_STATREG_UPDOWNBLK0	Reserved

Register	Contents
*.w_ERR_NR_UPDOWNBLK0	Reserved
*.w_SUBSL_NR_UPDOWNBLK0	Reserved
*.d_BASE_ADR_UPDOWNBLK0	Reserved
*.w_LENGTH_UPDOWNBLK0	Reserved
*.w_COUNTER_UPDOWNBLK0	Reserved
*.a_DATA_UPDOWNBLK0[0 to 74]	Reserved
*.b_STEUREG_UPDOWNBLK1	Control register of upload/download block 1
*.b_STATREG_UPDOWNBLK1	Status register of upload/download block 1
*.b_EN_UPDOWNBLK1	Enable of block 1
*.w_ERR_NR_UPDOWNBLK1	Error number of upload/download block 1
*.d_BASE_ADR_UPDOWNBLK1	Base address of upload/download block 1
*.w_LENGTH_UPDOWNBLK1	Length in bytes of upload/download block 1
*.w_COUNTER_UPDOWNBLK1	Counter of upload/download block 1
*.a_DATA_UPDOWNBLK1[0 to 74]	Data block of upload/download block 1

(* Corresponds, for example, to _CANsync_CTRL_SL).

NOTE

Block 0 is reserved for the OmegaOS.

Meanings of the control and status registers

Bit 0	**Active** Is set to 1 when the job has been received Must be set to 0 when the job has been processed
Bit 1	**Change** = 1: Indication that the CANsync master has changed the parameter command; must be acknowledged by reset
Bit 2 Bit 3	**Mode** Bit3 bit2 0 0: reserved 0 1: initialization 1 0: ongoing upload/download 1 1: End of block
Bit 4	**Upload/download** = 0: Upload = 1: Download
Bit 5	**Error display** Must be set to 1 if the job generates an error
Bit 6	**Busy** = 1: Response has not yet been sent = 0: Response has been sent to CANsync master
Bit 7	**Reset** = 1 confirms that the command change was seen at the change bit and that the old command will not be further-processed.

Sequence of an upload/download job in block 1

Upload/download jobs for the CANsync slave are received in block 1 or in the single message frame area.

The single message frame area is always used if the block length is greater than 300 bytes (75 double-words). With a shorter block length, the system evaluates the enable on *.b_EN_UPDOWNBLK1. If there is a value that is not equal to zero there, the job is logged on in block 1; otherwise, it is entered in the single message frame area.

The evaluation of the upload/download commands can be used in the CANsync event task as well as in the rest of the program. To ensure data consistency, the system allocates jobs and evaluates the response via the control register and the status register. To guarantee that the job is carried out without conflicts, first the control register and then the status register must always be read from the application program and if a new value is written, the program must always first set the status register and then the control register. (The control register is the crucial one for the CANsync interface module).

When a download is logged on, the system enters the base address in *.d_BASE_ADR_UPDOWNBLK1 and the block length in *.w_LENGTH_UPDOWNBLK1.

Base addresses 16#00000000 to 16#000000FF are reserved for operating system jobs.

The value 16#55 is reported in status register *.b_STATREG_UPDOWNBLK1 and in control register *.b_STEUREG_UPDOWNBLK1. If the download is allowed, the application program must clear bit 0 (active). Then, the CANsync master receives the entire block (you can read off the progress in the byte counter in *.w_COUNTER_UPDOWNBLK1) and at the end the system reports 16#5D in the registers. If the data is taken from area *.a_DATA_UPDOWNBLK1[0] to *.a_DATA_UPDOWNBLK1[74], you must clear bit 0 (active). To indicate that the response message frame has been sent to the CANsync master, the system clears bit 6 (busy).

If an error occurs in the CANsync slave while it is carrying out the job, the system must enter the error number in *.w_ERR_NR_UPDOWNBLK1, set bit 5 (error display) and clear bit 0 (active). Then a corresponding error message frame is sent to the CANsync master, which cancels the download.

Upload jobs run in a similar way with the only differences being that the command is 16#05 and that the data must be entered in the block area before acknowledgement of the initialization message frame.

If the CANsync master changes the upload/download command, the system indicates this by setting bit 1 (change). Then, the old command must be cancelled. The application program must detect the change and acknowledge this by setting bit 7 (reset) . Then, the current upload/download command is entered.

The error numbers that the slave CANsync can enter are above 16#00FF.

Upload/download error numbers that the CANsync slave interface module sends to the CANsync master; display in the CANsync master only:

Error number	Meaning
16#0001	CANsync slave acknowledges wrong block number
16#0002	Entered length greater than 300 bytes
16#0100	CANsync slave expects block with the number that is entered in the counter
16#0101	CANsync slave expects block end
16#0102	CANsync slave does not yet expect block end
16#0103	CANsync slave cancels upload/download
16#0104	Upload/download not possible
16#0105	Base address not allowed
16#0106	Reserved
16#0107	Block length > CANsync slave's maximum block length
16#0108	Message frame mode error (mode not allowed at this stage)

Single message frame area

Register	Contents
*.b_STEUREG_UPDOWN_SINGLE	Control register of upload/download
*.b_STATREG_UPDOWN_SINGLE	Status register of upload/download
*.w_ERR_NR_UPDOWN_SINGLE	Error number of upload/download
*.d_BASE_ADR_UPDOWN_SINGLE	Base address of upload/download
*.w_LENGTH_UPDOWN_SINGLE	Length in bytes of upload/download
*.w_COUNTER_UPDOWN_SINGLE	Counter of upload/download
*.d_DATA_DW_UPDOWN_SINGLE	Data of upload/download
*.w_DATA_W_UPDOWN_SINGLE	Data of upload/download

(* Corresponds, for example, to _CANsync_CTRL_SL).

Meanings of the control and status registers

Bit 0	**Active** Is set to 1 when the job has been received Must be set to 0 when the job has been processed
Bit 1	**Change** = 1: Indication that the CANsync master has changed the parameter command; must be acknowledged by reset
Bit 2 Bit 3	**Mode** Bit3 bit2 0 0: reserved 0 1: initialization 1 0: ongoing upload/download 1 1: End of block
Bit 4	**Upload/download** = 0: Upload = 1: Download
Bit 5	**Error display** Must be set to 1 if the job generates an error
Bit 6	**Busy** = 1: Response has not yet been sent = 0: Response has been sent to CANsync master
Bit 7	**Reset** = 1 confirms that the command change was seen at the change bit and that the old command will not be further-processed.

The single message frame area is used in a similar way to block 1 with the only differences being that each message frame must be acknowledged individually and that the corresponding mode must be entered.

7.3 CANsync Function Blocks

7.3.1 Function Blocks for the Synchronized CAN Overview

In addition to the standard functions, you can use manufacturer-defined functions if you have logged on libraries in a project.

Note: Logging on of libraries is described in the general help.

The following function blocks for synchronized CAN are available:

Function	Brief description
CANsync_BC_MA0	Sends the CANsync master's broadcast command in transmission range 0 (highest priority)
CANsync_BC_MA1	Sends the CANsync master's broadcast command in transmission range 1 (medium priority)
CANsync_BC_MA2	Sends the CANsync master's broadcast command in transmission range 2 (low priority)
CANsync_BC_SL	Receives the CANsync master's broadcast commands
CANsync_COMM_CONTROL_MA	Configures use of a CANsync master interface module's command channel
CANsync_CONTROLWORD_MA	Sends the CANsync-Master's control word commands
CANsync_CONTROLWORD_SL	Receives the CANsync-Master's control word in a CANsync slave interface module
CANsync_INIT	Initializes a CANsync interface module
CANsync_MODE_MA	Sets the operating mode of a CANsync master interface module
CANsync_MODE_SL	Sets the operating mode of a CANsync slave interface module
CANsync_PAR_READ_MA	The Ωmega-Master requests via CANsync a parameter value from the Ωmega-Slave
CANsync_PAR_SL	Detects the parameter request or the transferred parameter
CANsync_PAR_WRITE_MA	The Ωmega-Master sends via CANsync a parameter value to the Ωmega-Slave
CANsync_PD_CFG_MA	Configures assignment of the CAN interface module's reference value channels for a CANsync master
CANsync_PD_CFG_READ_MA	Configures assignment of the CAN interface module's actual value channels for a CANsync master
CANsync_PD_CFG_READ_SL	Configures assignment of the CAN interface module's actual value channels for a CANsync slave
CANsync_PD_CFG_SL	Configures assignment of the CAN interface module's reference value and actual value channels for a CANsync slave
CANsync_PD_COMM_MA	Copies the process data (CANsync interface module's reference values and actual values) for a CANsync master
CANsync_PD_COMM_READ_MA	Copies in a (master) CANsync interface module the process data actual values of a CANsync slave
CANsync_PD_COMM_READ_SL	Copies in a (slave) CANsync interface module the process data actual values of a CANsync slave
CANsync_PD_COMM_SL	Copies the process data (reference values and actual values) for a CANsync slave
CANsync_SL_TYP_INIT	CANsync slave types (initialization)
CANsync_UPDOWNLOAD_MA	CANsync upload download Master
CANsync_UPDOWNLOAD_SL	CANsync upload download slave

7.3.2 CANsync_BC_MA0

Description

You can use this function block for CANsync to send a CANsync-Master broadcast command in transmission range 0 (highest priority).

NOTE

FB CANsync_BC_MA0 uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
us_BC_CMD_NR	USINT	Command number of the broadcast command
si_BC_BYTE	SINT	Data byte of the broadcast command
i_BC_WORD	INT	Data word of the broadcast command
d_SL_MASK	DWORD	Bit strip to which CANsync slaves the broadcast command is to be sent
x_EN	BOOL	Enable

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_OK	BOOL	OK bit

With the CANsync, there are three transmission ranges for broadcast message frames with broadcast commands (transmission ranges 0 to 2). In any one CANsync interval, it is only possible to send one broadcast message frame. Transmission range 0 has the highest priority. FB CANsync_BC_MA0 uses transmission range 0 for sending the broadcast message frame.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input us_BC_CMD_NR:

At input us_BC_CMD_NR, you state the command number of the broadcast command.

Inputs si_BC_BYTE, i_BC_WORD:

At inputs si_BC_BYTE and i_BC_WORD, you must connect the associated data in dependence on the command number (see further below in the list of broadcast commands).

Input d_SL_MASK:

At input d_SL_MASK, you state which of the CANsync slaves is to receive the command. To do this, you must set to TRUE for each CANsync slave the bit number that matches its slave number. For the CANsync slave with slave number 0, you set bit 0 of d_SL_MASK to TRUE; or the CANsync slave with slave number 1, you set bit 1 of d_SL_MASK to TRUE, etc. If you want to address all the CANsync slaves, you must set d_SL_MASK = 16#FFFFFFFF.

Input x_EN:

If Input x_EN is set to TRUE, the system enters the broadcast command for sending. The CANsync interface module then sends the broadcast command in the broadcast message frame to the selected CANsync slaves (input d_SL_MASK). While x_EN is set to TRUE, the system enters the broadcast command for sending every time the FB is called. However, this would mean that the CANsync's command channel would be busy all the time. You should therefore set input x_EN to TRUE for only one CANsync interval to send the broadcast command once.

Output x_OK:

Output x_OK indicates by TRUE that the last broadcast message frame has been sent. Output x_OK is FALSE if no broadcast message frame has been sent.

List of broadcast commands:

Command number	**Meaning**
1	Control word si_BC_BYTE: not used i_BC_WORD: control word
2 to 127	Reserved
128 - 255	Are available for users

7.3.3 CANsync_BC_MA1

Description

You can use this function block for CANsync to send a CANsync-Master broadcast command in transmission range 1 (medium priority).

NOTE

FB CANsync_BC_MA1 uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
us_BC_CMD_NR	USINT	Command number of the broadcast command
si_BC_BYTE	SINT	Data byte of the broadcast command
i_BC_WORD	INT	Data word of the broadcast command
d_SL_MASK	DWORD	Bit strip to which CANsync slaves the broadcast command is to be sent
x_EN	BOOL	Enable

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_OK	BOOL	OK bit

With the CANsync, there are three transmission ranges for broadcast message frames with broadcast commands (transmission ranges 0 to 2). In any one CANsync interval, it is only possible to send one broadcast message frame. Transmission range 1 has the medium priority. FB CANsync_BC_MA1 uses transmission range 1 for sending the broadcast message frame.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input us_BC_CMD_NR:

At input us_BC_CMD_NR, you state the command number of the broadcast command.

Inputs si_BC_BYTE, i_BC_WORD:

At inputs si_BC_BYTE and i_BC_WORD, you must connect the associated data in dependence on the command number (see further below in the list of broadcast commands).

Input d_SL_MASK:

At input d_SL_MASK, you state which of the CANsync slaves is to receive the command. To do this, you must set to TRUE for each CANsync slave the bit number that matches its slave number. For the CANsync slave with slave number 0, you set bit 0 of d_SL_MASK to TRUE; or the CANsync slave with slave number 1, you set bit 1 of d_SL_MASK to TRUE, etc. If you want to address all the CANsync slaves, you must set d_SL_MASK = 16#FFFFFFFF.

Input x_EN:

If Input x_EN is set to TRUE, the system enters the broadcast command for sending. The CANsync interface module then sends the broadcast command in the broadcast message frame to the selected CANsync slaves (input d_SL_MASK). While x_EN is set to TRUE, the system enters the broadcast command for sending every time the FB is called. However, this would mean that the CANsync's command channel would be busy all the time. You should therefore set input x_EN to TRUE for only one CANsync interval to send the broadcast command once.

Output x_OK:

Output x_OK indicates by TRUE that the last broadcast message frame has been sent. Output x_OK is FALSE if no broadcast message frame has been sent.

List of broadcast commands:

Command number	**Meaning**
1	Control word si_BC_BYTE: not used i_BC_WORD: control word
2 - 127	Reserved
128 - 255	Are available for users

7.3.4 CANsync_BC_MA2

Description

You can use this function block for CANsync to send a CANsync-Master broadcast command in transmission range 2 (lowest priority).

NOTE

FB CANsync_BC_MA2 uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
us_BC_CMD_NR	USINT	Command number of the broadcast command
si_BC_BYTE	SINT	Data byte of the broadcast command
i_BC_WORD	INT	Data word of the broadcast command
d_SL_MASK	DWORD	Bit strip to which CANsync slaves the broadcast command is to be sent
x_EN	BOOL	Enable

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_OK	BOOL	OK bit

With the CANsync, there are three transmission ranges for broadcast message frames with broadcast commands (transmission ranges 0 to 2). In any one CANsync interval, it is only possible to send one broadcast message frame. Transmission range 2 has the lowest priority. FB CANsync_BC_MA2 uses transmission range 2 for sending the broadcast message frame.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input us_BC_CMD_NR:

At input us_BC_CMD_NR, you state the command number of the broadcast command.

Inputs si_BC_BYTE, i_BC_WORD:

At inputs si_BC_BYTE and i_BC_WORD, you must connect the associated data in dependence on the command number (see further below in the list of broadcast commands).

Input d_SL_MASK:

At input d_SL_MASK, you state which of the CANsync slaves is to receive the command. To do this, you must set to TRUE for each CANsync slave the bit number that matches its slave number. For the CANsync slave with slave number 0, you set bit 0 of d_SL_MASK to TRUE; or the CANsync slave with slave number 1, you set bit 1 of d_SL_MASK to TRUE, etc. If you want to address all the CANsync slaves, you must set d_SL_MASK = 16#FFFFFFFF.

Input x_EN:

If Input x_EN is set to TRUE, the system enters the broadcast command for sending. The CANsync interface module then sends the broadcast command in the broadcast message frame to the selected CANsync slaves (input d_SL_MASK). While x_EN is set to TRUE, the system enters the broadcast command for sending every time the FB is called. However, this would mean that the CANsync's command channel would be busy all the time. You should therefore set input x_EN to TRUE for only one CANsync interval to send the broadcast command once.

Output x_OK:

Output x_OK indicates by TRUE that the last broadcast message frame has been sent. Output x_OK is FALSE if no broadcast message frame has been sent.

List of broadcast commands:

Command number	**Meaning**
1	Control word si_BC_BYTE: not used i_BC_WORD: control word
2 to 127	Reserved
128 - 255	Are available for users

7.3.5 CANsync_BC_SL

Description

You can use this function block for CANsync to receive a CANsync-Master broadcast command with a CANsync-Slave.

NOTE

FB CANsync_BC_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
us_BC_CMD_NR	USINT	Command number of the broadcast command
si_BC_BYTE	SINT	Data byte of the broadcast command
i_BC_WORD	INT	Data word of the broadcast command
si_BC_RECEIVED	SINT 0, 2	Display indicating that a command was received

This FB indicates at si_BC_RECEIVED that a broadcast command of the CANsync-Master was received and outputs the command number (us_BC_CMD_NR) as well as the contents of the broadcast command (si_BC_BYTE, i_BC_WORD).

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Output us_BC_CMD_NR:

At output us_BC_CMD_NR, the system outputs the command number of the broadcast command.

Outputs si_BC_BYTE, i_BC_WORD:

At outputs si_BC_BYTE and i_BC_WORD, the data must be read that is associated with the respective command number.

Output si_BC_RECEIVED:

At output si_BC_RECEIVED, the system indicates whether a broadcast message frame was received (with a broadcast command). In this case, si_BC_RECEIVED displays a 2. Otherwise, si_BC_RECEIVED displays a 0. The system does not read out the data associated with the broadcast message frame (us_BC_CMD_NR, si_BC_BYTE, i_BC_WORD) until the message frame has been received. Otherwise, the old values continue to be displayed.

List of broadcast commands:

Command number	**Meaning**
1	Control word si_BC_BYTE: not used i_BC_WORD: control word
2 to 127	Reserved
128 - 255	Are available for users

7.3.6 CANsync_COMM_CONTROL_MA

Description

You can use this function block for CANsync to configure the use of a CANsync interface module's command channel.

NOTE

FB CANsync_COMM_CONTROL_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR_CTRL	SINT -128, 0 to 31	Slave number of the CANsync slave with send control word job
x_EN_CTRL	BOOL	Enable for si_SL_NR_CTRL
si_SL_NR_PAR	SINT -128, 0 to 31	Slave number of the CANsync slave with parameter job
x_EN_PAR	BOOL	Enable for si_SL_NR_PAR
si_SL_NR_UDL	SINT -128, 0 to 31	Slave number of the CANsync slave with upload/download job
x_EN_UDL	BOOL	Enable for si_SL_NR_UDL
si_MAX_SL_NR [a)]	SINT -1, 0 to 31	Maximum slave number of automatic incrementing[a)]

[a)] This input corresponds to input si_MAX_SL_NR on FB CANsync_PD_COMM_MA as the maximum slave number for an automatic actual value message frame request.

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module

Using this FB, you state the slave number of the CANsync slave for which the system is to request whether send control word jobs (si_SL_NR_CTRL, x_EN_CTRL), parameter jobs (si_SL_NR_PAR, x_EN_PAR) and upload/download jobs (si_SL_NR_UDL, x_EN_UDL) are present.

NOTE

The CANsync master can send a command message frame in every CANsync interval.

The system processes the various message frames on the command channel in a priority-based sequence:

Message frame type	Priority
Broadcast message frame 0	Highest
Broadcast message frame 1	↑
Broadcast message frame 2	|
Control word message frames	|
Parameter message frames	↓
Upload/download message frames	Lowest

As a result of these priorities, you cannot send another message frame if a higher-priority one is being transmitted. If you send the control word message frame in every CANsync interval, for example, you can never transmit a parameter message frame or an upload/download message frame!

If you need to poll several CANsync slaves for existing jobs, you set the CANsync interface module as follows:

In every CANsync interval the system automatically increments by 1 the slave number of the CANsync slaves for which polling for existing orders is being carried out. This incrementation is carried out up to si_MAX_SL_NR. After this, the system starts with polling for the CANsync slave with slave number 0, etc.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Inputs si_SL_NR_CTRL, x_EN_CTRL:

At input si_SL_NR_CTRL, you state the slave number of the CANsync slave for which the system is to carry out polling for an existing send control word job (FB CANsync_CONTROLWORD_MA).

Entering si_SL_NR_CTRL = -128 tells the system in every CANsync interval to automatically increment by 1 the slave number of the CANsync slave for which the system is to carry out polling for an existing send control word job until si_MAX_SL_NR is reached. After this, the system starts with polling for the CANsync slave with slave number 0, etc.

The default setting is si_SL_NR_CTRL = -128, i.e. automatic incrementing until si_MAX_SL_NR.

The system only applies the setting at si_SL_NR_CTRL (even if it is not assigned) if x_EN_CTRL = TRUE. If x_EN_CTRL = FALSE, the system does not change this part of the command channel configuration.

Inputs si_SL_NR_PAR, x_EN_PAR:

At input si_SL_NR_PAR , you state the slave number of the CANsync slave for which the system is to carry out polling for an existing parameter job (FB CANsync_PAR_WRITE_MA or CANsync_PAR_READ_MA).

Entering si_SL_NR_CTRL = -128 tells the system in every CANsync interval to automatically increment by 1 the slave number of the CANsync slave for which the system is to carry out polling for an existing send parameter job until si_MAX_SL_NR is reached. After this, the system starts with polling for the CANsync slave with slave number 0, etc.

The default setting is si_SL_NR_PAR = -128, i.e. automatic incrementing until si_MAX_SL_NR.

The system only applies the setting at si_SL_NR_PAR (even if it is not assigned) if x_EN_PAR = TRUE. If x_EN_PAR = FALSE, the system does not change this part of the command channel configuration.

Inputs si_SL_NR_UDL, x_EN_UDL:

At input si_SL_NR_UDL, you state the slave number of the CANsync slave for which the system is to carry out polling for an existing upload/download job (FB CANsync_UPDOWNLOAD_MA).

Entering si_SL_NR_CTRL = -128 tells the system in every CANsync interval to automatically increment by 1 the slave number of the CANsync slave for which the system is to carry out polling for an existing upload/download job until si_MAX_SL_NR is reached. After this, the system starts with polling for the CANsync slave with slave number 0, etc.

The default setting is si_SL_NR_UDL = -128, i.e. automatic incrementing until si_MAX_SL_NR.

The system only applies the setting at si_SL_NR_UDL (even if it is not assigned) if x_EN_PAR = TRUE. If x_EN_UDL = FALSE, the system does not change this part of the command channel configuration.

Input si_MAX_SL_NR:

You state the highest slave number of a CANsync slave at input si_MAX_SL_NR. The system keeps polling up to this slave number whether send control word jobs, parameter jobs and upload/download jobs are available. For this, the system sets si_SL_NR_CTRL, si_SL_NR_PAR, si_SL_UDL not assigned and si_EN_CTRL, si_EN_PAR and si_EN_UDL to TRUE.

If si_MAX_SL_NR = -1, the value remains unchanged on the CANsync interface module. The default setting is si_MAX_SL_NR = -1, i.e. the value remains unchanged on the CANsync interface module.

NOTE

This input corresponds to input si_MAX_SL_NR on FB CANsync_PD_COMM_MA. This means that you use input si_MAX_SL_NR on FB CANsync_COMM_CONTROL_MA **or** input si_MAX_SL_NR on FB CANsync_PD_COMM_MA to state the highest slave number of a CANsync slave. You must use only one of the two inputs!

7.3.7 CANsync_CONTROLWORD_MA

Description

You can use this function block for CANsync to send a control word command of the CANsync-Master.

NOTE

FB CANsync_CONTROLWORD_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR_CTRL	SINT 0 to 31	Slave number of the CANsync slaves to which the control word is to be sent
w_CONTROLWORD	WORD	Control word
x_EN	BOOL	Enable

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_OK	BOOL	OK bit

The system transfers to the CANsync interface module the control word (w_CONTROLWORD), the enable for sending (x_EN) and the slave number of the CANsync slave to which the control word is to be sent (si_SL_NR_CTRL). After the CANsync interface module has detected the job (see FB CANsync_COMM_CONTROL_MA), the system sends the control word at the CANsync slave using a control word message frame and acknowledges sending at output x_OK .

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input w_CONTROLWORD:

You connect the control word that is to be sent at input w_CONTROLWORD.

Input x_EN:

If Input x_EN is set to TRUE, the system enters the control word for sending. The CANsync interface module then sends the control word in the control word message frame to CANsync slave si_SL_NR_CTRL. While x_EN stays TRUE, the system enters the control word for sending every time the FB is called. The system carries this out again every time the FB is called while x_EN stays TRUE. However, this would mean that the CANsync's command channel would be busy all the time. You should therefore set input x_EN to TRUE for only one CANsync interval to send the control word once.

Output x_OK:

Output x_OK indicates by TRUE that the last control word message frame has been sent. Output x_OK is FALSE if no control word message frame has been sent.

7.3.8 CANsync_CONTROLWORD_SL

Description

You can use this function block for CANsync to receive a CANsync-Master control word of the CANsync-Master in a CANsync slave interface module.

NOTE

FB CANsync_CONTROLWORD_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
w_CONTROLWORD	WORD	Control word
si_RECEIVED	SINT	Display indicating that a control word was received

If the CANsync slave receives a control word message frame from the CANsync master, FB CANsync_CONTROLWORD_SL outputs the received control word at output w_CONTROLWORD.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Output w_CONTROLWORD:

The system outputs the received control word at output w_CONTROLWORD.

Output si_RECEIVED:

The system displays at output si_RECEIVED whether a control word message frame has been received. In this case, the output displays a 2. Otherwise the system displays a 0 at output si_RECEIVED.

The control word message frame is a special case of the broadcast message frame (with broadcast command number 1).

7.3.9 CANsync_INIT

Description

You can use this function block for CANsync to initialize a CANsync interface module (master or slave).

NOTE

FB CANsync_INIT uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_INIT_BMSTRUCT	Initialization data for the CANsync interface module
x_SL	BOOL	Selection of CANsync slave/CANsync master
x_SYNC_IN	BOOL	Configuration of send/receive SYNC signal
x_SYNC_MODE	BOOL	Set up synchronous operating mode
x_ASYNC_MODE0	BOOL	Reserved
x_ASYNC_MODE1	BOOL	Reserved
a_SL_TYP	BYTE_32_BMARRAY	Initialization data of slave types
b_ACCEPT_CODE	BYTE	Acceptance code
b_ACCEPT_MASK	BYTE	Acceptance Mask
b_BIT_TIMING0	BYTE	Bus timing 0
b_BIT_TIMING1	BYTE	Bus timing 1
us_BAUDRATE	USINT 3, 4, 5	Baud rate
us_SYNC_INTERVAL	USINT 8, 4, 2	CANsync interval (cycle scheme) in ms

Parameter output	Data type	Description
_BASE	CANsync_INIT_BMSTRUCT	Initialization data for the CANsync interface module
w_ERR	WORD	Error word
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CANsync_INIT offers several configuration options for initializing a CANsync interface module. You use the FB when initializing a CANsync master interface module or a CANsync slave interface module. If you are initializing both CANsync master as well as CANsync slave interface modules (→ a CANsync cluster), the FB is used twice with a different input assignment (see further below).

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_INIT_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_INIT_SL  AT  %MB3.100000  :  CANsync_INIT_BMSTRUCT;
```

Where:

`CANsync_INIT_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_INIT_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_INIT_MA  AT  %MB3.200000  :  CANsync_INIT_BMSTRUCT;
```

Where:

`CANsync_INIT_ MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_INIT_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input x_SL:

At input x_SL, you choose whether the system is to initialize the CANsync interface module as a master or a slave.

If x_SL = FALSE, the system initializes the CANsync interface module as a master; if x_SL = TRUE, the system initializes the CANsync interface module as a slave.

Input x_SYNC_IN:

If both CANsync interface modules on the Ωmega Drive-Line II are to be initialized as a CANsync cluster (i.e. a CANsync slave and a CANsync master), you must make the setting that the received SYNC signal of the CANsync slave interface module is to be used as the SYNC signal of the CANsync master interface module.

If you are only operating the CANsync master interface module (not a CANsync cluster), the CANsync master interface module must generate its own SYNC signal and x_SYNC_IN stays FALSE.

When x_SYNC_IN = FALSE, the CANsync master interface module generates its own SYNC signal; when x_SYNC_IN = TRUE, the CANsync master interface module takes the SYNC signal of the CANsync slave interface module.

Inputs x_SYNC_MODE, x_ASYNC_MODE0, x_ASYNC_MODE1:

You set the CANsync interface module's operating mode at these three inputs. Only one of the three inputs may be TRUE.

Setting x_SYNC_MODE = TRUE sets synchronous operation.

Inputs x_ASYNC_MODE0 and x_ASYNC_MODE1 are reserved and are not assigned.

Explanation of the operating mode. See “General” on page 115.

If all three inputs are FALSE, the system only transfers the initialization data to the CANsync interface module and does not set an operating mode. You can set the operating mode using FB CANsync_MODE_MA (CANsync master interface module) or CANsync_MODE_SL (CANsync slave interface module).

NOTE

The operating mode is enabled using FB CANsync_MODE_MA or CANsync_MODE_SL .

Input a_SL_TYP:

This input is only assigned if the CANsync interface module is initialized as a master. Here, you state which CANsync slaves are connected to the CANsync bus. You can also do this using FB CANsync_SL_TYP_INIT.

A variable of data type BYTE_32_BMARRAY is connected at input a_SL_TYP. Data type BYTE_32_BMARRAY is a field of 32 entries of data type byte:

```
BYTE_32_BMARRAY    : ARRAY [0..31] OF BYTE;
```

Example:

```
a_Slave_Typen      : BYTE_32_BMARRAY;
```

Where:

`a_Slave_Typen`	is the variable name with the data type short designation "a" for ARRAY
`BYTE_32_BMARRAY`	is the data type.

Data type BYTE_32_BMARRAY is a field of 32 entries of data type byte:

```
BYTE_32_BMARRAY    : ARRAY [0..31] OF BYTE;
```

In the individual entries of the field, you enter the slave type of the CANsync slave on the CANsync bus. In entry [0], there is the slave type of the CANsync slave with slave number 0; in entry [1] there is the slave type of the CANsync slave with slave number 1, etc.

A 0 in entry [x] means that there is no CANsync slave with slave number x on the CANsync bus.

A value $\neq 0$ in entry [x] means that there is one CANsync slave with slave number x on the CANsync bus.

Meanings of the slave types

Slave type	Meaning
0	No CANsync slave
1	CANsync slave interface module of an Ωmega Drive-Line II, V-controller with CANsync interface
2 - 255	Reserved

Inputs b_ACCEPT_MASK, b_ACCEPT_CODE:

At inputs b_ACCEPT_MASK and b_ACCEPT_CODE, you can set the acceptance filter of the CANsync interface module. If the inputs are not assigned, this yields default settings b_ACCEPT_MASK = 16#FF and b_ACCEPT_CODE = 16#FF, i.e. all the objects are taken into account.

No other settings are needed with the CANsync.

These inputs are present for reasons of compatibility.

Inputs b_BIT_TIMING0, b_BIT_TIMING1, us_BAUDRATE, us_SYNC_INTERVAL:

At input us_BAUDRATE, you set the baud rate for the CANsync bus. You must set the maximum baud rate that all the nodes on the CANsync bus can "understand".

NOTE

For limitations on the baud rate, refer to the respective technical description.

The bus timing is calculated for three different baud rates and it is transferred to the CANsync interface module by FB CANsync_INIT.

Baud rate	us_BAUDRATE	b_BIT_TIMING0	b_BIT_TIMING1
125 kbps	3	16#03	16#1C
250 kbps	4	16#01	16#1C
500 kbps	5	16#00	16#1C

If value us_BAUDRATE is less than 3 or greater than 5, the system sets the baud rate to 125 kbps and sets bit 1 in error word w_ERR to TRUE.

At inputs b_BIT_TIMING0 and b_BIT_TIMING1, you can set individually the bus timing of the CANsync interface module. For the values for this, refer to the respective technical description.

The system applies the settings of these inputs when input us_BAUDRATE = 0.

The system ignores the settings of these inputs when input us_BAUDRATE is assigned with a value from 3 to 6.

The default setting is us_BAUDRATE = 0, i.e. if us_BAUDRATE is not assigned, the system applies the settings of b_BIT_TIMING0 and b_BIT_TIMING1.

NOTE

The system only applies the values at inputs b_BIT_TIMING0 and b_BIT_TIMING1 if input us_BAUDRATE = 0 or it is not assigned.

At input us_SYNC_INTERVAL, you state the duration in milliseconds of the CANsync interval and the CANsync cycle time.

Together with input us_BAUDRATE, the following combinations are allowed:

CANsync baud rate and CANsync interval duration	us_BAUDRATE	us_SYNC_INTERVAL
500 kbps and 2 ms	5	2
250 kbps and 4 ms	4	4
125 kbps and 8 ms	3	8

Output x_OK:

The system sets output x_OK to TRUE if the CANsync interface module was initialized successfully. Output x_OK stays FALSE if the CANsync interface module was not initialized or an error occurred at initialization.

Outputs x_ERR, w_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error word b_ERR. In this case output x_OK stays FALSE.

Error word w_ERR:

Bit No.	Error
0	Timeout handshaking with the CANsync interface module
1	Input error with b_BIT_TIMING0, b_BIT_TIMING1 or us_BAUDRATE
2 - 10	Reserved
11	Initialization of CANsync interface module not completed
12 - 15	Reserved

7.3.10 CANsync_MODE_MA

Description

You can use this function block for CANsync to set the operating mode of a CANsync interface module.

NOTE

FB CANsync_MODE_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_RESET_SOFTWARE	BOOL	Software reset
x_SET_INIT_DATA	BOOL	Take over initialization data
x_CANsync_RUN	BOOL	Enable active operation
x_RESET_CANsync_CONTROLLER	BOOL	Reset CANsync controller (bus-off reset)
x_SYNC_MODE	BOOL	Set up synchronous operating mode
x_ASYNC_MODE0	BOOL	Reserved
x_ASYNC_MODE1	BOOL	Reserved

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_HS_ACTIV	BOOL	Handshake is active
x_INIT_POSSIBLE	BOOL	Initialization possible
x_WAIT	BOOL	Waiting for command for setting the operating mode
x_PREPARE_ACTIV	BOOL	Setting the operating mode is active
x_CANsync_ACTIV	BOOL	Operation active
x_SYNC_MODE_ACTIV	BOOL	Synchronous operation
x_ASYNC_MODE_ACTIV	BOOL	Asynchronous operation
x_SL	BOOL	Slave operation
x_MA	BOOL	Master operation

FB CANsync_MODE_MA makes it possible to set the operating modes on the CANsync interface module. The inputs correspond to commands. The outputs display the current actual status. The signals are each active when they are TRUE. If all the inputs are FALSE, the system does not execute any commands and the last status stays active.

NOTE

FB CANsync_MODE_MA does not wait for the CANsync interface module's status message. This means that if the FB is called in the cold and warm boot task, the outputs may not be set.
If you need a display of the status, the FB must be called again. The inputs must then be set to FALSE. The CANsync interface module's status is then displayed at the outputs.

Use in the cold and warm boot task:

It is possible to start an operating mode. To do this, you set an operating mode (e.g. set x_SYNC_MODE to TRUE for synchronous operation).

The set operating mode is started with x_CANsync_RUN = TRUE.

Use in the cyclical program:

The CANsync interface module can be reinitialized. To do this, you reset the CANsync interface module with x_RESET_SOFTWARE = TRUE (FB CANsync_MODE_MA).

After this, the system carries out reinitialization of the CANsync interface module (FB CANsync_INIT) and sets and enables an operating mode (FB CANsync_MODE_MA).

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input x_RESET_CANsync_CONTROLLER:

With x_RESET_CANsync_CONTROLLER = TRUE, you reset the CANsync controller. This causes the CANsync controller to leave BUS-OFF status and it can be active again on the CANsync bus.

Input x_SET_INIT_DATA:

With x_SET_INIT_DATA = TRUE, the CANsync interface module applies new initialization data.

This input is only needed if you explicitly program initialization without FB CANsync_INIT.

Input x_CANsync_RUN:

With x_CANsync_RUN = TRUE, you activate the operating mode set under x_SYNC_MODE, x_ASYNC_MODE0 or x_ASYNC_MODE1.

Inputs x_SYNC_MODE, x_ASYNC_MODE0, x_ASYNC_MODE1:

You set the CANsync interface module's operating mode at these three inputs. Only one of the three inputs may be TRUE.

Setting x_SYNC_MODE = TRUE sets synchronous operation.

Inputs x_ASYNC_MODE0 and x_ASYNC_MODE1 are reserved and stay FALSE.

After setting the operating mode, you use x_CANsync_RUN = TRUE to enable active operation.

The system issues a TRUE checkback signal at the outputs. Otherwise, the outputs are FALSE.

Output x_HS_ACTIV:
Output x_HS_ACTIV indicates with TRUE that handshaking is active. This is used by FB CANsync_INIT.

Output x_INIT_POSSIBLE:
Output x_INIT_POSSIBLE indicates with TRUE that the CANsync interface module is in the initialization status. The module can then receive new initialization data or the command for setting the operating mode.

Output x_WAIT:
Output x_WAIT indicates with TRUE that the CANsync interface module has applied the initialization data and is waiting for the command for setting the operating mode.

Output x_PREPARE_ACTIV:
Output x_PREPARE_ACTIV indicates with TRUE that an operating mode is being set up.

Output x_CANsync_ACTIV:
Output x_CANsync_ACTIV indicates with TRUE that an operating mode is active.

Output x_SYNC_MODE_ACTIV:
Output x_SYNC_MODE_ACTIV indicates with TRUE that synchronous operation has been set up.

Output x_ASYNC_MODE_ACTIV:
Output x_ASYNC_MODE_ACTIV indicates with TRUE that asynchronous operation (Mode 0 or Mode 1) has been set up.

Output x_SL:
Output x_SL indicates with TRUE that the CANsync interface module has been configured as a slave.

Output x_MA:
Output x_MA indicates with TRUE that the CANsync interface module has been configured as a master.

It is also possible to set combinations of outputs. If x_CANsync_ACTIV, x_SYNC_MODE_ACTIV and x_MA are set to TRUE, for example, this means that the CANsync interface module is a master in active synchronous operation.

7.3.11CANsync_MODE_SL

Description

You can use this function block for CANsync to set the operating mode of a CANsync interface module.

NOTE

FB CANsync_MODE_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_RESET_SOFTWARE	BOOL	Software reset
x_SET_INIT_DATA	BOOL	Take over initialization data
x_CANsync_RUN	BOOL	Enable active operation
x_RESET_CANsync_CONTROLLER	BOOL	Reset CANsync controller (bus-off reset)
x_SYNC_MODE	BOOL	Set up synchronous operating mode
x_ASYNC_MODE0	BOOL	Reserved
x_ASYNC_MODE1	BOOL	Reserved

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_HS_ACTIV	BOOL	Handshake is active
x_INIT_POSSIBLE	BOOL	Initialization possible
x_WAIT	BOOL	Waiting for command for setting the operating mode
x_PREPARE_ACTIV	BOOL	Setting the operating mode is active
x_CANsync_ACTIV	BOOL	Operation active
x_SYNC_MODE_ACTIV	BOOL	Synchronous operation
x_ASYNC_MODE_ACTIV	BOOL	Asynchronous operation
x_SL	BOOL	Slave operation
x_MA	BOOL	Master operation

FB CANsync_MODE_SL makes it possible to set the operating modes on the CANsync interface module. The inputs correspond to commands. The outputs display the current actual status. The signals are each active when they are TRUE. If all the inputs are FALSE, the system does not execute any commands and the last status stays active.

NOTE

FB CANsync_MODE_SL does not wait for the CANsync interface module's status message. This means that if the FB is called in the cold and warm boot task, the outputs may not be set.
If you need a display of the status, the FB must be called again. The inputs must then be set to FALSE. The CANsync interface module's status is then displayed at the outputs.

Use in the cold and warm boot task:

It is possible to start an operating mode. To do this, you set an operating mode (e.g. set x_SYNC_MODE to TRUE for synchronous operation).

The set operating mode is started with x_CANsync_RUN = TRUE.

Use in the cyclical program:

The CANsync interface module can be reinitialized. To do this, you reset the CANsync interface module with x_RESET_SOFTWARE = TRUE (FB CANsync_MODE_SL).

After this, the system carries out reinitialization of the CANsync interface module (FB CANsync_INIT) and sets and enables an operating mode (FB CANsync_MODE_SL).

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Input x_RESET_CANsync_CONTROLLER:

With x_RESET_CANsync_CONTROLLER = TRUE, you reset the CANsync controller. This causes the CANsync controller to leave BUS-OFF status and it can be active again on the CANsync bus.

Input x_SET_INIT_DATA:

With x_SET_INIT_DATA = TRUE, the CANsync interface module applies new initialization data.

This input is only needed if you explicitly program initialization without FB CANsync_INIT.

Input x_CANsync_RUN:

With x_CANsync_RUN = TRUE, you activate the operating mode set under x_SYNC_MODE, x_ASYNC_MODE0 or x_ASYNC_MODE1.

Inputs x_SYNC_MODE, x_ASYNC_MODE0, x_ASYNC_MODE1:

You set the CANsync interface module's operating mode at these three inputs. Only one of the three inputs may be TRUE.

Setting x_SYNC_MODE = TRUE sets synchronous operation.

Inputs x_ASYNC_MODE0 and x_ASYNC_MODE1 are reserved and stay FALSE.

After setting the operating mode, you use x_CANsync_RUN = TRUE to enable active operation.

The system issues a TRUE checkback signal at the outputs. Otherwise, the outputs are FALSE.

Output x_HS_ACTIV:
Output x_HS_ACTIV indicates with TRUE that handshaking is active. This is used by FB CANsync_INIT.

Output x_INIT_POSSIBLE:

Output x_INIT_POSSIBLE indicates with TRUE that the CANsync interface module is in the initialization status. The module can then receive new initialization data or the command for setting the operating mode.

Output x_WAIT:
Output x_WAIT indicates with TRUE that the CANsync interface module has applied the initialization data and is waiting for the command for setting the operating mode.

Output x_PREPARE_ACTIV:
Output x_PREPARE_ACTIV indicates with TRUE that an operating mode is being set up.

Output x_CANsync_ACTIV:
Output x_CANsync_ACTIV indicates with TRUE that an operating mode is active.

Output x_SYNC_MODE_ACTIV:
Output x_SYNC_MODE_ACTIV indicates with TRUE that synchronous operation has been set up.

Output x_ASYNC_MODE_ACTIV:
Output x_ASYNC_MODE_ACTIV indicates with TRUE that asynchronous operation (Mode 0 or Mode 1) has been set up.

Output x_SL:
Output x_SL indicates with TRUE that the CANsync interface module has been configured as a slave.

Output x_MA:
Output x_MA indicates with TRUE that the CANsync interface module has been configured as a master.

It is also possible to set combinations of outputs. If x_CANsync_ACTIV, x_SYNC_MODE_ACTIV and x_SL are set to TRUE, for example, this means that the CANsync interface module is a slave in active synchronous operation.

7.3.12 CANsync_PAR_READ_MA

Description

You can use this function block for CANsync for the CANsync master to request a parameter value from the CANsync slave via the CANsync.

NOTE

You can instantiate this FB several times if different CANsync slaves are addressed in each case.

FB CANsync_PAR_READ_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR	SINT 0 to 31	Slave number of the CANsync slave to which the read parameter job is addressed
u_PAR_NR	UINT	Parameter number
x_PAR_FORMAT	BOOL	Parameter format
i_SUB_SL	INT 0 to 31	Sub-slave address (reserved)
t_TIME	TIME	Monitoring time
x_EN	BOOL	Enable
x_RESET	BOOL	Reset:

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
ud_PAR_VALUE	UDINT	Read parameter value
x_PAR_FORMAT_READ	BOOL	Read parameter format
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
i_ERR	INT	Error word
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CANsync_PAR_READ_MA transfers with the values of inputs u_PAR_NR, x_PAR_FORMAT and i_SUB_SL a read parameter job to the CANsync slave with slave number si_SL_NR. The CANsync slave processes the read parameter job and returns the result of communication. The read parameter value is output at output ud_PAR_VALUE.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input si_SL_NR:

At input si_SL_NR, you state the slave number of the CANsync slave on the CANsync bus from which the parameter value is read.

Input u_PAR_NR:

You state the parameter number for the read parameter job at input u_PAR_NR.

Input x_PAR_FORMAT:

At input x_PAR_FORMAT, you set the format of the requested parameter value. x_PAR_FORMAT = FALSE means word format, x_PAR_FORMAT = TRUE means doubleword format.

Input i_SUB_SL:

This input is reserved and is not assigned.

Input t_TIME:

At input t_TIME, you state the monitoring time within which the system is to carry out the read parameter job. If the read parameter job is not completed within the monitoring time, the system sets bit 1 of error byte b_ERR to TRUE.

The default setting for t_TIME is 3 s.

Input x_EN:

Communication is started by means of x_EN = TRUE. Input x_EN must not be reset to FALSE until output x_BUSY drops to FALSE after communication is completed. Otherwise, it is assumed that communication was cancelled deliberately and you must reset the FB (x_RESET = TRUE).

Input x_RESET:

FB CANsync_PAR_READ_MA is reset by means of x_RESET = TRUE. This is necessary after aborting communication (by means of x_EN = FALSE) or after an error message, for example. After this, you must set x_RESET back to FALSE.

Output ud_PAR_VALUE:

The system makes available the read parameter value at output ud_PAR_VALUE.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

Output x_OK is set to TRUE if the read parameter job was executed correctly. Output x_OK is FALSE if the system did not execute a read parameter job or it was not executed correctly.

Outputs x_ERR, b_ERR, i_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR. In this case output x_OK stays FALSE.

If the CANsync slave reports an error number, the system outputs an error word at output i_ERR. The contents of the error word is determined by the application in the CANsync slave.

Error byte b_ERR:

Bit No.	Error
0	Communications error, error number is in i_ERR
1	Timeout
2, 3	Reserved
4	Invalid slave number of the CANsync slaves on the CANsync bus
5 - 7	Reserved

7.3.13CANsync_PAR_SL

Description

You can use this function block for CANsync to detect a read parameter job or a write parameter job. The FB is suitable for use with BAPS requirements data FBs BAPS_PAR_READ and BAPS_PAR_WRITE.

NOTE

FB CANsync_PAR_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
ud_PAR_VALUE_READ	UDINT	Parameter value (read)
x_PAR_FORMAT_READ	BOOL	Parameter format (read)
i_ERR	INT	Error number (application)
x_ERR_IN	BOOL	Error bit (application)
x_OK_IN	BOOL	OK bit (application)
x_EN	BOOL	Enable
x_RESET	BOOL	Reset:

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_RESET_JOB	BOOL	Reset (change of job)
x_READ	BOOL	Read parameter job
x_WRITE	BOOL	Write parameter job
u_PAR_NR	UINT	Parameter number
x_PAR_FORMAT_WRITE	BOOL	Parameter format (write)
ud_PAR_VALUE_WRITE	UDINT	Parameter value (write)
x_ACTIV	BOOL	Job logged on, waiting for result of application
x_BUSY	BOOL	Communication is active
x_OK	BOOL	OK bit

FB CANsync_PAR_SL detects a read parameter job or a write parameter job and makes available the respective data at the outputs. The application processes the jobs and transfers the results to the FB via the inputs. The FB then reports the results to the CANsync slave interface module and indicates that the results were sent to the CANsync master.

In the case of a read parameter job, the system indicates the job with x_READ = TRUE and outputs the parameter number at u_PAR_NR. The application expects the parameter value at ud_PAR_VALUE_READ and the parameter format at x_PAR_FORMAT_READ. With x_OK_IN = TRUE, the system takes the parameter value and the parameter format and sends them to the CANsync

master. If the application reports an error, the system can connect an error number at i_ERR and set x_ERR_IN = TRUE. In this case, the error number is sent to the CANsync master.

In the case of a write parameter job, the system indicates the job with x_WRITE = TRUE, and outputs the parameter number at u_PAR_NR, the parameter format at x_PAR_FORMAT_WRITE and the parameter value at ud_PAR_VALUE_WRITE. The application expects the result of communication. With x_OK_IN = TRUE, the system reports to the CANsync master that the write parameter job was executed successfully. If the application reports an error, the system can connect an error number at i_ERR and set x_ERR_IN = TRUE. In this case, the error number is sent to the CANsync master.

NOTE

In the Ωmega Drive-Line II with a CANsync slave interface module, it is possible to pass on read parameter jobs and write parameter jobs to the V-controller. The FBs of BAPS requirements data communication are used for this.
In the following section, some of the inputs and outputs have listed in brakkets the respective input or output of the FBs of BAPS-requirements data communication.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Input ud_PAR_VALUE_READ:

In the case of a read parameter job, the system expects the parameter value at ud_PAR_VALUE_READ. The application program makes available this parameter value. (Input ud_PAR_VALUE_READ can be connected to FB BAPS_PAR_READ, output ud_PAR_VALUE).

Input x_PAR_FORMAT_READ:

In the case of a read parameter job, the system expects the parameter format at x_PAR_FORMAT_READ. With x_PAR_FORMAT_READ = FALSE, you state that the parameter value at ud_PAR_VALUE_READ is of WORD format (16-bit); with x_PAR_FORMAT_READ = TRUE, you state that the parameter value at ud_PAR_VALUE_READ is of DOUBLEWORD format (32-bit).

The application program makes available the parameter format. (Input x_PAR_FORMAT_READ can be connected to FB BAPS_PAR_READ, output x_PAR_FORMAT).

Inputs x_ERR_IN, i_ERR:

If the application program cannot read or fulfill the CANsync-Master's parameter job, it is possible to state an error number at i_ERR and set x_ERR_IN = TRUE. In this case, the error number is sent to the CANsync master.

The application program makes available the error number, i_ERR, and the error bit x_ERR_IN. (Input x_ERR_IN can be linked to FB BAPS_PAR_WRITE, output x_ERR and/or FB BAPS_PAR_READ, output x_ERR. (Input i_ERR can be linked to FB BAPS_PAR_WRITE, output i_ERR_COMM and/or FB BAPS_PAR_READ, output i_ERR_COMM.)

Input x_OK_IN:

If the application program has fulfilled the CANsync master's parameter job, the system sets input x_OK_IN = TRUE.

In the case of a read parameter job, the system expects the read parameter value at ud_PAR_VALUE_READ and the format of the read parameter at x_PAR_FORMAT_READ. In the case of a write parameter job, the system does not expect any other values.

The application program must make available the OK bit. (Input x_OK can be linked to FB BAPS_PAR_WRITE, output x_OK and/or FB BAPS_PAR_READ, output x_OK.

Input x_EN:

FB CANsync_PAR_SL is activated with x_EN = TRUE . The system only reports parameter jobs and sends answers to the CANsync master when the FB is activated.

If FB CANsync_PAR_SL is deactivated (x_EN = FALSE), the system must wait until the last parameter job has been processed and sent to the CANsync master (x_BUSY = FALSE). Otherwise, it is assumed that communication was cancelled deliberately and you must then reset the FB with x_RESET = TRUE.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB. This is necessary after aborting communication (by means of x_EN = FALSE) or after an error message, for example. After this, you must set x_RESET back to FALSE.

Output x_RESET_JOB:

The CANsync master can cancel a parameter job . In this case, the system sets output x_RESET_JOB to TRUE. Output x_RESET_JOB is set back to FALSE when the CANsync master starts a new parameter job. (Output x_RESET_JOB can be linked to FB BAPS_PAR_WRITE, input x_RESET, FB BAPS_PAR_READ, input x_RESET and/or FB BAPS_SD_CONTROL, input x_RESET.)

Output x_READ:

Output x_READ is set to TRUE if a read parameter job is pending. (Output x_READ can be connected to FB BAPS_PAR_READ, input x_EN).

Output x_READ is set to FALSE if no read parameter job is pending.

Output x_WRITE:

Output x_WRITE is set to TRUE if a write parameter job is pending. (Output x_WRITE can be connected to FB BAPS_PAR_WRITE, input x_EN). Output x_WRITE is set to FALSE if no write parameter job is pending.

Output u_PAR_NR:

The parameter number of the parameter job is output at output u_PAR_NR. (Output u_PAR_NR can be linked to FB BAPS_PAR_WRITE, input u_PAR_NR or FB BAPS_PAR_READ, input u_PAR_NR).

Output x_PAR_FORMAT:

The system outputs at output x_PAR_FORMAT the parameter format of parameter u_PAR_NR in the case of a write parameter job.

x_PAR_FORMAT = FALSE means WORD format (16-bit) and x_PAR_FORMAT = TRUE means DOUBLEWORD format (32-bit). (Output x_PAR_FORMAT can be connected to FB BAPS_PAR_WRITE, input x_PAR_FORMAT).

Output ud_PAR_VALUE:

The system outputs at output x_PAR_VALUE the parameter value of parameter u_PAR_NR in the case of a write parameter job. (Output ud_PAR_VALUE_VALUE can be connected to FB BAPS_PAR_WRITE, input ud_PAR_VALUE).

Output x_ACTIV:

Output x_ACTIV indicates with TRUE that FB CANsync_PAR_SL is waiting during a parameter job for the result of the parameter job (input x_OK_IN or x_ERR_IN). Otherwise, output x_ACTIV is set to FALSE.

Output x_BUSY:

Output x_BUSY indicates with TRUE that FB CANsync_PAR_SL is waiting for the result of the read or write parameter job and that the answer to the CANsync master is ready but the CANsync master has not requested it. Otherwise, output x_BUSY is set to FALSE.

Output x_OK:

The system sets output x_OK to TRUE if the CANsync master has fetched the answer. In this connection, it does not matter whether the answer in question is the error answer or the answer for correct processing of the job. Output x_OK is FALSE if no parameter job has yet been executed, the parameter job has not been executed or the CANsync master has not fetched the answer.

7.3.14CANsync_PAR_WRITE_MA

Description

You can use this function block for CANsync for the CANsync master to send a parameter value to the CANsync slave via the CANsync bus.

NOTE

You can instantiate this FB several times if different CANsync slaves are addressed in each case.

FB CANsync_PAR_WRITE_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR	SINT 0 to 31	Slave number of the CANsync slave to which the write parameter job is addressed
u_PAR_NR	UINT	Parameter number
x_PAR_FORMAT	BOOL	Parameter format
i_SUB_SL	INT 0 to 31	Sub-slave address (reserved)
ud_PAR_VALUE	UDINT	Parameter value to be written
t_TIME	TIME	Monitoring time
x_EN	BOOL	Enable
x_RESET	BOOL	Reset:

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
x_BUSY	BOOL	Communication is active
i_ERR	INT	Error number
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CANsync_PAR_WRITE_MA transfers with the values of inputs u_PAR_NR, x_PAR_FORMAT, i_SUB_SL and ud_PAR_VALUE a write parameter job to the CANsync slave with slave number si_SL_NR. The CANsync slave processes the write parameter job and returns the result of communication.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input si_SL_NR:

At input si_SL_NR, you state the slave number of the CANsync slave on the CANsync bus to which the parameter value is sent.

Input u_PAR_NR:

You state the parameter number for the write parameter job at input u_PAR_NR.

Input x_PAR_FORMAT:

At input x_PAR_FORMAT, you set the format of the parameter value that is to be transferred. x_PAR_FORMAT = FALSE means word format, x_PAR_FORMAT = TRUE means doubleword format.

Input i_SUB_SL:

This input is reserved and is not assigned.

Input ud_PAR_VALUE:

You state the parameter value to be transferred/written at input ud_PAR_VALUE.

Input t_TIME:

At input t_TIME, you state the monitoring time within which the system is to carry out the write parameter job. If the write parameter job is not completed within the monitoring time, the system sets bit 1 of error byte b_ERR to TRUE.

The default setting for t_TIME is 3 s.

Input x_EN:

Communication is started by means of x_EN = TRUE. Input x_EN must not be reset to FALSE until output x_BUSY drops to FALSE after communication is completed. Otherwise, it is assumed that communication was cancelled deliberately and you must reset the FB (x_RESET = TRUE).

Input x_RESET:

FB CANsync_PAR_WRITE_MA is reset by means of x_RESET = TRUE. This is necessary after aborting communication (by means of x_EN = FALSE) or after an error message, for example. After this, you must set x_RESET back to FALSE.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active. Otherwise, x_BUSY = FALSE.

Output x_OK:

Output x_OK is set to TRUE if the write parameter job was executed correctly. Output x_OK is FALSE if the system did not execute a write parameter job or it was not executed correctly.

Outputs x_ERR, b_ERR, i_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR. In this case output x_OK stays FALSE.

If the CANsync slave reports an error number, the system outputs an error word at output i_ERR. The contents of the error word is determined by the application in the CANsync slave.

Error byte b_ERR:

Bit No.	Error
0	Communications error, error number is in i_ERR
1	Timeout
2, 3	Reserved
4	Invalid slave number of the CANsync slaves on the CANsync bus
5 - 7	Reserved

7.3.15 CANsync_PD_CFG_MA

Description

You can use this function block for CANsync to configure the assignment of the CANsync interface module's reference value message frames for a CANsync master.

NOTE

FB CANsync_PD_CFG_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_WRC1	SINT_4_BMARRAY	Reference value numbers for reference value message frame 1
a_HL_WRC1	BOOL_4_BMARRAY	Assignment of highword or lowword for reference value message frame 1
a_WRC2	SINT_4_BMARRAY	Reference value numbers for reference value message frame 2
a_HL_WRC2	BOOL_4_BMARRAY	Assignment of highword or lowword for reference value message frame 2

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module

Using FB CANsync_PD_CFG_MA, you:

- assign eight 32-bit reference values to reference value message frames 1 and 2 (send)

in a CANsync master.

In the CANsync master, the application can write eight 32-bit reference values. The reference value numbers are from 0 to 7.

A reference value is composed of a lowword (bits 0 to 15) and a highword (bit 16 to 31).

In the two reference value message frames, it is possible to send (in each case) 4 * 16-bit data to CANsync slaves. That is four words numbered 0 to 3.

Using FB CANsync_PD_CFG_MA, you specify which data is to be entered in the four words (words 0 to 3) of reference value message frame 1 and which is to be entered in the four words (words 0 to 3) of reference value message frame 2.

You can choose the data for reference value message frame 1 from reference values 0 to 3 and the data for reference value message frame 2 from reference values 4 to 7.

You can assign to each word in a reference value message frame a lowword or a highword of a reference value. Two words are needed in the reference value message frame when transferring a 32-bit reference value.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input a_WRC1:

Assignment of

"Reference value (0 to 3)" -> "word in reference value message frame 1" is carried out in a_WRC1.

a_WRC1[word number] := reference value number

Input a_HL_WRC1:

Assignment of

"Lowword or highword of the selected reference value" -> "word in reference value message frame 1" is carried out in a_HL_WRC1.

a_HL_WRC1[word number] := FALSE (if lowword)

a_HL_WRC1[word number] := TRUE (if highword)

Example:

Word number in reference value message frame 1	Selected reference value	Connection at input a_WRC1	Connection at input a_HL_WRC1
0	Reference value 1 lowword	a_WRC1[0] = 1	a_HL_WRC1[0] = 1
1	Reference value 1 highword	a_WRC1[1] = 1	a_HL_WRC1[1] = 1
2	Reference value 0 word	a_WRC1[2] = 0	a_HL_WRC1[2] = 0 or open
3	Reference value 2 word	a_WRC1[3] = 2	a_HL_WRC1[3] = 0 or open

Input a_WRC2:

Assignment of

"Reference value (4 to 7)" -> "word in reference value message frame 2" is carried out in a_WRC2.

a_WRC2[word number] := reference value number

Input a_HL_WRC2:

Assignment of

"Lowword or highword of the selected reference value" -> "word in reference value message frame 2" is carried out in a_HL_WRC2.

a_HL_WRC2[word number] := FALSE (if lowword)

a_HL_WRC2[word number] := TRUE (if highword)

If you do not want to assign a reference value to a word in reference value message frame 1 or 2, enter -1 as the reference value number at the corresponding entry in a_WRC1 or a_WRC2 respectively.

In this case, the corresponding setting in a_HL_WRC1 or a_HL_WRC2 is meaningless.

a_WRC1[word number] := SINT#-1

a_WRC2[word number] := SINT#-1

Example:
You do not want to assign a reference value to word 1 of reference value message frame 2.

Word number in reference value message frame 2	Selected reference value	Connection at input a_WRC2	Connection at input a_HL_WRC2
1	None	a_WRC2[1] = -1	a_HL_WRC2[1] meaningless

Reference value message frames are assigned to reference values in the CANsync slave. By default, this assignment is carried out in a similar way to the CANsync master. The system does not cross-check the assignment in the CANsync master and the CANsync slave, since there are also reasonable applications for assignments that are different.

7.3.16CANsync_PD_CFG_READ_MA

Description

You can use this function block for CANsync to configure the assignment of the actual value message frames of the CANsync slaves in the CANsync interface module for a CANsync master.

NOTE

FB CANsync_PD_CFG_READ_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR	SINT 0 to 31	Slave number of the CANsync slaves from which actual values are received
a_RDC1	SINT_4_BMARRAY	Actual value numbers for actual value message frame 1
a_HL_RDC1	BOOL_4_BMARRAY	Assignment of highword or lowword for actual value message frame 1
a_RDC2	SINT_4_BMARRAY	Actual value numbers for actual value message frame 2
a_HL_RDC2	BOOL_4_BMARRAY	Assignment of highword or lowword for actual value message frame 2

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module

Using FB CANsync_PD_CFG_READ_MA, you:

- assign actual value message frames 1 and 2 of a CANsync slave (receive) to eight 32-bit actual values

in a CANsync master.

In the CANsync master, the application can read eight 32-bit reference values of each CANsync slave.

The CANsync slaves have numbers 0 to 31.

The actual value numbers are from 0 to 7.

An actual value is composed of a lowword (bits 0 to 15) and a highword (bit 16 to 31).

In the CANsync master, it is possible to receive for each CANsync slave two actual value message frames (each) containing 4 * 16 bit data. That is four words (in each case) numbered 0 to 3.

Using FB CANsync_PD_CFG_READ_MA, you specify for a CANsync slave the actual values to which data from the four words (words 0 to 3) of the CANsync slave's actual value message frame 1 is to be assigned and the actual values to which the data from the four words (words 0 to 3) of its actual value message frame 2 is to be assigned.

You can assign the data from actual value message frame 1 to actual values 0 to 3 and the data for actual value message frame 2 to actual values 4 to 7.

You can assign each word in an actual value message frame to only one lowword or highword of an actual value.

When transferring a 32-bit actual value, two words are needed in the actual value message frame (one word from the actual value message frame is assigned to the lowword of an actual value and another word in this actual value message frame is assigned to the highword of this actual value).

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input si_SL_NR:

At input si_SL_NR, you state the slave number of the CANsync slave on the CANsync bus whose actual value message frames are to be configured.

Input a_RDC1:

Assignment of

"Word in actual value message frame 1" -> "actual value (0 to 3)" is carried out in a_RDC1.

a_RDC1[word number] := actual value number

Input a_HL_RDC1:

Assignment of

"Word in actual value message frame 1" -> "lowword or highword of the selected actual value" is carried out in a_HL_RDC1.

a_HL_RDC1[word number] := FALSE (if lowword)

a_HL_RDC1[word number] := TRUE (if highword)

Example:

Word number in actual value message frame 1	Selected actual value	Connection at input a_RDC1	Connection at input a_HL_RDC1
0	Actual value 1 lowword	a_RDC1[0] = 1	a_HL_RDC1[0] = 0
1	Actual value 1 highword	a_RDC1[1] = 1	a_HL_RDC1[1] = 1
2	Actual value 0 word	a_RDC1[2] = 0	a_HL_RDC1[2] = 0
3	Actual value 2 word	a_RDC1[3] = 2	a_HL_RDC1[3] = 0

Input a_RDC2:

Assignment of

"Word in actual value message frame 2" -> "actual value (4 to 7)" is carried out in a_RDC2.

a_RDC2[word number] := actual value number

Input a_HL_RDC2:

Assignment of

"Word in actual value message frame 2" -> "lowword or highword of the selected actual value" is carried out in a_HL_RDC2.

a_HL_RDC2[word number] := FALSE (if lowword)

a_HL_RDC2[word number] := TRUE (if highword)

If you do not want to assign an actual value to a word in actual value message frame 1 or 2, enter -1 as the reference value number at the corresponding entry in a_RDC1 or a_RDC2 respectively. In this case, the corresponding setting in a_HL_RDC1 or a_HL_RDC1 is meaningless.

a_RDC1[word number] := SINT#-1

a_RDC2[word number] := SINT#-1

Example:
You do not want to assign word 1 of actual value message frame 2 to an actual value.

Word number in actual value message frame 2	Selected actual value	Connection at input a_RDC2	Connection at input a_HL_RDC2
1	None	a_RDC2[1] = -1	a_HL_RDC2[1] meaningless

Actual values are assigned to the words of the actual value message frames in the CANsync slave. By default, this assignment is carried out in a similar way to the CANsync master. The system does not cross-check the assignment in the CANsync master and the CANsync slave, since there are also reasonable applications for assignments that are different.

7.3.17 CANsync_PD_CFG_READ_SL

Description

You can use this function block for CANsync to configure the assignment of the actual value message frames of the CANsync slaves in the CANsync interface module for a CANsync slave.

NOTE

FB CANsync_PD_CFG_READ_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR	SINT 0 to 31	Slave number of the CANsync slaves from which actual values are received
a_RDC1	SINT_4_BMARRAY	Actual value numbers for actual value message frame 1
a_HL_RDC1	BOOL_4_BMARRAY	Assignment of highword or lowword for actual value message frame 1
a_RDC2	SINT_4_BMARRAY	Actual value numbers for actual value message frame 2
a_HL_RDC2	INOUTPUT4_BOOL_BMARRAY	Assignment of highword or lowword for actual value message frame 2

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module

Using FB CANsync_PD_CFG_READ_SL, you:

- assign actual value message frames 1 and 2 of another CANsync slave (receive) to eight 32-bit actual values

in a CANsync slave.

Every CANsync slave can monitor the actual value message frames of the other CANsync slaves and use the data in the actual value message frames.

In the CANsync slave, the application can read eight 32-bit actual values of every other CANsync slave on the CANsync bus.

The CANsync slaves have numbers 0 to 31.

The actual value numbers are from 0 to 7.

An actual value is composed of a lowword (bits 0 to 15) and a highword (bit 16 to 31).

In the CANsync slave, it is possible to receive the two actual value message frames of each CANsync slave, which each contain 4 * 16 bit data. That is four words each numbered 0 to 3.

Using FB CANsync_PD_CFG_READ_SL, you specify for another CANsync slave the actual values to which data from the four words (words 0 to 3) of actual value message frame 1 is to be assigned and the actual values to which the data from the four words (words 0 to 3) of the other CANsync slave's actual value message frame 2 is to be assigned.

You can assign the data from actual value message frame 1 to actual values 0 to 3 and the data for actual value message frame 2 to actual values 4 to 7.

When transferring a 32-bit actual value, two words are needed in the actual value message frame (one word from the actual value message frame is assigned to the lowword of an actual value and another word in this actual value message frame is assigned to the highword of this actual value).

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Input si_SL_NR:

At input si_SL_NR, you state the slave number of the CANsync slave on the CANsync bus whose actual value message frames are to be configured.

Input a_RDC1:

Assignment of

"Word in actual value message frame 1" -> "actual value (0 to 3)" is carried out in a_RDC1.

a_RDC1[word number] := actual value number

Input a_HL_RDC1:

Assignment of

"Word in actual value message frame 1" -> "lowword or highword of the selected actual value" is carried out in a_HL_RDC1.

a_HL_RDC1[word number] := FALSE (if lowword)

a_HL_RDC1[word number] := TRUE (if highword)

Example:

Word number in actual value message frame 1	Selected actual value	Connection at input a_RDC1	Connection at input a_HL_RDC1
0	Actual value 1 lowword	a_RDC1[0] = 1	a_HL_RDC1[0] = 0
1	Actual value 1 highword	a_RDC1[1] = 1	a_HL_RDC1[1] = 1
2	Actual value 0 word	a_RDC1[2] = 0	a_HL_RDC1[2] = 0
3	Actual value 2 word	a_RDC1[3] = 2	a_HL_RDC1[3] = 0

Input a_RDC2:

Assignment of

"Word in actual value message frame 2" -> "actual value (4 to 7)" is carried out in a_RDC2.

a_RDC2[word number] := actual value number

Input a_HL_RDC2:

Assignment of

"Word in actual value message frame 2" -> "lowword or highword of the selected actual value" is carried out in a_HL_RDC2.

a_HL_RDC2[word number] := FALSE (if lowword)

a_HL_RDC2[word number] := TRUE (if highword)

If you do not want to assign an actual value to a word in actual value message frame 1 or 2, enter -1 as the reference value number at the corresponding entry in a_RDC1 or a_RDC2 respectively. In this case, the corresponding setting in a_HL_RDC1 or a_HL_RDC1 is meaningless.

a_RDC1[word number] := SINT#-1

a_RDC2[word number] := SINT#-1

Example:
You do not want to assign word 1 of actual value message frame 2 to an actual value.

.

Word number in actual value message frame 2	Selected actual value	Connection at input a_RDC2	Connection at input a_HL_RDC2
1	None	a_RDC2[1] = -1	a_HL_RDC2[1] meaningless

In the other CANsync slave, actual values are assigned to the words of the actual value message frames. By default, this assignment is carried out in a similar way to the (receiving) CANsync slave. The system does not cross-check the assignment in the (receiving) CANsync slave and the other CANsync slave, since there are also reasonable applications for assignments that are different.

7.3.18CANsync_PD_CFG_SL

Description

You can use this function block for CANsync to configure the assignment of the CANsync interface module's specified and actual value message frames for a CANsync slave.

NOTE

FB CANsync_PD_CFG_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_WRC1	SINT_4_BMARRAY	Reference value numbers for reference value message frame 1
a_HL_WRC1	BOOL_4_BMARRAY	Assignment of highword or lowword for reference value message frame 1
a_WRC2	SINT_4_BMARRAY	Reference value numbers for reference value message frame 2
a_HL_WRC2	BOOL_4_BMARRAY	Assignment of highword or lowword for reference value message frame 2
a_RDC1	SINT_4_BMARRAY	Actual value numbers for actual value message frame 1
a_HL_RDC1	BOOL_4_BMARRAY	Assignment of highword or lowword for actual value message frame 1
a_RDC2	SINT_4_BMARRAY	Actual value numbers for actual value message frame 2
a_HL_RDC2	BOOL_4_BMARRAY	Assignment of highword or lowword for actual value message frame 2

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module

Using FB CANsync_PD_CFG_SL you carry out in a CANsync slave:

- assignment of reference value message frames 1 and 2 (receive) to eight 32-bit reference values

and

- assignment of eight 32-bit actual values to actual value message frames 1 and 2 (send)

Receiving reference values:

In the CANsync slave, the application can write eight 32-bit reference values. The reference value numbers are from 0 to 7.

A reference value is composed of a lowword (bits 0 to 15) and a highword (bit 16 to 31).

In the CANsync slave, it is possible to receive two reference value message frames each containing 4 * 16 bit data. That is four words each numbered 0 to 3.

Using FB CANsync_PD_CFG_MA, you specify the reference values to which the data from the four words (words 0 to 3) of reference value message frame 1 is to be assigned and the reference values to which the data from the four words (words 0 to 3) of reference value message frame 2 is to be assigned.

You can assign the data from reference value message frame 1 to reference values 0 to 3 and the data from reference value message frame 2 can be assigned to reference values 4 to 7.

You can assign each word in a reference value message frame to only one lowword or highword of an actual value.

When transferring a 32-bit reference value, two words are needed in the reference value message frame (one word from the reference value message frame is assigned to the lowword of a reference value and another word in this reference value message frame is assigned to the highword of this reference value).

Sending actual values:

In the CANsync slave, the application can also write eight 32-bit actual values. The actual value numbers are from 0 to 7.

An actual value is composed of a lowword (bits 0 to 15) and a highword (bit 16 to 31).

In the two actual value message frames, it is possible to send 4 * 16-bit data each to the CANsync master. That is four words numbered 0 to 3.

Using FB CANsync_PD_CFG_SL, you specify which data is to be entered in the four words (words 0 to 3) of actual value message frame 1 and which is to be entered in the four words (words 0 to 3) of actual value message frame 2.

You can choose the data for actual value message frame 1 from actual values 0 to 3 and the data for actual value message frame 2 from actual values 4 to 7.

You can assign to each word in an actual value message frame a lowword or a highword of an actual value. Two words are needed in the actual value message frame when transferring a 32-bit actual value.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Input a_WRC1:

Assignment of

"Word in reference value message frame 1" -> "reference value (0 to 3)" is carried out in a_WRC1.

a_WRC1[word number] := reference value number

Input a_HL_WRC1:

Assignment of

"Word in reference value message frame 1" -> "Lowword or highword of the selected reference value" is carried out in a_HL_WRC1.

a_HL_WRC1[word number] := FALSE (if lowword)

a_HL_WRC1[word number] := TRUE (if highword)

Example:

Word number in reference value message frame 1	Selected reference value	Connection at input a_WRC1	Connection at input a_HL_WRC1
0	Reference value 1 lowword	a_WRC1[0] = 1	a_HL_WRC1[0] = 1
1	Reference value 1 highword	a_WRC1[1] = 1	a_HL_WRC1[1] = 1
2	Reference value 0 word	a_WRC1[2] = 0	a_HL_WRC1[2] = 0
3	Reference value 2 word	a_WRC1[3] = 2	a_HL_WRC1[3] = 0

Input a_WRC2:

Assignment of

"Word in reference value message frame 2" -> "reference value (4 to 7)" is carried out in a_WRC2.

a_WRC2[word number] := reference value number

Input a_HL_WRC2:

Assignment of

"Word in reference value message frame 2" -> "Lowword or highword of the selected reference value" is carried out in a_HL_WRC2.

a_HL_WRC2[word number] := FALSE (if lowword)

a_HL_WRC2[word number] := TRUE (if highword)

If you do not want to assign a word from reference value reference value message frame 1 or 2 to a reference value, enter -1 as the reference value number at the corresponding entry in a_WRC1 or a_WRC2 respectively.

In this case, the corresponding setting in a_HL_WRC1 or a_HL_WRC2 is meaningless.

a_WRC1[word number] := SINT#-1

a_WRC2[word number] := SINT#-1

Example:

You do not want to assign word 1 of reference value message frame 2 to a reference value.

Word number in reference value message frame 2	Selected reference value	Connection at input a_WRC2	Connection at input a_HL_WRC2
1	None	a_WRC2[1] = -1	a_HL_WRC2[1] meaningless

Reference values are assigned to the words of the reference value message frames in the CANsync master. By default, this assignment is carried out in a similar way to the CANsync slave. The system does not cross-check the assignment in the CANsync master and the CANsync slave, since there are also reasonable applications for assignments that are different.

Input a_RDC1:

Assignment of

"Actual value (0 to 3)" -> "word in actual value message frame 1" is carried out in a_RDC1.

a_RDC1[word number] := actual value number

Input a_HL_RDC1:

Assignment of

"Lowword or highword of the selected actual value" -> "word in actual value message frame 1" is carried out in a_HL_RDC1.

a_HL_RDC1[word number] := FALSE (if lowword)

a_HL_RDC1[word number] := TRUE (if highword)

Example:

Word number in actual value message frame 1	Selected actual value	Connection at input a_RDC1	Connection at input a_HL_RDC1
0	Actual value 1 lowword	a_RDC1[0] = 1	a_HL_RDC1[0] = 0
1	Actual value 1 highword	a_RDC1[1] = 1	a_HL_RDC1[1] = 1
2	Actual value 0 word	a_RDC1[2] = 0	a_HL_RDC1[2] = 0
3	Actual value 2 word	a_RDC1[3] = 2	a_HL_RDC1[3] = 0

Input a_RDC2:

Assignment of

"Actual value (4 to 7)" -> "Word in actual value message frame 2" is carried out in a_RDC2.

a_RDC2[word number] := actual value number

Input a_HL_RDC2:

Assignment of

"Lowword or highword of the selected actual value" -> "word in actual value message frame 2" is carried out in a_HL_RDC2.

a_HL_RDC2[word number] := FALSE (if lowword)

a_HL_RDC2[word number] := TRUE (if highword)

If you do not want to assign an actual value to a word in actual value message frame 1 or 2, enter -1 as the reference value number at the corresponding entry in a_RDC1 or a_RDC2 respectively. In this case, the corresponding setting in a_HL_RDC1 or a_HL_RDC1 is meaningless.

a_RDC1[word number] := SINT#-1

a_RDC2[word number] := SINT#-1

Example:
You do not want to assign an actual value to word 1 of actual value message frame 2.

Word number in actual value message frame 2	Selected actual value	Connection at input a_RDC2	Connection at input a_HL_RDC2
1	None	a_RDC2[1] = -1	a_HL_RDC2[1] meaningless

In the CANsync master, the words of the actual value message frames are assigned to actual values. By default, this assignment is carried out in a similar way to the CANsync slave. The system does not cross-check the assignment in the CANsync master and the CANsync slave, since there are also reasonable applications for assignments that are different.

7.3.19 CANsync_PD_COMM_MA

Description

You can use this function block for CANsync to carry out process data communication (of reference values and actual values) of the CANsync master interface module.

NOTE

FB CANsync_PD_COMM_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_RD_ARRAY	CANsync_RD_BMARRAY	Output array for the actual values of the CANsync slaves
a_WR_VALUES_SEND	DINT_8_BMARRAY	Reference values that are to be sent
si_WRC1_SEND	SINT 5	Command for reference value message frame 1 to be sent
si_WRC2_SEND	SINT 5	Command for reference value message frame 2 to be sent
si_RD_SL_NR1_RECEIVE	SINT -1, 0 to 31	Slave number of the CANsync slave from which actual value message frame 1 is to be requested
si_RD_SL_NR2_RECEIVE	SINT -1, 0 to 31	Slave number of the CANsync slave from which actual value message frame 2 is to be requested
si_MAX_SL_NR	SINT 0 to 31	Maximum slave number for automatic incrementing[a)]
x_COPY_TO_RD_ARRAY	BOOL	Indication of whether actual values are to be copied into RD_ARRAY

a) This input corresponds to input si_MAX_SL_NR am FB CANsync_COMM_CONTROL_MA as the maximum slave number for automatic polling for send control word jobs, parameter jobs and/or upload/download jobs.

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_RD_ARRAY	CANsync_RD_BMARRAY	Output array for the actual values of the CANsync slaves
si_RD_SL_NR1_RECEIVED	SINT -1, 0 to 31	Display of the slave number of the CANsync slave from which actual value message frame 1 was received

Parameter output	Data type	Description
si_RD_SL_NR2_RECEIVED	SINT -1, 0 to 31	Display of the slave number of the CANsync slave from which actual value message frame 2 was received

Using this FB, the system transfers reference values (a_WR_VALUES_SEND) to the CANsync interface module that are sent to the CANsync slaves by means of the reference value message frames [a)]. In addition, the CANsync slaves request the actual value message frames and [a)] output the actual values (in a_RD_ARRAY).

In each CANsync interval, the system sends reference value message frames 1 and 2 to the CANsync slaves.

In each CANsync interval, CANsync slave number si_RD_SL_NR1_RECEIVE requests actual value message frame 1; and in each CANsync interval, CANsync slave number si_RD_SL_NR2_RECEIVE requests actual value message frame 2.

The CANsync slaves' request for the actual value message frames runs automatically as follows:

In each CANsync interval, the system automatically increments by one the slave number of the CANsync slave (from which actual value message frames 1 and 2 are requested). This incrementation is carried out up to si_MAX_SL_NR. After this, the system starts with the request for the CANsync slave with slave number 0, etc. (si_RD_SL_NR1_RECEIVE and si_RD_SL_NR2_RECEIVE not then assigned).

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input/output a_RD_ARRAY (actual values 0 to 7 of CANsync slaves 0 to 31):

A variable of data type CANsync_RD_BMARRAY is connected at a_RD_ARRAY.

[a)] Assuming you carried out corresponding configuration for reference value message frames 1 and 2 using FB CANsync_PD_CFG_MA for actual value message frames 1 and 2 of each CANsync slave using FB CANsync_PD_CFG_READ_MA.

Data type CANsync_RD_BMARRAY is a two-dimensional field of 32 (CANsync slaves) with 16 actual values each [a)].

This means that data type CANsync_RD_BMARRAY is a field of 32 entries of data type DINT_16_BMARRAY. Data type DINT_16_BMARRAY is a field of 16 entries of data type double integer:

```
DINT_16_BMARRAY          : ARRAY [0..15] OF DINT;
CANsync_RD_BMARRAY       : ARRAY [0..31] OF DINT_16_BMARRAY
```

Example:

```
a_Istwerte               : CANsync_RD_BMARRAY;
```

Where:

`a_Istwerte`	is the variable name with the data type short designation "a" for ARRAY
`CANsync_RD_BMARRAY`	is the data type

The system accesses the individual actual values according to this pattern:

Variable name[slave number of the CANsync slave][number of the actual value]

NOTE

There is no period between the variable name and the square brackets or between the square brackets themselves.

Example: The system writes (in structured text (ST)) variable di_Istwert_21_6 with actual value 6 of the CANsync slave with slave number 21:

```
di_Istwert_21_6 := a_Istwerte[21][6];
```

Inputs a_WR_VALUES_SEND, si_WRC1_SEND and si_WRC2_SEND:

A variable of data type DINT_8_BMARRAY is connected at input a_WR_VALUES_SEND. Data type DINT_8_BMARRAY is a field with 8 entries of data type double integer:

```
DINT_8_BMARRAY           : ARRAY [0..7] OF DINT;
```

Example:

```
a_Sollwerte              : DINT_8_BMARRAY;
```

Where:

`a_Sollwerte`	is the variable name with the data type short designation "a" for ARRAY
`DINT_8_BMARRAY`	is the data type

The system then expects reference values 0 to 7 in field elements `a_Sollwerte[0]` to `a_Sollwerte[7]`, for example.

[a)] Currently, actual values 0 to 3 (actual value message frame 1) and actual values 4 to 7 (actual value message frame 2) are supported.

At input si_WRC1_SEND, the system states when reference values 0 to 3 are valid. Reference value message frame 1 can then be sent A value of 5 indicates that reference values 0 to 3 are valid, whereas any other value indicates that reference values 0 to 3 are not valid.

At input si_WRC2_SEND, the system states when reference values 4 to 7 are valid. Reference value message frame 2 can then be sent A value of 5 indicates that reference values 4 to 7 are valid, whereas any other value indicates that reference values 4 to 7 are not valid.

NOTE

In reference value message frame 1, the system sends reference values 0 to 3; and in reference value message frame 2, it sends reference values 4 to 7. Gaps in reference value numbers are permissible.

Inputs si_RD_SL_NR1_RECEIVE, si_RD_SL_NR2_RECEIVE and si_MAX_SL_NR:

At input si_RD_SL_NR1_RECEIVE, you state the slave number of the CANsync slave from which actual value message frame 1 is requested.

For automatic requesting of the CANsync slaves' actual value message frame 1, si_RD_SL_NR1_RECEIVE is not assigned (or set equal to -128) and the system states the highest slave number at si_MAX_SL_NR.

At input si_RD_SL_NR2_RECEIVE, you state the slave number of the CANsync slave from which actual value message frame 2 is requested.

For automatic requesting of the CANsync slaves' actual value message frame 2, si_RD_SL_NR2_RECEIVE is not assigned (or set equal to -128) and the system states the highest slave number at si_MAX_SL_NR.

In every CANsync interval, the system then increments by one the slave number of the CANsync slave (from which actual value message frame 1 and/or 2 is/are requested automatically) until the number at input si_MAX_SL_NR is reached.

After this, the system starts with polling for the CANsync slave with slave number 0, etc.

It is also possible to mix explicit specification and automatic incrementing for actual value message frames 1 and 2.

The highest slave number of a CANsync slave from which actual value message frames are requested (input si_MAX_SL_NR) is also used for requirements data communication (control word, parameters, upload/download).

If si_MAX_SL_NR = -1, the value remains unchanged on the CANsync interface module.

The default setting is si_MAX_SL_NR = -1, i.e. the value remains unchanged on the CANsync interface module.

NOTE

This input corresponds to input si_MAX_SL_NR on FB CANsync_COMM_CONTROL_MA.
This means that you use input si_MAX_SL_NR on FB CANsync_COMM_CONTROL_MA or input si_MAX_SL_NR on FB CANsync_PD_COMM_MA to state the highest slave number of a CANsync slave.
You must use only one of the two inputs!

Input x_COPY_RD_ARRAY:

At input x_COPY_TO_RD_ARRAY, the system indicates with TRUE that the received actual values are entered in the two-dimensional field at a_RD_ARRAY.

If the system indicates FALSE at x_COPY_TO_RD_ARRAY or x_COPY_TO_RD_ARRAY is not assigned, the received actual values are not entered in the two-dimensional field at a_RD_ARRAY.

Outputs si_RD_SL_NR1_RECEIVED, si_RD_SL_NR2_RECEIVED:

At output si_RD_SL_NR1_RECEIVED, the system displays the slave number of the CANsync slave from which actual value message frame 1 was received in the last CANsync interval. If no actual value message frame 1 was received in a CANsync interval, the system displays -128 at si_RD_SL_NR1_RECEIVED.

At output si_RD_SL_NR2_RECEIVED, the system displays the slave number of the CANsync slave from which actual value message frame 2 was received in the last CANsync interval. If no actual value message frame 2 was received in a CANsync interval, the system displays -128 at si_RD_SL_NR2_RECEIVED.

7.3.20 CANsync_PD_COMM_READ_MA

NOTE

FB CANsync_PD_COMM_READ_MA is present for reasons of compatibility and you should not use it in new projects. The system uses FB CANsync_PD_COMM_MA to output the actual values of all the CANsync slaves to a_RD_BMARRAY.

Description

You can use this function block for CANsync to output in a CANsync master interface module the process data actual values of a CANsync slave.

NOTE

FB CANsync_PD_COMM_READ_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR	SINT 0 to 31	Slave number of the CANsync slaves from which actual values are read

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_RD_VALUES	DINT_8_BMARRAY	Output array for actual values that were received from the CANsync slave
si_RD_SL_NR1_RECEIVED	SINT	Display that actual value message frame 1 was received
si_RD_SL_NR2_RECEIVED	SINT	Display that actual value message frame 2 was received

Using this FB, the system, outputs the actual values of a CANsync slave (a_RD_VALUES). The CANsync slave's request for the actual value message frames must be made via FB CANsync_PD_COMM_MA.

Using FB CANsync_PD_COMM_READ_MA is only reasonable if the actual values are not output at FB CANsync_PD_COMM_MA, i.e. when FB CANsync_PD_COMM_MA, input x_COPY_TO_RD_ARRAY = FALSE.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input si_SL_NR:

At input si_SL_NR, you state the slave number of the CANsync slave on the CANsync bus whose actual values are to be output.

Input/output a_RD_VALUES (actual values 0 to 7 of CANsync slave si_SL_NR):

A variable of data type DINT_8_BMARRAY is connected at a_RD_VALUES. Data type DINT_8_BMARRAY is a field of 8 entries of data type double integer:

```
DINT_8_BMARRAY                  : ARRAY [0..7] OF DINT;
```

Example:

```
a_Istwerte_3 : DINT_8_BMARRAY;
```

Where:

`a_Istwerte_3`	is the variable name with the data type short designation "a" for ARRAY
`DINT_8_BMARRAY`	is the data type

The system accesses the individual actual values according to this pattern:

Variable name[number of the actual value]

Example: The system writes (in structured text (ST)) variable `di_Istwert_3_6` with actual value 6 of the CANsync slave with slave number 3:

```
di_Istwert_3_6 := a_Istwerte_3[6];
```

Currently, actual values 0 to 3 (actual value message frame 1) and actual values 4 to 7 (actual value message frame 2) are supported.

Outputs si_RD_SL_NR1_RECEIVED, si_RD_SL_NR2_RECEIVED:

At output si_RD_SL_NR1_RECEIVED, the system indicates with 1 whether actual value message frame 1 was received. If actual value message frame 1 was not received, si_RD_SL_NR1_RECEIVED = 0. The new actual values 0 to 3 are only output at a_RD_VALUES when actual value message frame 1 was received.

At output si_RD_SL_NR2_RECEIVED, the system indicates with 2 whether actual value message frame 2 was received. If actual value message frame 2 was not received, si_RD_SL_NR2_RECEIVED = 0. The new actual values 4 to 7 are only output at a_RD_VALUES when actual value message frame 2 was received.

7.3.21 CANsync_PD_COMM_READ_SL

NOTE

FB CANsync_PD_COMM_READ_SL is present for reasons of compatibility and you should not use it in new projects. The system uses FB CANsync_PD_COMM_MA to output the monitored actual values to a_RD_BMARRAY of the other CANsync slaves.

Description

You can use this function block for CANsync to output in a CANsync slave interface module the process data actual values of a CANsync slave.

NOTE

FB CANsync_PD_COMM_READ_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
si_SL_NR	SINT 0 to 31	Slave number of the CANsync slaves from which actual values are monitored

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_RD_VALUES	DINT_8_BMARRAY	Output array for actual values that were received from the CANsync slave
si_RD_SL_NR1_RECEIVED	SINT	Display that actual value message frame 1 was received
si_RD_SL_NR2_RECEIVED	SINT	Display that actual value message frame 2 was received

Using this FB, the system, outputs the actual values that were monitored of another CANsync slave (a_RD_VALUES). The CANsync master must use FB CANsync_PD_COMM_MA to request the actual value message frames of the other CANsync slave.

Using FB CANsync_PD_COMM_READ_SL is only reasonable if the actual values are not output at FB CANsync_PD_COMM_SL, i.e. when FB CANsync_PD_COMM_SL, input x_COPY_TO_RD_ARRAY = FALSE.

Input/output _BASE:

A global variable of data type CANsync_SL_CTRL_BMSTRUCT must be connected at _BASE. You must assign this variable via declaration of global variables to the base address of the CAN interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Input si_SL_NR:

At input si_SL_NR, you state the slave number of the CANsync slave on the CANsync bus whose monitored actual values are to be output.

Input/output a_RD_VALUES (actual values 0 to 7 of CANsync slave si_SL_NR):

A variable of data type DINT_8_BMARRAY is connected at a_RD_VALUES. Data type DINT_8_BMARRAY is a field of 8 entries of data type double integer:

```
DINT_8_BMARRAY                  : ARRAY [0..7] OF DINT;
```

Example:

```
a_Istwerte_3 : DINT_8_BMARRAY;
```

Where:

`a_Istwerte_3`	is the variable name with the data type short designation "a" for ARRAY
`DINT_8_BMARRAY`	is the data type

The system accesses the individual actual values according to this pattern:

Variable name[number of the actual value]

Example: The system writes (in structured text (ST)) variable `di_Istwert_22_6` with actual value 6 of the CANsync slave with slave number 22:

```
di_Istwert_22_6 := a_Istwerte_22[6];
```

Currently, actual values 0 to 3 (actual value message frame 1) and actual values 4 to 7 (actual value message frame 2) are supported.

Outputs si_RD_SL_NR1_RECEIVED, si_RD_SL_NR2_RECEIVED:

At output si_RD_SL_NR1_RECEIVED, the system indicates with 1 whether actual value message frame 1 was received. If actual value message frame 1 was not received, si_RD_SL_NR1_RECEIVED = 0. The new actual values 0 to 3 are only output at a_RD_VALUES when actual value message frame 1 was received.

At output si_RD_SL_NR2_RECEIVED, the system indicates with 2 whether actual value message frame 2 was received. If actual value message frame 2 was not received, si_RD_SL_NR2_RECEIVED = 0. The new actual values 4 to 7 are only output at a_RD_VALUES when actual value message frame 2 was received.

7.3.22CANsync_PD_COMM_SL

Description

You can use this function block for CANsync to carry out process data communication (of reference values and actual values) of the CANsync slave interface module.

NOTE

FB CANsync_PD_COMM_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_RD_ARRAY	CANsync_RD_BMARRAY	Output array for the actual values of the CANsync slaves
a_RD_VALUES_SEND	DINT_8_BMARRAY	Actual values that are sent to the CANsync master
si_RDC1_SEND	SINT 5	Command for actual value message frame 1 to be sent
si_RDC2_SEND	SINT 5	Command for actual value message frame 2 to be sent
x_COPY_TO_RD_ARRAY	BOOL	Indication of whether received actual values are to be copied into RD_ARRAY

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_RD_ARRAY	CANsync_RD_BMARRAY	Output array for the actual values of the CANsync slaves
a_WR_VALUES_RECEIVED	DINT_8_BMARRAY	Reference values that were received by the CANsync master
si_WRC1_RECEIVED	SINT	Display that reference value message frame 1 was received
si_WRC2_RECEIVED	SINT	Display that reference value message frame 2 was received
si_RD_SL_NR1_RECEIVED	SINT	Display of the slave number of the CANsync slave from which actual value message frame 1 was received
si_RD_SL_NR2_RECEIVED	SINT	Display of the slave number of the CANsync slave from which actual value message frame 2 was received

Using this FB, the system transfers actual values (a_RD_VALUES_SEND) to the CANsync interface module that are sent to the CANsync master by means of the actual value message frames [a)]. In addition, the other CANsync slaves on the CANsync bus monitor the actual value message frames [a)] and output the actual values (in a_RD_ARRAY).

In each CANsync interval, the system sends reference value message frames 1 and 2 to the CANsync slaves and this FB outputs them to a_WR_VALUES_RECEIVED.

In each CANsync interval, the CANsync master requests actual value message frame 1 from a CANsync slave. The system only sends actual value message frame 1 when the CANsync master requests it from this CANsync slave.

In each CANsync interval, the CANsync master requests actual value message frame 2 from a CANsync slave. The system only sends actual value message frame 2 when the CANsync master requests it from this CANsync slave.

This FB can monitor and evaluate actual value message frames 1 and 2 of the other CANsync slaves when the CANsync master requests these actual value message frames.

NOTE

A CANsync slave can evaluate the actual value message frames of other CANsync slaves but not request them!

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Input/output a_RD_ARRAY (actual values 0 to 7 of [the other] CANsync slaves 0 to 31):

A variable of data type CANsync_RD_BMARRAY is connected at a_RD_ARRAY.

[a)] Assuming you carried out corresponding configuration for reference value message frames 1 and 2 using FB CANsync_PD_CFG_SL for actual value message frames 1 and 2 of each other CANsync slave (except this one) using FB CANsync_PD_CFG_READ_SL .

Data type CANsync_RD_BMARRAY is a two-dimensional field of 32 (CANsync slaves) with 16 actual values each [a].

This means that data type CANsync_RD_BMARRAY is a field of 32 entries of data type DINT_16_BMARRAY. Data type DINT_16_BMARRAY is a field of 16 entries of data type double integer:

```
DINT_16_BMARRAY                 : ARRAY [0..15] OF DINT;
CANsync_RD_BMARAY               : ARRAY [0..31] OF DINT_16_BMARRAY
```

Example:

```
a_Istwerte : CANsync_RD_BMARRAY;
```

Where:

`a_Istwerte`	is the variable name with the data type short designation "a" for ARRAY
`CANsync_RD_BMARRAY`	is the data type

The system accesses the individual actual values according to this pattern:

Variable name[slave number of the CANsync slave][number of the actual value]

NOTE

There is no period between the variable name and the square brackets or between the square brackets themselves.

Example: The system writes (in structured text (ST)) variable `di_Istwert_21_6` with actual value 6 of the CANsync slave with slave number 21:

```
di_Istwert_21_6 := a_Istwerte[21][6];
```

NOTE

The system enters the monitored actual values of the other CANsync slaves in a_RD_ARRAY, but not the actual values of this CANsync slave.

Inputs a_RD_VALUES_SEND, si_RDC1_SEND and si_RDC2_SEND:

A variable of data type DINT_8_BMARRAY is connected at input a_RD_VALUES_SEND. Data type DINT_8_BMARRAY is a field with 8 entries of data type double integer:

```
DINT_8_BMARRAY                  : ARRAY [0..7] OF DINT;
```

[a] Currently, actual values 0 to 3 (actual value message frame 1) and actual values 4 to 7 (actual value message frame 2) are supported.

Example:

```
a_Istwerte_Senden : DINT_8_BMARRAY;
```

Where:

`a_Istwerte_Senden`	is the variable name with the data type short designation "a" for ARRAY
`DINT_8_BMARRAY`	is the data type

The system enters actual values 0 to 7 in entries 0 to 7 (e.g. `a_Istwerte_Senden[0]` to `a_Istwerte_Senden[7]`).

At input si_RDC1_SEND, the system states when actual values 0 to 3 are valid. The system can then send actual value message frame 1, assuming that the CANsync master requests it. A value of 5 indicates that actual values 0 to 3 are valid, whereas any other value indicates that actual values 0 to 3 are not valid.

At input si_RDC2_SEND, the system states when reference values 4 to 7 are valid. The system can then send actual value message frame 2, assuming that the CANsync master requests it. A value of 5 indicates that actual values 4 to 7 are valid, whereas any other value indicates that actual values 4 to 7 are not valid.

NOTE

In actual value message frame 1, the system sends actual values 0 to 3; and in actual value message frame 2, it sends actual values 4 to 7. Gaps in actual value numbers are permissible.

Input x_COPY_RD_ARRAY:

At input x_COPY_TO_RD_ARRAY, the system indicates with TRUE that the monitored actual values are entered in the two-dimensional field at a_RD_ARRAY.

If the system indicates FALSE at x_COPY_TO_RD_ARRAY or x_COPY_TO_RD_ARRAY is not assigned, the monitored actual values are not entered in the two-dimensional field at a_RD_ARRAY.

Outputs a_WR_VALUES_RECEIVED, si_WRC1_RECEIVED, si_WRC2_RECEIVED:

A variable of data type DINT_8_BMARRAY is connected at output a_WR_VALUES_RECEIVED. Data type DINT_8_BMARRAY is a field with 8 entries of data type double integer:

```
DINT_8_BMARRAY                  : ARRAY [0..7] OF DINT;
```

Example:

```
a_Sollwerte : DINT_8_BMARRAY;
```

Where:

`a_Sollwerte`	is the variable name with the data type short designation "a" for ARRAY
`DINT_8_BMARRAY`	is the data type

The system then enters reference values 0 to 7 that the CANsync master received in field elements `a_Sollwerte[0]` to `a_Sollwerte[7]`, for example. The reference values can be of word or double-word format.

At output si_WRC1_RECEIVED, the system indicates with 1 whether reference value message frame 1 was received. If reference value message frame 1 was not received, si_WRC1_RECEIVED = 0. The new reference values 0 to 3 are only output at a_WR_VALUES_RECEIVED when reference value message frame 1 was received.

At output si_WRC2_RECEIVED, the system indicates with 2 whether reference value message frame 2 was received. If reference value message frame 2 was not received, si_WRC2_RECEIVED = 0. The new reference values 4 to 7 are only output at a_WR_VALUES_RECEIVED when reference value message frame 2 was received.

NOTE

In reference value message frame 1, the system receives reference values 0 to 3; and in reference value message frame 2, it receives reference values 4 to 7. Gaps in reference value numbers are permissible.

Outputs si_RD_SL_NR1_RECEIVED, si_RD_SL_NR2_RECEIVED:

At output si_RD_SL_NR1_RECEIVED, the system displays the slave number of the CANsync slave from which actual value message frame 1 was monitored in the last CANsync interval. If no actual value message frame 1 was monitored in a CANsync interval, the system displays -128 at si_RD_SL_NR1_RECEIVED.

At output si_RD_SL_NR2_RECEIVED, the system displays the slave number of the CANsync slave which monitored actual value message frame 2 in the last CANsync interval. If no actual value message frame 2 was monitored in a CANsync interval, the system displays -128 at si_RD_SL_NR2_RECEIVED.

7.3.23 CANsync_SL_TYP_INIT

Description

You can use this function block for CANsync to state the state the slave types for CANsync initialization.

NOTE

This FB is used together with FB CANsync_INIT.

Parameter input	Data type	Description
us_SL_TYP0	USINT	Slave type 0
us_SL_TYP1	USINT	Slave type 1
us_SL_TYP2	USINT	Slave type 2
...	...	...
us_SL_TYP31	USINT	Slave type 31

Parameter output	Data type	Description
a_SL_TYP	BYTE_32_BMARRAY	Initialization data of slave types

Using this FB, the system outputs the slave types of the CANsync slaves on the CANsync bus. The data is entered in a field (output a_SL_TYP). This field is connected at input a_SL_TYP of the FB CANsync_INIT.

Inputs us_SL_TYP0 to us_SL_TYP31:

For the CANsync slave with slave number 0, you state the slave type at input us_SL_TYP0.

us_SL_TYP0 = 0 means that no CANsync slave is present with slave number 0

us_SL_TYP0 = 1 means that a CANsync slave is present with slave number 0

For the CANsync slave with slave number 1, you state the slave type at input us_SL_TYP1.

us_SL_TYP1 = 0 means that no CANsync slave is present with slave number 1

us_SL_TYP1 = 1 means that a CANsync slave is present with slave number 1

etc.

For the CANsync slave with slave number 31, you state the slave type at input us_SL_TYP31.

us_SL_TYP31 = 0 means that no CANsync slave is present with slave number 31

us_SL_TYP31 = 1 means that a CANsync slave is present with slave number 31

Meanings of the slave types:

Slave type	Meaning
0	No CANsync slave with slave number x
1	CANsync slave interface module of an Ωmega Drive-Line II with slave number x, controller with CANsync-Interface with slave number x present
2 - 255	Reserved

Output a_SL_TYP:

A variable of data type BYTE_32_BMARRAY is connected at output a_SL_TYP. Data type BYTE_32_BMARRAY is a field of 32 entries of data type byte:

```
BYTE_32_BMARRAY                    : ARRAY [0..31] OF BYTE;
```

Example:

```
a_Slave_Typen : BYTE_32_BMARRAY;
```

Where:

a_Slave_Typen	is the variable name with the data type short designation "a" for ARRAY
BYTE_32_BMARRAY	is the data type

In the individual entries of the field is located the slave type of the CANsync slave on the CANsync bus. In entry [0], there is the slave type of the CANsync slave with slave number 0; in entry [1] there is the slave type of the CANsync slave with slave number 1, etc.

A 0 in entry [x] means that there is no CANsync slave with slave number x on the CANsync bus.

A value ≠ 0 in entry [x] means that there is one CANsync slave with slave number x on the CANsync bus.

NOTE

This variable is connected at input a_SL_TYP of the FB CANsync_INIT.

7.3.24CANsync_UPDOWNLOAD_MA

Description

You can use this function block for CANsync to carry out an upload or a download in the Block 1 area (see description of CANsync). It is specially designed for use with FB CANsync_UPDOWNLOAD_SL in the CANsync slave.

NOTE

FB CANsync_UPDOWNLOAD_MA uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_DOWNLOAD	CANsync_UPDOWN_BMARRAY	Download values
a_UPLOAD	CANsync_UPDOWN_BMARRAY	Upload values
si_SL_NR	SINT 0 to 31	Slave number of the CANsync slave to which the upload or download job is addressed
x_UPDOWN	BOOL	Upload or download
u_LENGTH	UINT 1 to 2048	Length of the up down block
us_MAX_NR_OS_COMM	USINT	Maximum number of communications attempts
t_TIME	TIME	Monitoring time
x_EN	BOOL	Enable
x_RESET	BOOL	Reset:

Parameter output	Data type	Description
_BASE	CANsync_MA_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_DOWNLOAD	CANsync_UPDOWN_BMARRAY	Download values
a_UPLOAD	CANsync_UPDOWN_BMARRAY	Upload values
x_BUSY	BOOL	Communication is active
w_ERR_SL	WORD	Error word (CANsync slave -> CANsync master)
x_ERR_SL	BOOL	Error bit (group error bit of w_ERRSL)
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit (group error bit of b_ERR)
x_OK	BOOL	OK bit

FB CANsync_UPDOWNLOAD_MA carries out an upload or a download in the block 1 area. FB CANsync_UPDOWNLOAD_MA is specially designed for use with FB CANsync_UPDOWNLOAD_SL in the CANsync slave.

FB CANsync_UPDOWN_MA transfers download data to the CANsync slave with slave number si_SL_NR.

The download job consists of u_LENGTH values from field a_DOWNLOAD.

(FB CANsync_UPDOWN_MA carries out a download job to the CANsync slave in several download message frame blocks. The system sends per download message frame block a maximum of 75 values from field a_DOWNLOAD to the CANsync slave).

The CANsync slave processes the download job and returns the result of communication.

FB CANsync_UPDOWN_MA requests upload data from the CANsync slave with slave number si_SL_NR.

The system requests u_LENGTH values from the CANsync slave.

(FB CANsync_UPDOWN_MA carries out an upload job to the CANsync slave in several upload message frame blocks. The system receives per upload message frame block a maximum of 75 values from the CANsync slave).

The CANsync slave processes the upload job and returns the result of communication. The values are output at a_UPLOAD.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_MA_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 2 (node 2) on Ωmega Drive-Line II

```
_CANsync_CTRL_MA  AT  %MB3.200000  :  CANsync_MA_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_MA`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_MA_CTRL_BMSTRUCT`	is the data type
`%MB3.200000`	is the base address of the CANsync 2 interface module on the Ωmega Drive-Line II

Input/output a_DOWNLOAD:

A variable of data type CANsync_UPDOWN_BMARRAY is connected at a_DOWNLOAD. Data type CANsync_UPDOWN_BMARRAY is a field of 2048 entries of data type double integer:

```
CANsync_UPDOWN_BMARRAY          : ARRAY [0..2047] OF DINT;
```

Example:

```
a_Downloadwerte : CANsync_UPDOWN_BMARRAY;
```

Where:

`a_Downloadwerte`	is the variable name with the data type short designation "a" for ARRAY
`CANsync_UPDOWN_BMARRAY`	is the data type

The system then expects the data for the download in field elements `a_Downloadwerte[0]`to `a_Downloadwerte[2047]`, for example.

Input/output a_UPLOAD:

A variable of data type CANsync_UPDOWN_BMARRAY is connected at a_DOWNLOAD. Data type CANsync_UPDOWN_BMARRAY is a field of 2048 entries of data type double integer:

```
CANsync_UPDOWN_BMARRAY          : ARRAY [0..2047] OF DINT;
```

Example:

```
a_Uploadwerte : CANsync_UPDOWN_BMARRAY;
```

Where:

`a_Uploadwerte`	is the variable name with the data type short designation "a" for ARRAY
`CANsync_UPDOWN_BMARRAY`	is the data type

At uploading, the system then outputs the data in field elements `a_Uploadwerte[0]` to `a_Uploadwerte[2047]`, for example.

Input si_SL_NR:

At input si_SL_NR, you state the slave number of the CANsync slave on the CANsync bus from which the system is to upload data or to which it is to download it.

NOTE

The upload/download to/from this CANsync slave must be enabled via FB CANsync_COMM_CONTROL_MA (see description of FB CANsync_COMM_CONTROL_MA).

Input x_UPDOWN:

At input x_UPDOWN, you set an upload job with x_UPDOWN = FALSE; and with x_UPDOWN = TRUE, you set a download job.

Input u_LENGTH:

At input u_LENGTH, you state the length of the upload/download area to be transferred. The system transfers a maximum of 2048 32-bit values. If you enter a 0 at u_LENGTH, the system sets the bit TRUE in error byte b_ERR.

The system transfers the data block-by-block in message frame blocks. One message frame block is 75 32-bit values in size. This means that with an upload or download job of more than 75 32-bit values, the system needs to send correspondingly more message frame blocks.

Input us_MAX_NR_OF_COMM:

At input us_MAX_NR_OF_COMM, you can specify how often a block of the upload/download area is to be repeated if the following cases apply: a) the monitoring time (of the CANsync master) has expired; and b) the CANsync slave did not report an error (the default setting is us_MAX_NR_OF_COMM = 1). The system does not issue the timeout message (see input t_TIME) until time us_MAX_NR_OF_COMM * t_TIME has expired and up to this instant no answer from the CANsync slave is present.

Input t_TIME:

You state the monitoring time at input t_TIME. If the download job or the upload job is not completely processed within the monitoring time, the system sets bit 2 in the error byte (see also input us_MAX_NR_OF_COMM).

Incomplete processing of an upload or download job can be due to the command channel being busy with higher-priority messages (broadcast commands, send control word jobs, parameter jobs; See “Requirements Data” on page 124.)

Input x_EN:

With x_EN = TRUE, the system enables FB CANsync_UPDOWNLOAD_MA, i.e. upload or download jobs can be sent to the CANsync slave.

If x_EN is set to FALSE before the upload or download job is completed, it is assumed that the job was cancelled deliberately. You must then reset FB CANsync_UPDOWNLOAD_MA with x_RESET = TRUE to start a new upload or download job via the specified length (input u_LENGTH).

Input x_RESET:

You can use x_RESET = TRUE to reset FB CANsync_UPDOWNLOAD_MA. This is necessary after cancelling the upload or download job (using x_EN = FALSE) or following an error message, for example. After this, you must set x_RESET back to FALSE.

Output x_BUSY:

Output x_BUSY indicates with TRUE that FB CANsync_UPDOWNLOAD_MA is processing a job.

Output x_OK:

Output x_OK indicates with TRUE that the upload or download job has been carried out correctly. Output x_OK is FALSE if the system did not execute an upload or download job or it was not executed correctly.

Outputs x_ERR_SL, w_ERR_SL:

If an error occurs in the CANsync slave while the upload or download job is being carried out, the system sets error bit x_ERR_SL to TRUE and outputs error word w_ERR_SL (see below). In this case output x_OK stays FALSE.

Outputs x_ERR, b_ERR:

If an error occurs while the upload or download job is being carried out, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR (see below). In this case output x_OK stays FALSE.

Error byte b_ERR:

Bit No.	Meaning
0	Timeout
1	Length of the upload/download area to be transferred is equal to 0 (input u_LENGTH)
2 to 7	Reserved

Error word w_ERR_SL:

w_ERR_SL	Meaning
16#0000	Reserved
16#0001	CANsync slave acknowledges wrong block number
16#0002	Entered transfer block length > 300 bytes / 75 doublewords
16#0003 to 16#00FF	Reserved
16#0100	CANsync slave expects transfer block with the number that is entered in the counter
16#0101	CANsync slave expects transfer block end
16#0102	CANsync slave does not yet expect transfer block end
16#0103	CANsync slave cancels job
16#0104	Job not possible
16#0105	Base address not allowed
16#0106	Reserved
16#0107	Upload/download area CANsync master > Upload/download area CANsync slave
16#0108	Message frame mode error (mode not allowed at this stage)
16#0109 to 16#FFFF	Reserved

7.3.25CANsync_UPDOWNLOAD_SL

Description

You can use this function block for CANsync to carry out an upload or a download in the Block 1 area (see description of CANsync) depending on the job from the CANsync master. It is specially designed for use with FB CANsync_UPDOWNLOAD_MA in the CANsync master.

NOTE

FB CANsync_UPDOWNLOAD_SL uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_DOWNLOAD	CANsync_UPDOWN_BMARRAY	Download values
a_UPLOAD	CANsync_UPDOWN_BMARRAY	Upload values
t_TIME	TIME	Monitoring time
x_EN	BOOL	Enable
x_RESET	BOOL	Reset:

Parameter output	Data type	Description
_BASE	CANsync_SL_CTRL_BMSTRUCT	Operating data for the CANsync interface module
a_DOWNLOAD	CANsync_UPDOWN_BMARRAY	Download values
a_UPLOAD	CANsync_UPDOWN_BMARRAY	Upload values
x_UPDOWN	BOOL	Display of upload or download
x_ORDER_ACTIV	BOOL	Display of pending job
u_LENGTH	UINT	Length of the transferred block
x_BUSY	BOOL	Display of FB is active
w_ERR	WORD	Error word
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CANsync_UPDOWNLOAD_SL carries out a download job or an upload job of the to the CANsync master. The system outputs download values (a_DOWNLOAD) and sends upload values (a_UPLOAD) to the CANsync master.

The system transfers the data in several message frame blocks. A maximum of 75 32-bit values are transferred per message frame block.

FB CANsync_UPDOWNLOAD_SL is specially designed for use with FB CANsync_UPDOWNLOAD_MA in the CANsync master.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CANsync_SL_CTRL_BMSTRUCT. You must assign this variable via declaration of global variables to the base address of the CANsync interface module.

Example:

CANsync interface module 1 (node 1) on Ωmega Drive-Line II

```
_CANsync_CTRL_SL  AT  %MB3.100000  :  CANsync_SL_CTRL_BMSTRUCT;
```

Where:

`CANsync_CTRL_ SL`	is the variable name with the data type short designation "_" for STRUCT
`CANsync_SL_CTRL_BMSTRUCT`	is the data type
`%MB3.100000`	is the base address of the CANsync 1 interface module on the Ωmega Drive-Line II

Input/output a_DOWNLOAD:

A variable of data type CANsync_UPDOWN_BMARRAY is connected at a_DOWNLOAD. Data type CANsync_UPDOWN_BMARRAY is a field of 2048 entries of data type double integer:

```
CANsync_UPDOWN_BMARRAY          : ARRAY [0..2047] OF DINT;
```

Example:

```
a_Downloadwerte : CANsync_UPDOWN_BMARRAY;
```

Where:

`a_Downloadwerte`	is the variable name with the data type short designation "a" for ARRAY
`CANsync_UPDOWN_BMARRAY`	is the data type

At downloading, the system then outputs the data in field elements `a_Downloadwerte[0]` to `a_Downloadwerte[2047]`, for example.

Input/output a_UPLOAD:

A variable of data type CANsync_UPDOWN_BMARRAY is connected at a_DOWNLOAD. Data type CANsync_UPDOWN_BMARRAY is a field of 2048 entries of data type double integer:

```
CANsync_UPDOWN_BMARRAY          : ARRAY [0..2047] OF DINT;
```

Example:

```
a_Uploadwerte : CANsync_UPDOWN_BMARRAY;
```

Where:

`a_Uploadwerte`	is the variable name with the data type short designation "a" for ARRAY
`CANsync_UPDOWN_BMARRAY`	is the data type

The system then expects the data for the upload in field elements `a_Uploadwerte[0]` to `a_Uploadwerte[2047]`, for example.

Input t_TIME:

You state the monitoring time at input t_TIME. If the download job or the upload job is not completely processed within the monitoring time, the system sets bit 0 in the error word. If input t_TIME is not assigned, this yields a preset monitoring time of 30 s.

Incomplete processing of an upload or download job can be due to the command channel being busy with higher-priority messages (broadcast commands, send control word jobs, parameter jobs; See “Requirements Data” on page 124.

Input x_EN:

With x_EN = TRUE, the system enables FB CANsync_UPDOWNLOAD_SL, i.e. upload or download jobs can be processed.

If x_EN is set to FALSE before the upload or download job is completed, it is assumed that it was cancelled deliberately. You must then reset FB CANsync_UPDOWNLOAD_SL with x_RESET = TRUE to be able to carry out a new upload or download job.

Input x_RESET:

You can use x_RESET = TRUE to reset FB CANsync_UPDOWNLOAD_SL. This is necessary after cancelling the upload or download job (using x_EN = FALSE) or following an error message, for example. After this, you must set x_RESET back to FALSE.

Outputs x_UPDOWN, x_ORDER_ACTIV:

Output x_ORDER_ACTIV indicates with TRUE that a job is present. Output x_UPDOWN displays the type of job. In the case of an upload job, x_UPDOWN = FALSE; with a download job, x_UPDOWN = TRUE.

Output u_LENGTH:

Output u_LENGTH displays the number of 32-bit values of the transferred message frame block. One message frame block is a maximum of 75 32-bit values in size. In the case of an upload or download job that is larger than 75 32-bit values, the system transfers correspondingly more message frame blocks and displays at u_LENGTH the number of 32-bit values of the currently transferred message frame block.

Output x_BUSY:

Output x_BUSY indicates with TRUE that FB CANsync_UPDOWNLOAD_SL is processing an upload or download job.

Output x_OK:

Output x_OK indicates with TRUE that the upload or download job has been carried out correctly. Output x_OK is FALSE if the system did not execute an upload or download job or it was not executed correctly.

Outputs x_ERR, w_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error word b_ERR. In this case output x_OK stays FALSE.

Error word w_ERR:

Bit No.	Meaning
0	Timeout
1 to 15	Reserved

8 INDEX

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Z