

Operating Manual | Bedienungsanleitung

English

Deutsch

QUANTUM^X / SOMAT^{XR} **CANBus**



Hottinger Baldwin Messtechnik GmbH
Im Tiefen See 45
D-64239 Darmstadt
Tel. +49 6151 803-0
Fax +49 6151 803-9100
info@hbm.com
www.hbm.com

Mat.: 7-2002.4461
DVS: A4461-2.0 HBM: public
03.2017

© Hottinger Baldwin Messtechnik GmbH.

Subject to modifications.
All product descriptions are for general information only.
They are not to be understood as a guarantee of quality or
durability.

Änderungen vorbehalten.
Alle Angaben beschreiben unsere Produkte in allgemeiner
Form. Sie stellen keine Beschaffenheits- oder Haltbarkeits-
garantie dar.

Operating Manual | Bedienungsanleitung

English

Deutsch

QUANTUM^X / SOMAT^{XR}

CANBus

Receive / Transmit



1	Safety instructions	5
2	Markings used	12
2.1	The markings used in this document	12
3	QuantumX / SomatXR documentation	14
4	CANbus	15
5	QuantumX / SomatXR and CAN	16
5.1	General information	16
5.2	CAN bus	18
5.2.1	LEDs status display	20
5.2.2	Receiving CAN messages	22
6	Functional description	23
6.1	Global parameters	23
6.1.1	Bit rates	23
6.1.2	CANBus line termination	23
6.1.3	Error handling	24
6.2	Error events	25
6.2.1	Detecting transmit and receive path errors	25
6.2.2	LED and error status behavior	25
6.2.3	Possible error causes in CAN mode	26
6.2.3.1	CANbus warning, CANbus error, CANbus OFF	26
6.2.3.2	CAN “Receiver Overrun”	26
6.2.3.3	CAN “Transmitter Overrun”	27
6.2.3.4	CAN decoder “Timeout”	27
6.2.3.5	CAN decoder “Loss of Signal”	27
6.2.3.6	Module resources	28
6.3	State indication by LED	28
6.3.1	MX840B	28
6.3.2	MX471B	29
6.4	CAN decoder: receiving CAN data	31

6.4.1	Remote frames (RTR)	31
6.4.2	User-defined data type selection	31
6.4.3	Calculation rule data types	32
6.4.4	Floating point scaling	32
6.4.6	Integer scaling	36
6.5	CAN encoder (mappable CAN transmit messages)	39
6.5.1	Motivation	39
6.5.2	Signal source definition	39
6.5.3	Measured value scaling	40
6.5.4	Data types and bit positions of a measured value	41
6.5.5	Transmit data in the event of an error	42
6.5.6	CAN message parameters	43
6.5.6.1	Data length of the CAN message	43
6.5.7	Example of the different signal sources within a single CAN message	44
6.5.8	Transmission type	46
6.5.8.1	Control	47
6.5.8.2	Timer	47
6.5.8.3	SourceChange	47
6.5.8.4	IsoEvent	48
6.5.9	Constraints for MX840B	48
6.6	Data bit numbering systems according to the vector DBC format .	49
6.6.1	Numbering systems used in QuantumX / SomatXR parameterization	49
6.6.1.1	INTEL Standard format	50
6.6.1.2	MOTOROLA Forward MSB format	51
6.6.2	Other numbering systems not used in QuantumX / SomatXR parameterization	52
6.6.2.1	MOTOROLA Forward LSB format	53
6.6.2.2	MOTOROLA Backward format	54
6.6.2.3	INTEL Sequential format	55
6.6.2.4	MOTOROLA Sequential format	56

7	QuantumX / SomatXR and CCP / XCP-on-CAN	57
7.1	Introduction to CCP and XCP	57
7.2	MX471B and CAN / XCP-on-CAN	58
7.3	Initialization per XML	60
7.4	Starting and stopping with the "CANECU" control item	60
7.5	General Workflow	61
8	Glossary	71

1 Safety instructions

Notice

The safety instructions described here also apply to the power pack NTX001 and the active backplane BPX001 and BPX002.

Appropriate use

A module integrated into the CANbus by the relevant connector is to be used exclusively for measurement and test tasks. Use for any purpose other than the above is deemed to be non-designated use.

To ensure safe operation, the module must only be operated as described in the general operating manual and in accordance with the information detailed in this document. It is also essential to comply with the legal and safety requirements for the application concerned during use. The same applies to the use of accessories.

Each time you start up the module, you must first run a project planning and risk analysis that takes into account all the safety aspects of automation technology. This particularly concerns personal and machine protection.

Additional safety precautions must be taken in plants where malfunctions could cause major damage, loss of data or even personal injury. In the event of a fault, these precautions establish safe operating conditions.

This can be done, for example, by mechanical interlocking, error signaling, etc.

Notice

The module must not be connected directly to a power supply system. The supply voltage must be 10 V ... 30 V (DC).

General dangers of failing to follow the safety instructions

Every module is a state of the art device and as such is failsafe. The module may give rise to residual dangers if it is inappropriately installed and operated by untrained personnel. Any person instructed to carry out installation, commissioning, maintenance or repair of the modules must have read and understood the Operating Manuals and in particular the technical safety instructions.

The scope of supply and performance of the modules only covers a small area of measurement technology. In addition, equipment planners, installers and operators should plan, implement and respond to the safety engineering considerations of measurement technology in such a way as to minimize residual dangers. On-site regulations must be complied with at all times. There must be reference to the residual dangers connected with measurement technology. After making settings and carrying out activities that are password-protected, you must make sure that any controls that may be connected remain in safe condition until the switching performance of the module has been tested.

Maintenance and cleaning

The modules are maintenance-free.

- Before cleaning, disconnect all connections.
- Clean the housing with a soft, slightly damp (not wet!) cloth. *Never use* solvent, as this could damage the label or the housing.
- When cleaning, ensure that no liquid gets into the module or connections.

Bus connecting and Outputs

Particular attention must be paid to safety when connecting to the CANbus and when sending CANbus messages. Ensure that integration alone, status or control signals cannot initiate any actions that may pose a danger to persons or the environment.

Product liability

In the following cases, the protection provided for the device may be adversely affected. Liability for device functionality then passes to the operator:

- The device is not used in accordance with the operating manual.
- The device is used outside the field of application described in this section.
- The operator makes unauthorized changes to the device.

Warning signs and danger symbols

Important instructions for your safety are specifically identified. It is essential to follow these instructions in order to prevent accidents and damage to property.

Safety instructions are structured as follows:






Type of danger

Consequences of non-compliance

Averting the danger

- **Warning sign:** draws attention to the danger
- **Signal word:** indicates the severity of the danger (see table below)
- **Type of danger:** identifies the type or source of the danger
- **Consequences:** describes the consequences of non-compliance
- **Defense:** indicates how the danger can be avoided/ bypassed.

Danger classes as per ANSI

Warning sign, signal word	Meaning
	This marking warns of a <i>potentially</i> dangerous situation in which failure to comply with safety requirements may result in death or <i>serious physical injury</i> .
	This marking warns of a <i>potentially</i> dangerous situation in which failure to comply with safety requirements <i>may result in slight or moderate physical injury</i> .
	This marking draws your attention to a situation in which failure to comply with safety requirements <i>may lead to damage to property</i> .

Working safely

The supply connection, as well as the signal and sensor leads, must be installed in such a way that electromagnetic interference does not adversely affect device functionality (HBM recommendation: "Greenline shielding design", downloadable from the Internet at <http://www.hbm.com/Greenline>).

Automation equipment and devices must be covered over in such a way that adequate protection or locking against unintentional actuation is provided (e.g. access checks, password protection, etc.).

When devices are working in a network, these networks must be designed in such a way that malfunctions in individual nodes can be detected and shut down.

Safety precautions must be taken both in terms of hardware and software, so that a line break or other interruptions to signal transmission, e.g. via the bus interfaces, do not cause undefined states or loss of data in the automation device.

Error messages should only be acknowledged once the cause of the error is removed and no further danger exists.

Conversions and modifications

The module must not be modified from the design or safety engineering point of view except with our express agreement. Any modification shall exclude all liability on our part for any resultant damage.

In particular, any repair or soldering work on motherboards (exchanging components) is prohibited. When exchanging complete modules, use only original parts from HBM.

The module is delivered from the factory with a fixed hardware and software configuration. Changes can only be made within the possibilities documented in the manuals.

Qualified personnel



Important

This device is only to be installed and used by qualified personnel strictly in accordance with the specifications and with the safety rules and regulations which follow.

Qualified persons means persons entrusted with the installation, fitting, commissioning and operation of the product who possess the appropriate qualifications for their function. This module is only to be installed and used by qualified personnel, strictly in accordance with the specifications and the safety rules and regulations.

This includes people who meet at least one of the three following requirements:

- Knowledge of the safety concepts of automation technology is a requirement and as project personnel, you must be familiar with these concepts.
- As automation plant operating personnel, you have been instructed how to handle the machinery and are familiar with the operation of the modules and technologies described in this documentation.
- As commissioning engineers or service engineers, you have successfully completed the training to qualify you to repair the automation systems. You are also authorized to activate, ground and label circuits and equipment in accordance with safety engineering standards.

It is also essential to comply with the legal and safety requirements for the application concerned during use. The same applies to the use of accessories.





2 Markings used

catman[®] is a registered trademark of Hottinger Baldwin Messtechnik GmbH.

All trademarks and brands used in this document are trade names and/or trademarks belonging to the respective product or the manufacturer/owner. Hottinger Baldwin Messtechnik GmbH does not lay claim to any other than their own trade names/trademarks.

2.1 The markings used in this document

Important instructions for your safety are specifically identified. It is essential to follow these instructions, in order to prevent damage.

Symbol	Significance
	This marking draws your attention to a situation in which failure to comply with safety requirements <i>can lead</i> to damage to property.
	This marking warns of a <i>potentially</i> dangerous situation in which failure to comply with safety requirements <i>may result in slight or moderate physical injury</i> .
	This marking draws your attention to <i>important</i> information about the product or about handling the product.
	This marking indicates application tips or other information that is useful to you.
Device -> New	Bold text indicates menu items, as well as dialog and window titles in the user interfaces. Arrows between menu items indicate the sequence in which the menus and sub-menus are opened.

Symbol	Significance
<i>Bitrate, 500</i>	Bold text in italics indicates inputs and input fields in the user interfaces.
<i>Emphasize</i> <i>See...</i>	Italics are used to emphasize and highlight text and identify references to sections, diagrams, or external documents and files.

3 QuantumX / SomatXR documentation

The QuantumX / SomatXR family documentation consists of

- the QuantumX / SomatXR operating manual in PDF format
- the data sheets in PDF format
- the operating manuals for the CX22B-W data recorder
- the operating manual for the EtherCAT® ¹⁾ / Ethernet gateway CX27B in PDF format
- the product descriptions for accessories in PDF-format
- a comprehensive online help with index and easy search options which is available after the installation of a software package (e.g. MX Assistant, cat-man®EASY). Information about module and channel configuration can also be found here.

These documents can be found

- on the QuantumX / SomatXR system CD supplied with the device
- After installation of the MX Assistant on the hard drive of your PC, which can be reached through the Windows start menu
- Up-to date versions are always available from our Internet site at www.hbm.com

¹⁾ EtherCAT® is a registered brand and patented technology, licensed by Beckhoff Automation GmbH, Germany

4 CANbus

This manual is intended to support you in connecting your QuantumX or SomatXR system to a CANbus with the following modules:

- MX471B(-R)
- MX840B(-R)

In the following we will refer to these modules as MX471B and MX840B and only state the SomatXR modules in case of significant differences.

When using the CX23-R data recorder there are some restrictions related to the CANbus options (see CX23-R manual).

The catman[®]EASY or MX Assistant software packages have extensive online Help and can be used to configure a CAN port.

This manual shows you:

- How to parameterize a CANbus node

The following documentation is also available:

- General operating manual
- Data sheets MX840B or MX471B
- Online Help in the catman[®]EASY and MX-Assistant software

5 QuantumX / SomatXR and CAN

5.1 General information

The MX471B module provides four independent CAN bus nodes that are all electrically isolated from each other and power supply. Module MX840B provides an electrically isolated CANbus node on channel 1.

Connected devices are not directly addressed during data transmission on a CAN bus. A unique identifier denotes the contents of a message (e.g. rotational speed or engine temperature).

The identifier also signifies the priority of the message.

Message = identifier + signal + additional information

Device connected to the bus = node

Each node on the MX471B can be parameterized either as a receiver or as transmitter (gateway). The online help that comes with the respective software package provides detailed information about parameterization.

Notice

To ensure normal operation, the CAN bus needs to be terminated at both ends, and only there, using appropriate termination resistors.

A 120-ohm termination resistor can be individually connected in the module by software. Termination is also required when short cables with low bit rates are used.

Please refer to the data sheet for the relation between bit rate and maximum bus line length.

The configuration of a node is retained after switching the modules off and on.

For decoding signals at a rate greater than 2000/s, please set up signal inputs 1 to 8 on the MX471B. The signal buffers of these signal inputs have been expanded accordingly.

5.2 CAN bus

Receiving CAN signals:

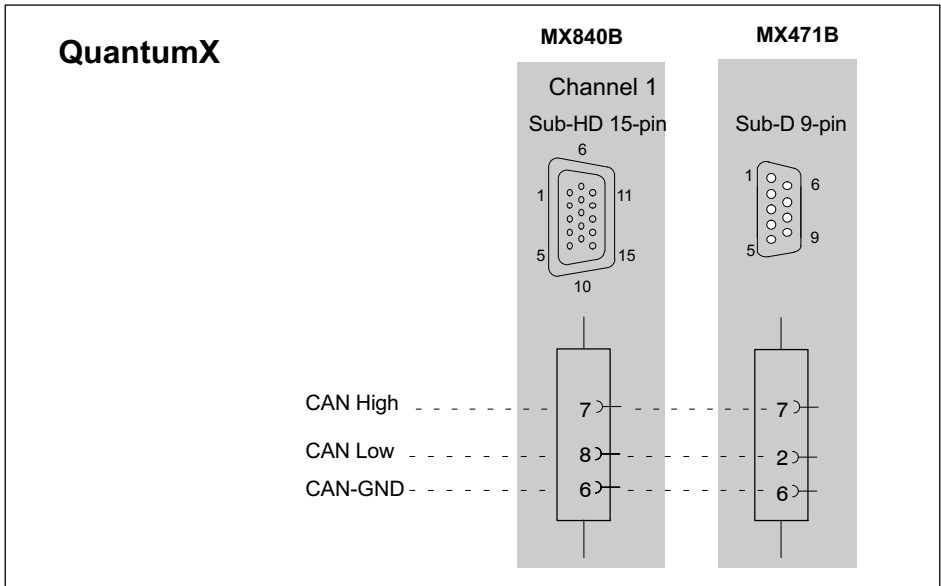
- MX471B, MX840B (channel 1)

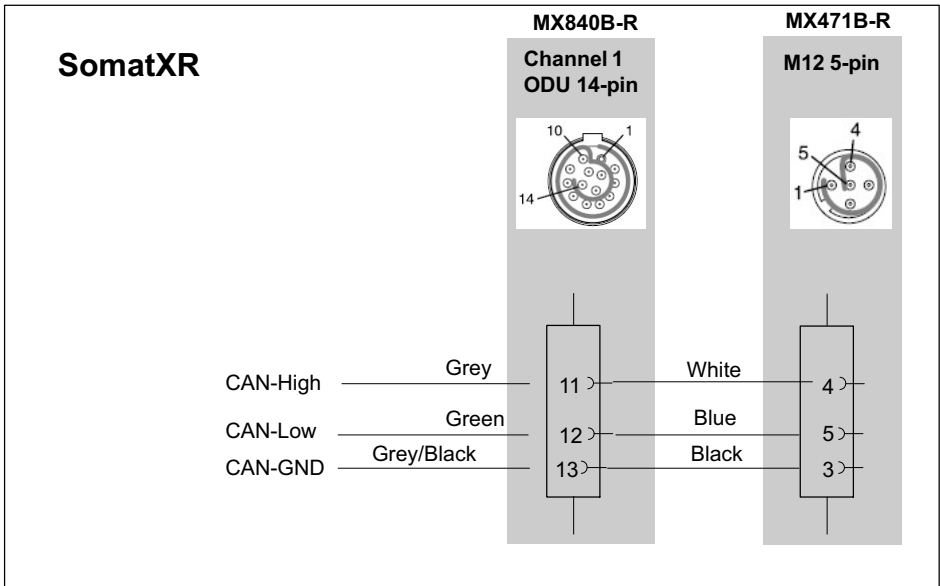
Transmitting CAN signals:

- MX471B
- MX840B (measurement signals within the module only)
- The MX Assistant software can generate a DBC file from the list of all the messages that have been sent

Receiving CCP or XCP-over-CAN signals:

- MX471B only





Notice

Ensure correct termination with termination resistors is made, as shown in Fig. 5.1. The MX840B does not have any termination. The MX471B and the SomatXR MX840B-R have an internal termination that can be activated via software.

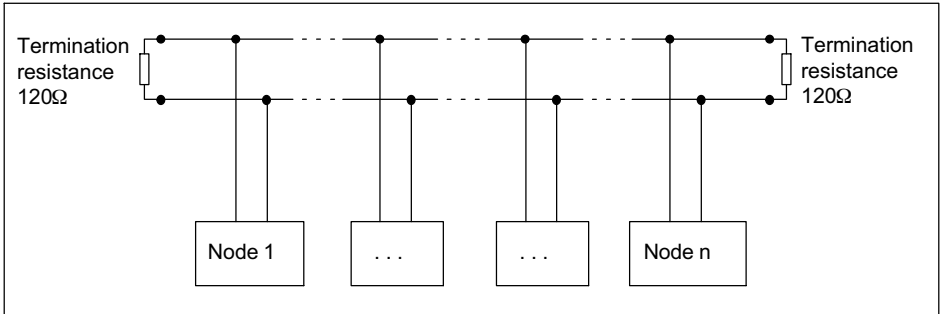


Fig. 5.1 Bus termination resistors

The adapter cable 1-KAB418 is used to connect the D-SUB-15HD device sockets of the QuantumX MX840B to standard D-SUB plugs with a standardized CiA assignment of the MX840B to standard CAN plugs (D-SUB-9).

5.2.1 LEDs status display

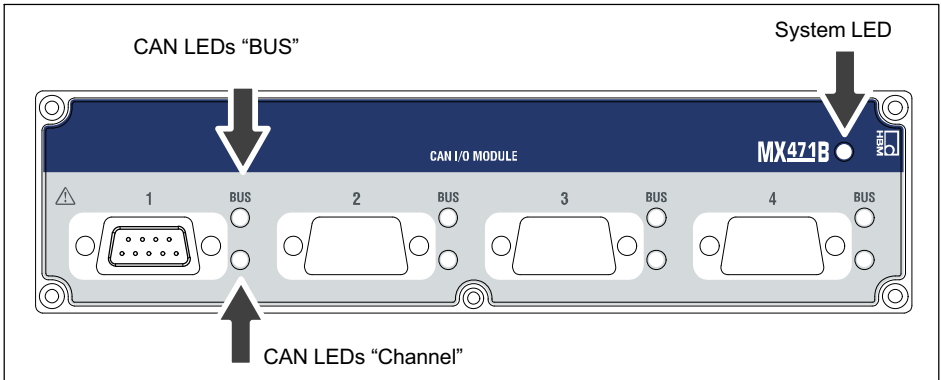


Fig. 5.2 QuantumX MX471B front view

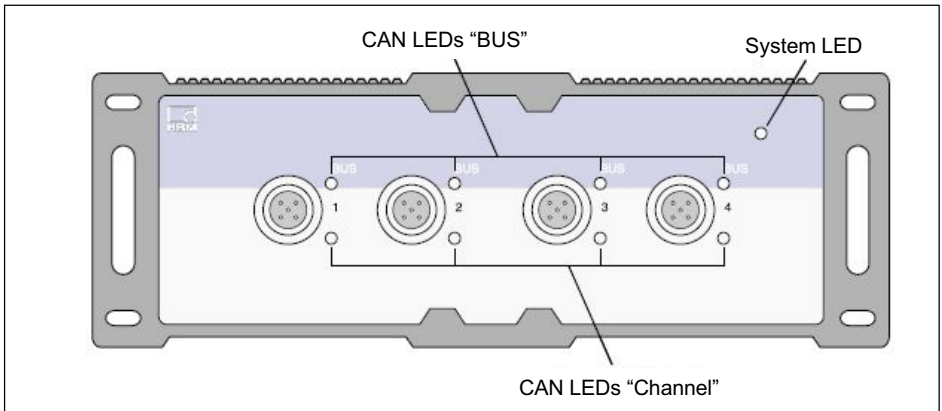


Fig. 5.3 SomatXR MX471B-R front view

For module-related status displays (error messages) see also section 6.3.

System LED

Green	Error-free operation
Yellow	System is not ready, boot procedure running
Flashing yellow	Download active, system is not ready
Red	Error, faulty synchronization

CAN-LEDs (BUS)

Green flickering	Bus is error-free and activity on CAN
Constant green	Bus is error-free and no activity on CAN
Yellow flickering	Intermittent bus errors (warning) and activity on CAN
Constant yellow	Intermittent bus errors (warning) and no activity on CAN
Red on	Bus error, CAN interface in "Bus-OFF" status

CAN LEDs (channel)

Constant green	Channel is ready for operation
Flashing yellow	Firmware download active
Yellow on	Boot process running
Red on	Channel has errors

Ethernet LED (only QuantumX)

Green on	Ethernet link status is OK
Flashing yellow	Ethernet data transmission ongoing

5.2.2 Receiving CAN messages

To be able to receive CAN messages, the node must be able to identify the relevant messages. This can be done directly on the node or, in a reproducible way, by previously generated messages in the sensor database. Individual messages can be linked to the node by dragging from the sensor database and dropping them where required.

CAN databases type *.dbc can also be read into the sensor database. If no CAN database is available, it can also be created. Editors for this purpose are provided by different companies.

Received CAN messages are instantly "time-stamped" in measurement mode. This enables directly acquired measured quantities and CAN messages to be acquired and analyzed in parallel and synchronously in the entire system.

6 Functional description

6.1 Global parameters

6.1.1 Bit rates

Parameterization of the CAN bit rate is extended as of firmware version 4.6. You can select a required bit rate, the sample point and the synchronization jump width (SJW) in a view.

A bit rate (value in bits/s) must be specified for parameterization. In addition to this, “SamplePointRatio” selects a required sample point and “SJW” a synchronization jump width between 1 and 4 time quanta.

Once the parameterization has been forwarded, it is verified in the module and a setting is made that is as close as possible to the required parameters. The parameterization can now be read back, and the values actually set at the time can be read out from the relevant XML tags preceded by “Active”. Users are free to assign the values that are read back to meet their own requirements.

The chosen factory setting is: Bit rate = 1000 kbit/s, SamplePointRatio = 87.5 %, SJW = 4.

6.1.2 CANBus line termination

All the MX471B variants and the MX840B-R can switch CAN bus line termination on or off via parameterization. The MX840, MX840A and MX840B do not support internal line termination.

6.1.3 Error handling

There are applications in which individual error events are critical and must be permanently indicated, even if the cause of the error has long been eliminated. In other applications, it is desirable for the LED and the error status to indicate the current state of CAN bus and module, and for errors to automatically be deleted when their cause is eliminated.

So in the MX Assistant, make a selection in the “CAN bus settings” under “Error mode” to indicate how the module should behave when transient errors occur: If the **“automatic, delayed”** method is selected, the error is automatically deleted from the error status after the set delay time, and the state of the LED changes to normal once the cause of the error has been eliminated. Accordingly, the module shows the current state of the CAN bus after the set delay.

If the **“when reading”** method is selected, the error status remains, and is also indicated by the LED, until the error status has been read out of the module. So provided the cause of the error no longer exists, the error status and the state of the LED only change to normal once the error status has been read out. This is also a reliable way to detect individual error events without having to constantly monitor the error status during measurement. However, this method does not always reflect the current state of the CAN bus.

6.2 Error events

6.2.1 Detecting transmit and receive path errors

Errors in the transmit direction (from the module to the CAN bus) are then only monitored if data has been configured for sending in the module parameterization. Conversely, errors are then only detected in the receive direction (CAN bus to module) if at least one CAN receive decoder is actively configured.

6.2.2 LED and error status behavior

The LED glows yellow whenever a BUS warning occurs, and red in all other error situations.

A timeout and “loss of signal” are indicated by a yellow LED, provided monitoring has been activated. This is because the “MaxRepetitionTime” value selected in the decoder parameterization was greater than “0”.

The error register can be read out via the MX Assistant. The LED follows the status messages in the error register, see Chapter 3.

Error events often only occur sporadically and briefly. If the application also requires reliable detection of one-off events, you can set “error mode” in the parameterization (see Section 1.3) so that errors are only deleted when the error status is read out.

6.2.3 Possible error causes in CAN mode

6.2.3.1 CANbus warning, CANbus error, CANbus OFF

An internal counter in the module's CAN controller counts the bus errors in accordance with the CAN standard. When the first threshold is exceeded, a "Warning" is set, and after that, an "Error". If the cause of the error is promptly eliminated, Warning and Error are automatically deleted again.

If the error still remains, "BUS OFF" will ultimately be set. The bus will become inactive/switch off. Once the cause of the error is eliminated, the state in the module is deleted by an external BUS reset, parameterization or re-starting the module. A BUS reset can be triggered by all MX471B and MX840B variants, but not by MX840 and MX840A.

Errors on the CAN bus are then counted in the CAN controller, for instance, if no correct response is received from the other side during transmission (because no recipient responds by acknowledging), or if the bus is malfunctioning due to a short circuit, incorrect bus termination, mixed operation at different bit rates, or other (usually) electrical properties.

6.2.3.2 CAN "Receiver Overrun"

If the CAN controller in the module cannot read out the data quickly enough, the data present on the CAN bus are lost, and the "Overrun" flag is set. At this moment in time, no-one knows whether the lost data are relevant, that is, whether the CAN identifier for this message is parameterized in the decoder.

Once the data from the CAN bus have been read, they are placed in a temporary buffer, from which messages are then regularly read out and decoded. All the MX840A module variants can decode on average 20,000 measured values per second from the CAN data. If this value is exceeded, the temporary buffer overflows, which results in an error message.

6.2.3.3 CAN “Transmitter Overrun”

If the transmit data cannot be properly sent because the bus is malfunctioning or overloaded, for example, this error is entered into the error status.

6.2.3.4 CAN decoder “Timeout”

The parameter “MaxRepetitionTime” is used to define a timeout. Whenever a CAN message has been decoded, the time to the earlier receipt of this message is determined. If this is greater than this timeout, a measured value marked as invalid is included in the signal current, and an error is indicated.

6.2.3.5 CAN decoder “Loss of Signal”

Unlike a “Timeout”, there is no additional decoded CAN message when there is a “Loss of signal”. The following applies to MX840B: If the signal fails completely within 128 ms, this is detected and a measured value marked as invalid is included in the signal current. This message is deleted again as soon as the next message has been decoded.

6.2.3.6 Module resources

It is true to say that totaled across all CAN connections, the MX471B can decode up to 100,000 signals per second and create CAN messages for sending. If this value is exceeded, CAN message decoding and creation is interrupted until once again, less signals are received or being created. This ensures that the module remains accessible even in an overload situation. A relevant error code is entered in the module error status, and all status LEDs glow constantly red.

6.3 State indication by LED

A distinction is made between the general kind of error message (see Chapter 5.2.1), such as an incorrect bit rate setting, and errors that are dependent on the receive or transmit parameterization. If no transmit channel is parameterized, errors in the sending direction will be ignored. Conversely, there is no error signaling from CAN receiving, if no CAN signals are parameterized in the receive direction.

If the connection LED does not glow green in normal mode, there are usually error messages present, which can be read out via the error status.

6.3.1 MX840B

The behavior of the channel LED in CAN operating mode is described for the MX840B. The flickering of the LED during activity on the CAN bus is not synchronized with the individual messages.

Connection LED	Function
constant green	CAN BUS activated, no errors or failures
flickering green	CAN BUS activated, no errors, activity on the CAN. (since firmware 4.0)
constant orange	CAN controller in "BUS WARNING" state. Receive signal time monitoring indicates a constant or brief failure of the external CAN signal source.
flickering orange	CAN controller defective at times ("BUS WARNING"). Activity on the CAN. (since firmware 4.0)
flashing orange	Firmware is being updated. Only switch off the module once prompted to do so by the update program!
constant red	CAN controller in "BUS ERROR" state, or loss of data in the receiver because of buffer overflow. Transmit data are lost because no CAN recipient is accepting the CAN messages, or because of other CAN BUS malfunctions.
flashing red	CAN controller indicates "BUS OFF", it is not possible to receive or transmit CAN messages. Check the bit rate of all the connected nodes, and for reverse polarity or short circuits. It may take a reset to reactivate the CAN controller. If the other channel LEDs and the system LED are also flashing red, there is a hardware defect.

6.3.2 MX471B

BUS LED	Function
constant green	CAN BUS activated, no errors, no activity on the CAN
flickering green	CAN BUS activated, no errors, activity on the CAN
constant orange	CAN controller defective at times ("BUS WARNING"). No activity on the CAN.

BUS LED	Function
flickering orange	CAN controller defective at times ("BUS WARNING"). Activity on the CAN.
constant red	CAN controller indicates "BUS OFF", it is not possible to receive or transmit CAN messages.

Connection LED	Function
constant green	CAN BUS activated, no errors or failures
constant orange	CAN controller in "BUS WARNING" state. Receive signal time monitoring indicates a constant or brief failure of the external CAN signal source.
flickering orange	CAN controller defective at times ("BUS WARNING"). Activity on the CAN. (since firmware 4.0)
flashing orange	Firmware is being updated. Only switch off the module once prompted to do so by the update program!
constant red	CAN controller in "BUS ERROR" state, or loss of data in the receiver because of buffer overflow. Transmit data are lost because no CAN recipient is accepting the CAN messages, or because of other CAN BUS malfunctions. Computing resources in the module are exceeded: Too many signals have to be decoded and/or too many CAN messages are to be sent.
flashing red	CAN controller indicates "BUS OFF", it is not possible to receive or transmit CAN messages. Check the bit rate of all the connected nodes, and for reverse polarity or short circuits. It may take a reset to reactivate the CAN controller. If the other channel LEDs and the system LED are also flashing red, there is a hardware defect.

6.4 CAN decoder: receiving CAN data

6.4.1 Remote frames (RTR)

A CAN master transmits remote frames to promptly request data. None of the MX module types support remote frames (RTR). These messages are discarded when received.

6.4.2 User-defined data type selection

The CAN decoder constantly distinguishes between the data format in which data are extracted from the CAN message and the data format in which they are available as a signal in the QuantumX / SomatXR system. The CAN decoder behaves differently, depending on the combination of the two data types. This may seem confusing at first, but it allows integer values to constantly be handled as such, so that the processing of the digital I/O data is not corrupted and the floating point calculations are as accurate as possible.

Once the data bytes have been received by the CAN bus, they are handled as the raw data type, as specified in the parameterization (“RawValueFormat”). The signal that will later be used throughout the system is defined by the output format (“SignalFormat”). The integer data types will also be checked to establish whether they are signed (INT32 and INT64) or not (UINT32 and UINT64).

The data type for interpreting the mode-dependent signal is always assumed to be UINT64. The mode length can be selected between 1 and 64. It is not possible to scale the “ModeValue”; nor does there seem to be any point to doing so.

If the selected combination of start bit and length is invalid, the CAN decoder will not change over to the conversion rule and set an error message. The CAN decoder continues to work with the old rule, until a valid one has been transmitted.

6.4.3 Calculation rule data types

Since firmware version 4.3.1, the rules described in the table below apply to the conversion of data types, taking into account the scaling calculation:

$$\text{signal_value} = (\text{raw_value} * \text{factor}) + \text{offset}$$

To make sure that the calculation remains as accurate as possible, and that integer values will not be corrupted even when scaling is used, the scaling calculation is divided into two processes. The data types for “raw”, “factor” and “offset” are calculated differently.

6.4.4 Floating point scaling

Provided at least one of the values of “factor” or „offset“ have been parameterized with a floating point value, the CAN raw signal “raw_value” will first be converted to a floating point value (REAL64). In this case, “factor” and “offset” are always of the floating point data type (REAL64). The intermediate result will then be converted into the data type that has been specified as the “Signal-Format”. The application of floating point numbers means that rounding errors are inherent.

If the data type for “SignalFormat” is 64-bit integer, the decimal places of the floating point value are truncated before the final conversion from REAL64 to UINT64 or

INT64. In all other cases, the floating point value is first rounded to an integer.

The table below only applies to the “floating point scaling” described here:

Signal format	Data format Raw data	Raw data are converted to the calculation format	Data format used for “factor” and “offset”
REAL32 ¹⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
REAL64 ¹⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
INT32 ¹⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
UINT32 ¹⁾	REAL32	REAL64	REAL64
	REAL64		
	INT32		
	INT64		
	UINT32		
	UINT64		

INT64²⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
UINT64²⁾	REAL32	REAL64	REAL64
	REAL64		
	INT32		
	INT64		
	UINT32		
	UINT64		

- 1) When the intermediate value (REAL64) is converted to the 64-bit data format of the signal format, it is rounded up to the next integer.
- 2) When the intermediate value (REAL64) is converted to the 64-bit data format of the signal format, the decimal places are truncated.

REAL32: single precision floating point, size 32-bit, gemäß IEEE 754

REAL64: double precision floating point, size 64-bit, as per IEEE 754

Conversion of the example in the module highlighted in color:

```
C:    int32_t raw;
        double factor, offset;
        int32_t signal = (int32_t)
        (round((double)raw * factor) +
        offset) );
```

```
Pascal: var
        raw: LongInt;
        factor, offset: Double;
        signal: LongInt
    begin
        signal:= LongInt(Round ((Int64(raw)
        * factor) + offset) );
```

6.4.5 CAN raw data reception

All CAN messages received are available as unfiltered signals. At present, the signal can be transmitted by streaming protocol (<DaqAvailable>), however, not within the module group via Firewire (<RtAvailable>).

When raw data reception has been activated, the signal list displays the signal for Connector 2 with a reference: „CanRawReceiver_Connector2.Signal1“

6.4.5.1 CAN raw data in the signal

The signal makes available the following data:

CAN-Id, with Bit 31 determining the Frame format:

Bit 31 = 0: Base Format; Bit 0010: CAN-Identifier.

Bit 31 = 1: Extended Format; Bit 0028: CAN-Identifier

DLC: Number of data bytes; value between 0 and 8

Data: 0 to 8 bytes are transmitted in the signal, depending on “DLC”.

The time stamp on data reception is part of the higher-level streaming data.

Details on signal encoding can be found in the documentation defining the streaming protocol.

6.4.5.2 Parameterization

Parameterization individually switches on and off CAN raw data reception for each CAN connector (<Active>). In addition, the signal, like many other signals in the

QuantumX / SomatXR system, includes a name (<ChannelName>) and an ID specifying whether this name was set by the user or taken from the default settings (<OriginOfName>).

6.4.6 Integer scaling

If the values for both “factor” and “offset” have been parameterized with an integer, particular emphasis is given to the accuracy of the integer. The aim is to avoid rounding errors that are critical if the integer values represent boolean states.

The “raw_value” raw data are also taken from the CAN bus in the raw data format (“RawValueFormat”) first, but are then converted to a calculation format that depends on the data type of “signal_value” (“SignalFormat”), as given in the table below. The “factor” and “offset” are also calculated with the data type given in the table. The intermediate result is ultimately converted to the data format specified as the “SignalFormat”.

The table below only applies to the “integer scaling” described here.

Signal format	Data format Raw data	Raw data are converted to the calculation format	Data format used for "factor" and "offset"
REAL32	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
REAL64	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
INT32	REAL32	REAL32	INT32
	REAL64	INT64	
	UINT32	INT32	
	INT32		
	UINT64	INT64	
	INT64		
UINT32	REAL32	REAL32	INT32
	REAL64	REAL64	
	INT32	INT32	
	INT64	INT64	
	UINT32	UNIT32	UNIT32
	UINT64	UNIT64	

INT64	REAL32	INT64	INT64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
UINT64	REAL32	INT64	INT64
	REAL64	INT32	
	INT32	INT64	
	INT64	UNIT64	UNIT64
	UINT32		
	UINT64		

REAL32: single precision floating point, size 32-bit, gemäß IEEE 754

REAL64: double precision floating point, size 64-bit, as per IEEE 754

Conversion of the example in the module highlighted in color:

```
C:    double raw;
        int32_t factor, offset;
        int32_t signal = (int32_t)
        ( ((int64_t)raw * factor) +
        offset );
```

```
Pascal: var
        raw: Double;
        factor, offset: LongInt;
        signal: LongInt
    begin
        signal := LongInt((Int64(raw)
        * factor) + offset );
```


6.5 CAN encoder (mappable CAN transmit messages)

6.5.1 Motivation

Since firmware version 4.6, the MX471B can send several signals within a CAN message, and measured values can also be converted to an integer format, with a resultant signal length of 1 to 64 bits.

With the generally defined CAN encoder, it is possible to send up to 128 signal sources in up to 128 different CAN messages, with several different signal sources being “mapped” within a single CAN message. This allows up to 64 signal sources to be transmitted in each CAN message. The number of signal sources is determined on the basis of compatibility with the previous <CanTransmit> implementation, and is limited by the internal system resources.

The user parameterizes an individual CAN identifier <Identifier> for each CAN Nachricht, as well as the frame format <ExtendedFrame> (Base/Extended Identifier) and a name <ChannelName>, to appear in the PC software interfaces.

6.5.2 Signal source definition

What is sent and where does it go?

Data from several different sources can be transmitted in each CAN Nachricht. As data 1 bit in length can also be “mapped”, it is conceivable to have up to 64 different sources in a single CAN message. The signal source in the system (e. g. “ModuleReference” = “UUID = 1234”, “SignalReference” = “AnalogIn_Connector1.Signal2”), the

<DataFormat> (e. g. Real32, Integer32) and the kind of coding <BitSequence> (Intel/Motorola) are defined for each source. The sources can be measured values from the QuantumX / SomatXR system transmitted via FireWire, or can also be decoded data received by the CAN bus.

A signal source is typically taken from another MX module. The “ModuleReference” must be set. If the “ModuleReference” points to the module itself, or if its specification is left empty, the data to be received at this or another CAN bus connection within this module, can act as the transmit data source. The module behaves as a “gateway”.

The point at which the data are written and the number of bits in the CAN message are defined for each source, see Section 5.4.

The CAN message in which the data defined in this way will be sent, is specified with the parameters <Identifier> and <ExtendedFrame>. These parameters can also be found in the XML Subtree <CanMessage>, which defines the CAN message and its transmission. This means that the <CanMessage> only has to be parameterized once. The different signal sources refer directly to the relevant CAN message.

6.5.3 Measured value scaling

As with the decoder of the CAN receiver, signals from the source can be scaled with <Factor> and <Offset>, before they are transmitted. This is particularly important if you want to transmit a floating point value as an integer, a practice that is widespread in the CAN world. For example, if you want to send the floating point value with an accuracy of 3 decimal places as an integer, you

imagine a decimal point at the third position of the integer value and multiply the source signal by 1000. (“1.2345“ becomes “1234”).

Scaling is carried out with double precision (REAL64), i.e. the resulting integer value will be rounded in the lower bits.

Selection of the UINT64 data type guarantees one-to-one transfer of all bits, with no scaling being performed. The values in <Factor> and <Offset> will be ignored.

6.5.4 Data types and bit positions of a measured value

Measured values usually occur in the system as REAL32 (float). If the measured values are also to be transmitted as a floating point value, the data format <DataFormat> must be chosen accordingly. The number of bits to be transmitted is defined in <SignalLength> as 32 (float) or 64 (double).

Type conversion is also possible, so that the data in the CAN message will be sent as integers, with any length between 1 and 64 bits. The resolution of the measured value is therefore variable. This achieves a differenzierte Datentyp-Umwandlung.

First the measured value is scaled as a floating point value. If the transmit data format <DataFormat> is selected as an integer value, the conversion to a 32-bit or 64-bit integer takes place now. Then, starting with the MSB of the resultant integer value, the number of bits defined in <SignalLength> are mapped into the CAN message. This means that the most significant bits of a measured value are always transmitted.

Ultimately, the parameter <StartBit> defines at which point the resultant data are included in the CAN message, and <BitSequence> defines which rule is to be applied (INTEL or MOTOROLA, see Chapter 6).

The parameters <DataFormat>, <StartBit>, <SignalLength> and <BitSequence> have a corresponding meaning in the CAN decoder, which is why no other names for the parameters are selected here either.

6.5.5 Transmit data in the event of an error

If the signal source has yet to return a value, or if the signal source transmits an error value (e. g. because of overloading, or an unconnected sensor), a defined value must be entered in the CAN message. The user can select this value with the XML parameters for <ValueOnError>.

“Internal”: The selection of the error value is determined by the module. If <DataFormat> defines a floating point value, “2,0E15” is sent in the event of an error. If <Type> is set to “BitArray”, or <DataFormat> is parameterized to an integer or boolean, “0” is transmitted. This corresponds to the behavior of the module when <ValueOnError> did not yet exist, and is the factory setting.

“Float”: This selection is only permitted if the <DataFormat> defines a floating point value. Then im Fehlerfall der Wert aus <Value> gesendet.

“Integer”: This selection is only permitted if the <DataFormat> defines an integer value. Then the value from <Value> is transmitted in the event of an error.

“Hex”: This selection is valid for all values of <DataFormat>. In the event of an error, the hex string is

sent as a hexadecimal numerical value, taking <BitSequence> into account. The number of bits is determined by the <SignalLength> of the source. The bits of the error value are transmitted, starting with Bit 0 of the error value.

6.5.6 CAN message parameters

How are CAN messages transmitted?

The *contents* of a CAN message, that is, which source data are to be “mapped” in which CAN message at which point, are defined in the subtree <Source>. The header of the CAN message is defined in the <CanMessage> subtree.

The parameters <Identifier> and <ExtendedFrame> in the subtree <Source> establish the <CanMessage> into which this signal source is to be “mapped”.

6.5.6.1 Data length of the CAN message

The data in CAN messages are always multiple bytes. The length of the CAN message corresponds to the number of bytes required to transmit all the signal sources actively “mapped” into this CAN message.

If, for example, only the first 18 data bits of a CAN message are mapped, the data length of the CAN message <ByteCount> is 3 bytes.

It is always possible and acceptable to have “gaps” in the CAN message. Their unspecified bits are transmitted with “0”.

The parameter <ByteCount> provides information about the length of the currently parameterized CAN message. It can only be read out, it cannot be set.

If you want to reduce the load on the CAN bus as a test, you can temporarily set the <Active> tag of a <Source> to “false”. The remaining parameters for this source are retained, but are not active. When the source is later reactivated by setting the <Active> tag to “true”, all the previously defined parameters of this source become active again. When the inactive source at the end of a CAN message is “mapped”, the number of data bytes of the CAN message is shortened accordingly. If, on the other hand, it is situated between additional sources, the data length of the CAN message does not change, the relevant bits of this source are filled in with “0”.

6.5.7 Example of the different signal sources within a single CAN message

Data from different sources are to be transmitted within a single CAN message.

First a measured value is coded into the CAN message as a float value. Then a float value of a mathematical unit is transmitted at bit position 33 as an 18-bit long signed integer taking 2 decimal places into account. Ultimately, starting from bit position 52, three bits are sent with switching states that were previously received in the same module via the CAN bus.

Required parameters, specified by the user. ***Bold italic*** marks the values that are mandatory, in order to meet the above conditions:

The same values for <Identifier> and <Frameformat> apply to all sources.

Source 1:

Input: ModuleReference = "UUID = 001234",
 Signal Reference = "AnalogIn_Connector1.Signal2"
 Type = **MeasVal**
 DataFormat = **REAL32**
 Factor = 1.0
 Offset = 0.0
 BitSequence = INTEL
 StartBit = **0**
 (SignalLength wird automatisch auf 32 gesetzt aufgrund DataFormat = REAL32)

Source 2:

Input: ModuleReference = "UUID = 001256",
 Signal Reference = "Math_Index2.Signal1"
 Type = **MeasVal**
 DataFormat = **Signed Integer32**
 Factor = **100.0** (verschiebt den Messwert der Signalquelle um 2 „gedachte“ Nachkommastellen)
 Offset = 0.0
 StartBit = **33**
 SignalLength = **18**
 BitSequence = INTEL
 (Nachdem der Float-Wert mit 100 multipliziert und danach in einen Signed-Integer32-Wert konvertiert wurde, werden aus diesem Integer-Wert die Bits 31 bis 8 INTEL-codiert gesendet)

Source 3:

Input: ModuleReference = " ", SignalReference =
 "CanReceiver_Connector2_Decoder1.Signal1"
 Type = **MeasVal**
 DataFormat = **Unsigned Integer32**
 Factor = 1.0
 Offset = 0.0
 StartBit = **52**
 SignalLength = **3**

BitSequence = INTEL
 This produces this CAN message

Bit 55 (1 bit)	Bits 54 ... 52 (3 bits)	Bit 51 ... 49 (3 bits)	Bits 48 ... 33 (24 bits)	Bit 32 (1 bit)	Bits 31 ... 0 (32 bits)
0 (not used)	Signal from the CAN decoder (Source 3)	0 0 0 (not used)	Measured value from the mathe- matical unit (Source 2) as an 18-bit inte- ger value	0 (not used)	Measured value from the analog input (Source 1) as a float value (REAL32)

As packets can only ever be transmitted in byte size on the CANbus, this CAN message has a data length of 7 bytes.

With the CAN Receive functionality (CAN decoder <CanInChannel>) in the MX840B, or of another CAN channel of an MX471B, this CAN message can be directly decoded again to four different signals, by taking over <DataFormat>, <StartBit>, <BitSequence> and <SignalLength> from the above parameterization.

6.5.8 Transmission type

When does transmission take place?

For a CAN message, <TransmissionType> defines when the message on the CAN bus is to be sent. There are various modes to choose from. The transmitted value is always the one actually present at this time. If a value for a source is not yet known, the value parameterized in Section 5.5 is transmitted.

6.5.8.1 Control

A CAN message is only sent when a control indicating the connector and the index of the CAN message has been transmitted. A control can be transmitted, for example, by using the MX Assistant menu or button.

6.5.8.2 Timer

The user defines a time interval (in microseconds). The CAN messages are sent automatically at this time interval. If "Interval = 0" is set, no CAN messages are sent.

The transmission of a CAN message can also be triggered by a control (see Section 5.8.1).

The data in the CAN message correspond to the instantaneous value and are not synchronized to the data source.

In the MX471B, we implement timer resolution by counting the isochronous events (1200 Hz). The period of the CAN messages thus corresponds to multiples of 1/1200 seconds.

In the MX840B, triggering occurs in 1 ms steps, with the shortest period being 10 ms per CAN message.

6.5.8.3 SourceChange

If the value of the source for the value most recently sent changes by the absolute value <Delta>, the CAN message is transmitted. <Delta> is defined in the parameters of each source. If several measured values are "mapped" in a CAN message, the entire CAN message is sent when a source changes.

The transmission of a CAN message can also be triggered by a control (see Section 5.8.1).

6.5.8.4 IsoEvent

This corresponds to the behavior implemented in the MX471B up to and including firmware version 4.4: The CAN message is sent when one of the “mapped” signal sources has been received isochronously (via Firewire). <Divisor> can also be used to establish that only every n received isochronous events are sent. This can reduce the workload of the CAN bus, if measured values with low dynamics (e. g. temperature) are to be sent, for example.

In addition, the transmission of a CAN message can also be triggered by a control item (see Section 5.8.1).

6.5.9 Constraints for MX840B

The MX840B is able to send measured values obtained within the module via CAN messages. There are clear constraints here, for reasons of performance. CAN mapping is not possible.

To maintain compatibility with the new XML trees in the MX471B, the MX840B will also only use the new XML trees. This allows both the module types to be parameterized with the same software interface.

The MX840B will also only be able to send a maximum of 10 different CAN messages, and each CAN message will only be able to have 1 source, corresponding to an analog measured value within the module.

So these parameters cannot be freely selected in the MX840B, they are fixed as stated:

Type = MeasVal
 DataFormat = REAL32
 Factor = 1.0
 Offset = 0.0
 StartBit = 0 bzw. 7 (depending on „BitSequence“)
 SignalLength = 32
 BitSequence = INTEL or MOTOROLA
 CanMessage TransmissionType = “Control” or
 “Timer”

A received CAN message cannot be forwarded via CAN. The signal source must also be taken from within the module, that is:

ModuleReference (empty or “UUID=[own UUID]”),
 SignalReference = “AnalogIn_Connector[2...8].Signal1”

The parameters from <ValueOnError> are supported.

For timer-controlled transmission, a measured value can only be sent every 10 ms, or less often. Please note that up to and including firmware version 4.4, the parameter <Period> has been given in milliseconds. To ensure that the future use of this parameter is the same as in the MX471B, <Period> must be given in microseconds in the MX840B as well in future, with a resolution of 1000 µs.

6.6 Data bit numbering systems according to the vector DBC format

6.6.1 Numbering systems used in QuantumX / SomatXR parameterization

The data for the measured values can be interpreted in different numbering systems.

The formats “INTEL Standard”, “MOTOROLA Forward” and “MOTOROLA Backward” use the following “saw-tooth” data bit numbering system:

Byte 0 (from the CAN controller)							
Bit 7	6	5	4	3	2	1	0

Byte 1							
15	14	13	12	11	10	9	8

The formats “INTEL Sequential” and “MOTOROLA Sequential” use the following sequential data bit numbering system. It corresponds to the sequence in which the bits are received by the CAN bus:

Byte 0 (from the CAN controller)							
Bit 0	1	2	3	4	5	6	7

Byte 1							
0	1	2	3	4	5	6	7

6.6.1.1 INTEL Standard format

For signals in “INTEL Standard” format, the stated “Start-bit” is the position of the least significant bit (“lsb”) of the measured value, and the bit significance increases to the left, numbering from the start bit, as shown in the example below.

The following applies for data reception in the CAN decoder: If the data type is signed, this is always assumed in the “msb” and is filled with 1 when right-shifting to the intermediate value. If the “BitSequence” or “ModeBitSequence” element has the value “1”, this

decoding is used both for the measured value and for the mode information.

“INTEL Standard” format example, start bit = 9, length = 12:

Bit no. within the data byte									
7	6	5	4	3	2	1	0		
7	6	5	4	3	2	1	0	0	Data index
15 ←	14 ←	13 ←	12 ←	11 ←	10 ←	9 Start bit/ lsb	8	1	
23	22	21	20 msb ←	19 ←	18 ←	17 ←	16 ←	2	
31	30	29	28	27	26	25	24	3	
39	38	37	36	35	34	33	32	4	
47	46	45	44	43	42	41	40	5	
55	54	53	52	51	50	49	48	6	
63	62	61	60	59	58	57	56	7	

6.6.1.2 MOTOROLA Forward MSB format

For signals in the internal CANdb “MOTOROLA Forward MSB” format, the stated “Startbit” is the position of the most significant bit (“msb”) of the measured value, and the bit significance decreases to the right, as shown in the example below.

The following applies to receiving data in the CAN decoder: If the data type is signed, this is always assumed in the “msb” and when receiving CAN messages, is filled with 1 when right-shifting to the intermediate value. If the “BitSequence” or “ModeBitSequence” element has the value “0”, this decoding is used

both for the measured value and for the mode information.

Internal CANdb “MOTOROLA Forward MSB” format example, start bit = 13, length = 12:

Bit no. within the data byte								Data Index
7	6	5	4	3	2	1	0	
7	6	5	4	3	2	1	0	0
15	14	13 Start bit/ msb	12 ←	11 ←	10 ←	9 ←	8 ←	1
23 ←	22 ←	21 ←	20 ←	19 ←	18 ← lsb	17	16	2
31	30	29	28	27	26	25	24	3
39	38	37	36	35	34	33	32	4
47	46	45	44	43	42	41	40	5
55	54	53	52	51	50	49	48	6
63	62	61	60	59	58	57	56	7

6.6.2 Other numbering systems not used in QuantumX / SomatXR parameterization

The following tables are for your information, so that other numbering systems can be transposed into a format that is used for MX parameterization.

6.6.2.1 MOTOROLA Forward LSB format

Internal CANdb “MOTOROLA Forward LSB” format
example, start bit = 18, length = 12:

Bit no. within the data byte								Data Index
7	6	5	4	3	2	1	0	
7	6	5	4	3	2	1	0	0
15	14	13 msb ←	12 ←	11 ←	10 ←	9 ←	8 ←	1
23 ←	22 ←	21 ←	20 ←	19 ←	18 Start bit/ lsb	17	16	2
31	30	29	28	27	26	25	24	3
39	38	37	36	35	34	33	32	4
47	46	45	44	43	42	41	40	5
55	54	53	52	51	50	49	48	6
63	62	61	60	59	58	57	56	7

6.6.2.2 MOTOROLA Backward format

In this format, numbering starts from the last bit to be sent. This means that the start bit depends on the number of bytes to be sent.

Internal CANdb “MOTOROLA Backward” format
 example, data length = 8, start bit = **42**, length = 12:

Bit no. within the data byte								Data Index
7	6	5	4	3	2	1	0	
63	62	61	60	59	58	57	56	0
55	54	53 msb ←	52 ←	51 ←	50 ←	49 ←	48 ←	1
47 ←	46 ←	45	44 ←	43 ←	42 Start bit/ lsb	41	40	2
39	38	37	36	35	34	33	32	3
31	30	29	28	27	26	25	24	4
23	22	21	20	19	18	17	16	5
15	14	13	12	11	10	9	8	6
7	6	5	4	3	2	1	0	7

The same use information, sent in 3 data bytes:

Internal CANdb “MOTOROLA Backward” format, data length = 3, start bit = 2, length = 12:

Bit no. within the data byte								
7	6	5	4	3	2	1	0	
23	22	21	20	19	18	17	16	0
15	14	13 msb ←	12 ←	11 ←	10 ←	9 ←	8 ←	1
7 ←	6 ←	5	4 ←	3 ←	2 Start bit/ lsb	1	0	2

6.6.2.3 INTEL Sequential format

Internal CANdb “INTEL Sequential” format example, start bit = 14, length = 12:

Bit no. within the data byte								
0	1	2	3	4	5	6	7	
0	1	2	3	4	5	6	7	0
8 ←	9	10 ←	11 ←	12 ←	13 ←	14 Start bit/ lsb	15	1
16	17	18	19 msb ←	20 ←	21 ←	22 ←	23 ←	2
24	25	26	27	28	29	30	31	3
32	33	34	35	36	37	38	39	4
40	41	42	43	44	45	46	47	5
48	49	50	51	52	53	54	55	6
56	57	58	59	60	61	62	63	7

6.6.2.4 MOTOROLA Sequential format

Internal CANdb “MOTOROLA Sequential” format example, start bit = 10, length = 12:

Bit no. within the data byte								
0	1	2	3	4	5	6	7	
0	1	2	3	4	5	6	7	0
8	9	10 msb ←	11 ←	12 ←	13 ←	14 ←	15 ←	1
16 ←	17 ←	18	19 ←	20 ←	21 Start bit/ lsb	22	23	2
24	25	26	27	28	29	30	31	3
32	33	34	35	36	37	38	39	4
40	41	42	43	44	45	46	47	5
48	49	50	51	52	53	54	55	6
56	57	58	59	60	61	62	63	7

Data
Index

7 QuantumX / SomatXR and CCP / XCP-on-CAN

7.1 Introduction to CCP and XCP

Modern Electronic Control Units (ECUs) in vehicles communicate in different protocols with each other or with tuning or analysis tools. The “Universal Measurement and Calibration Protocol” or short XCP is a modern protocol focusing parameterization / calibration or tuning of parameters and data fields of ECUs. As a side task one can use XCP acquiring signal information from ECUs which are normally not part of the standard ECU-2-ECU communication - for example internal signaling or sensor information of the ECU.

The physical connection between the measurement tool and the ECU is still the same bus but the “measurement and calibration protocol” is activated by separate tools like data recorders or tuning tools. XCP has been established as a world-wide by an ASAM working committee (Association for Standardization of Automation and Measuring Systems) as standard.

The “X” stands for the variable and interchangeable transport layer, which might be CANbus (XCP-on-CAN), FlexRay (XCP-on-FlexRay) or Ethernet (XCP-on-Ethernet). XCP succeeds CCP (CAN Calibration Protocol), which has been developed with the conceptual idea of a read and write access to internal ECU data only via CAN.

XCP offers the ability to acquire measured values “event synchronous” to processes in ECUs. This ensures consistency of the data between one another.

7.2 MX471B and CAN / XCP-on-CAN

The MX471B offers 4 individually configurable CAN ports which can be parameterized to work on CCP or XCP-on-CAN and thus to read signal values from ECUs. In this mode the MX471B works as logger. Once communication to the ECU is established, the ECU sends data in a cyclic period.

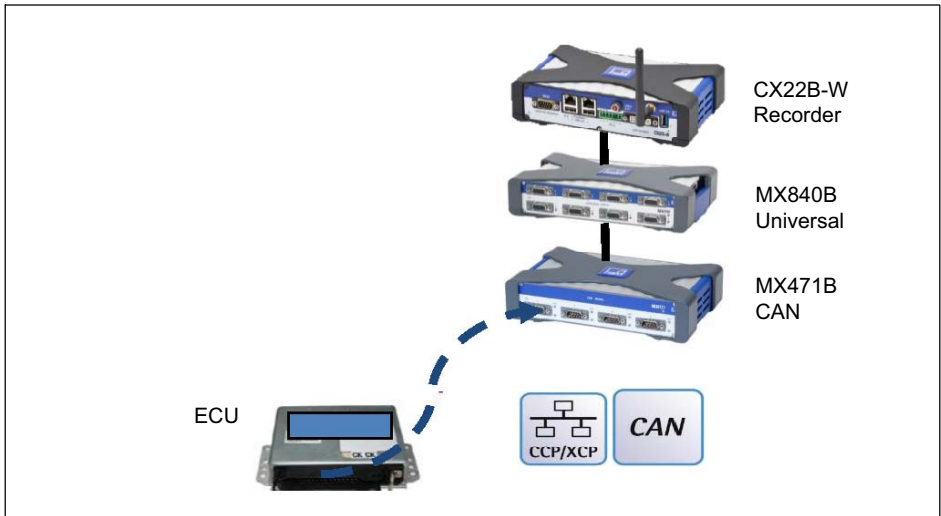


Fig. 7.1 Example configuration MX471B and communication over CCP / XCP-on-CAN

Preconditions

- ECU supports one of the following protocols:
- CCP version 2.1 or higher
- XCP-on-CAN Version 1.1 or higher
- .a2l parameter description file from ECU supplier available

- If ECU is locked by “Seed & Key” mechanism, corresponding *.skb file has to be available
- CANape version 10.0 or higher installed (might also work with previous versions, but not tested)
- MX471B with latest firmware available
- Data Recorder CX22B-W or PC with latest MX Assistant tool available

Hardware Setup

- Connect one of the MX471B ports to the CAN network
- Connect MX471B Ethernet port to a Laptop

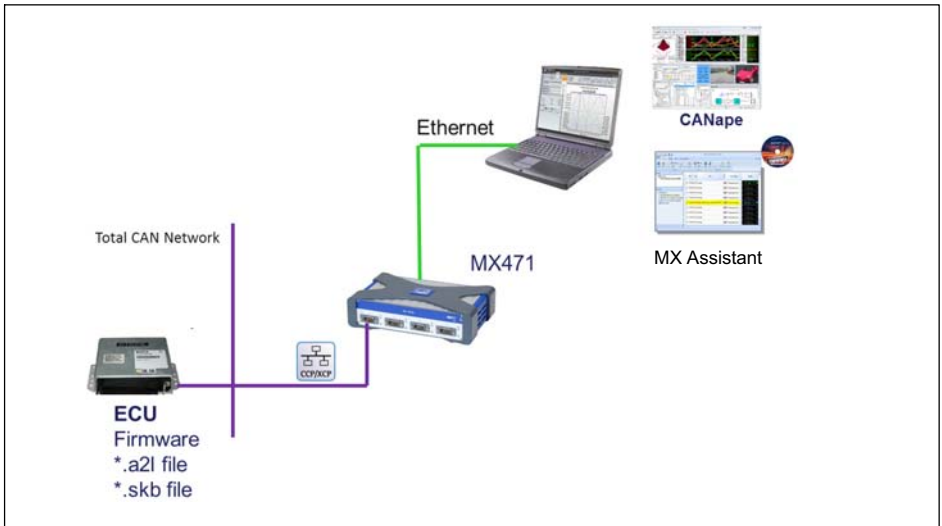


Fig. 7.2 Hardware Setup

7.3 Initialization per XML

The initialization is performed with XML parameters in file "AddConnParam.xml", XPath:

```
...<CanBus><ECU>.
```

The content of a vector DBC file must be written to the <DBC> tag as a string. If a "seed and key binary" file will be used, its content must be written as a string to the <SKB> tag.

Writing these data items initializes the CAN bus for the CCP/XCP protocol.

If the XML tag <Active> was set to "1", processing starts immediately. Otherwise processing does not begin until a control item is written.

After the module is restarted the data items that were previously entered in the XML tags are automatically activated.

The "MXAssistant" PC software provides the "CAN bus" - "Settings..." dialog in the "Channel" tab. The required parameters can be set in the following "CCP / XCP via CAN" dialog.

7.4 Starting and stopping with the "CANECU" control item

This control item can be used with firmware versions 4.8 and later.

If the <Active> tag has been set to "0", processing will not begin until the "CANECU" control item is written.

As a precondition, protocol processing must have been initialized by setting the <DBC> parameter and in some cases also the <SKB> parameter.

Processing can be stopped and started again at any time using this control item. The "CANECU" control item uses these parameters to start and stop:

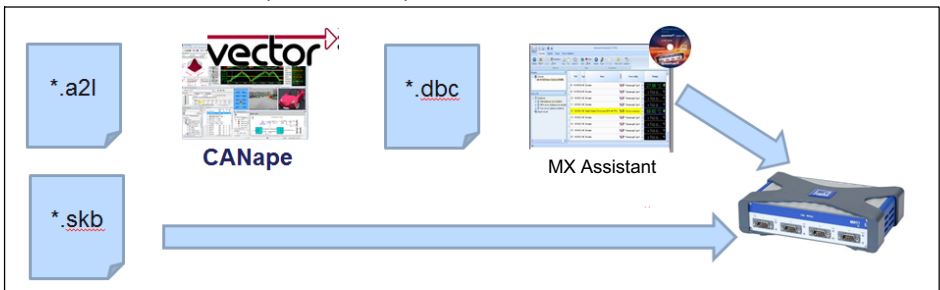
- Connector Number of the CAN bus connections
(„1...4“, „0“: all)
- Index Index of the ECU, currently always "1".
- Mode Choose the functionality "1": Start / Stop
- Value Choose between Start ("1") and Stop ("0")

The "catman" PC software maps control in a corresponding dialog based on the control item.

7.5 General Workflow

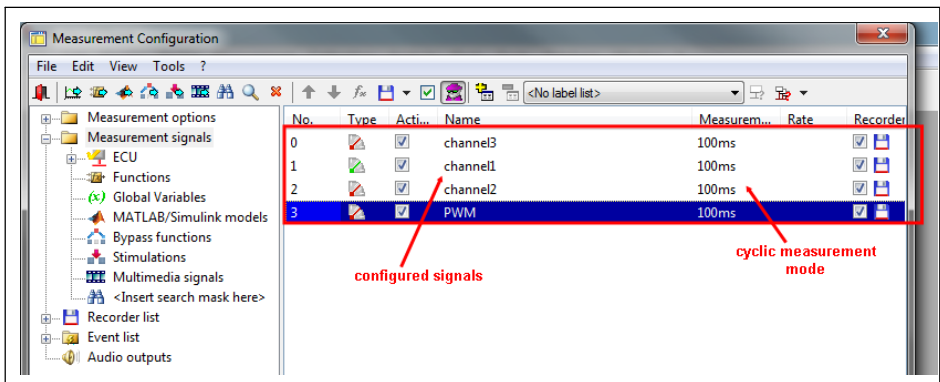
Start CANape from Vector Informatik and read in the *.a2l file. Now create a measurement configuration with all the relevant signals you need acquire by MX471B. Now convert the measurement configuration to a *.dbc file.

Start MX Assistant and read in *.dbc and *.skb files (if applicable) and download it to the device. Immediately CCP or XCP-on-CAN communication will be started between ECU and MX471B. This service can also be activated and deactivated by a script running in catman (under work).



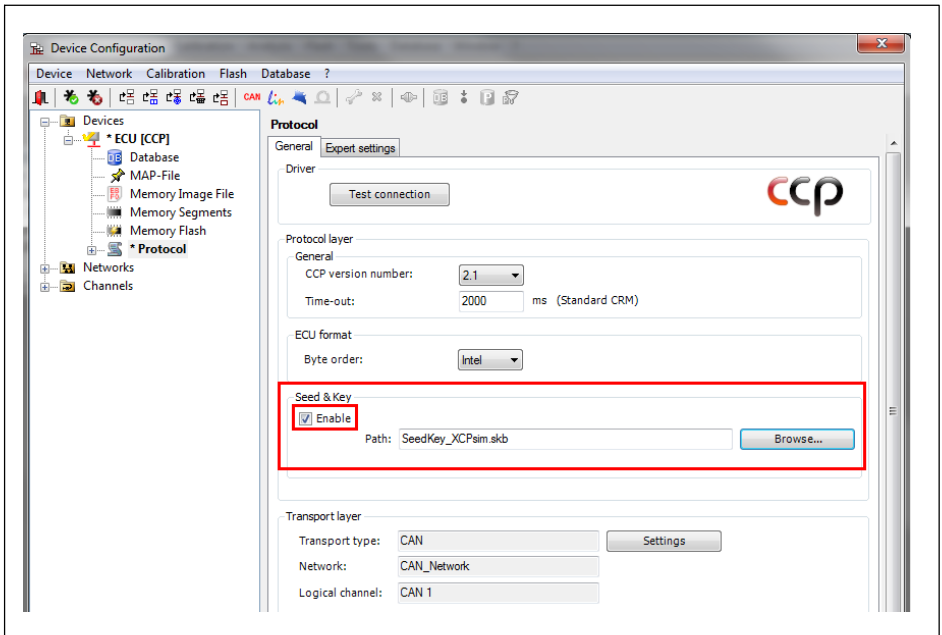
Step 1 - Create Measurement Configuration using CANape

- Create new CANape project
- Create new CCP or XCP device, depending on the protocol that is implemented on the ECU
- Create a measurement configuration with all the signals that you want to record with MX471B. Take care that measurement mode is cyclic. Polling mode is not yet supported.

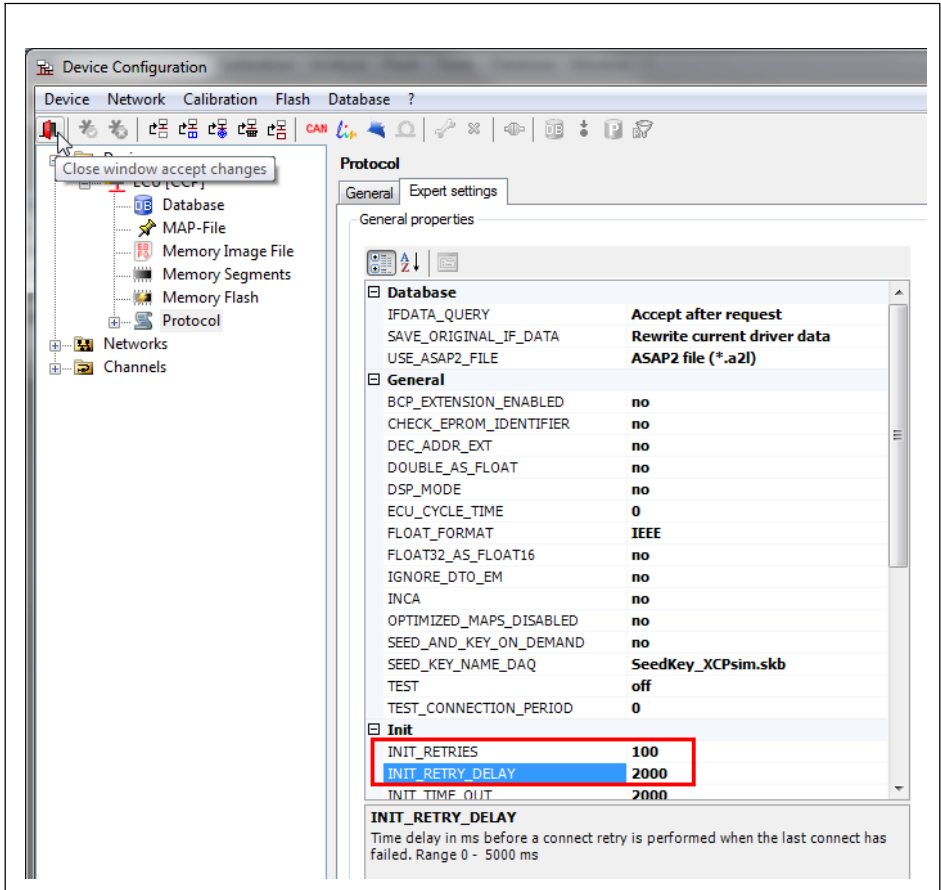


Step 2 – Configure “Seed & Key” option and adjust “Init” settings using CANape

- If the ECU does not support “Seed & Key” option you have to disable it in the Protocol Settings of the Device Configuration
- If the ECU supports “Seed & Key” you have to select the *.skb file.

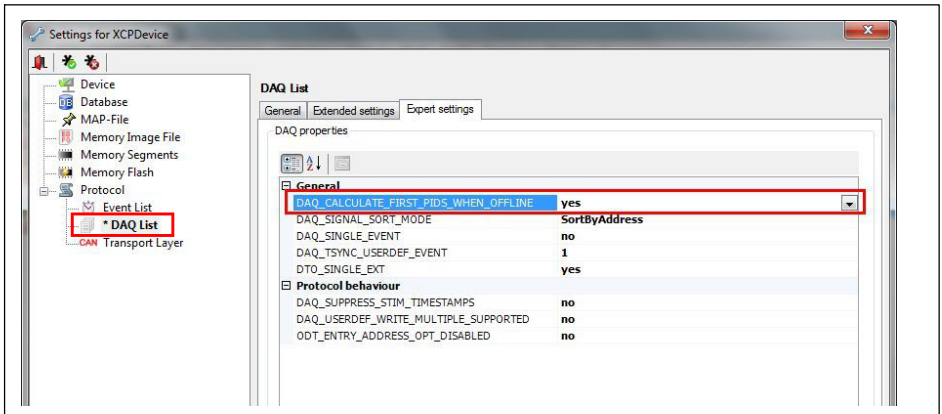


- Switch to Expert settings tab. Modify INIT_RETRIES = 100 and INIT_RETRY_DELAY = 2000 (recommended values).
MX471B will try to start communication with ECU 100 times and every 2000ms. This is important if ECU is powered up after MX471B.

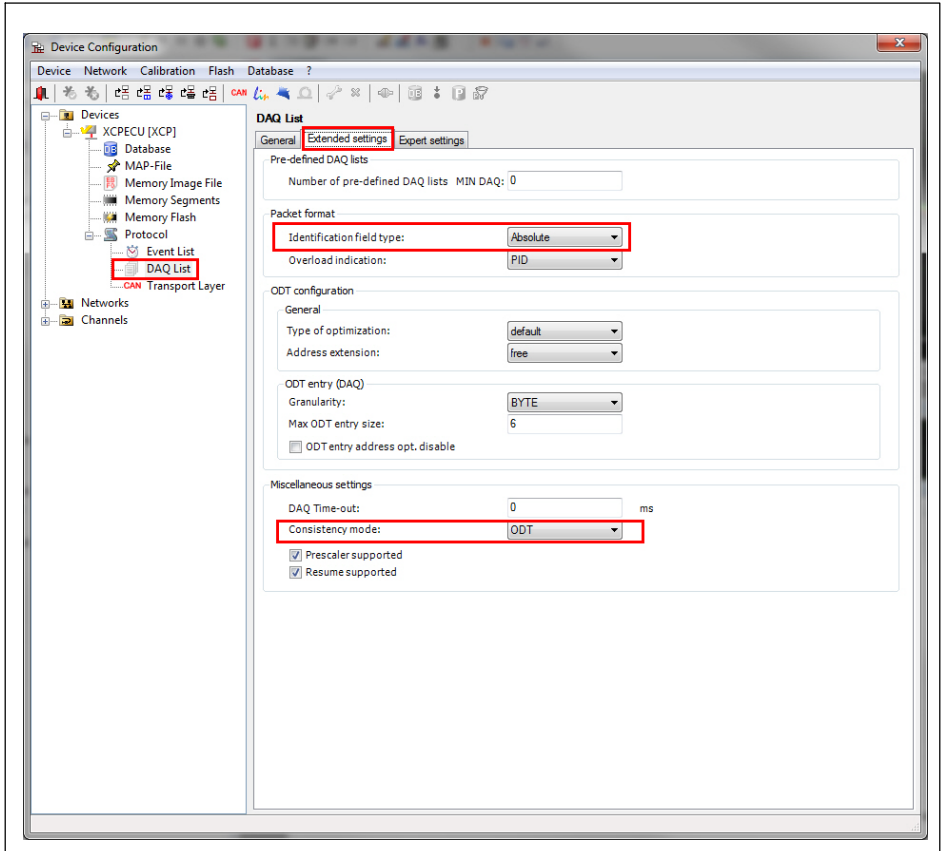


Step 2b – Only for XCP-on-CAN – Adjust Protocol Settings in Device configuration (not necessary for CCP):

- Set parameter "DAQ_CALCULATE_FIRST_PIDS_WHEN_OFFLINE" to "yes"

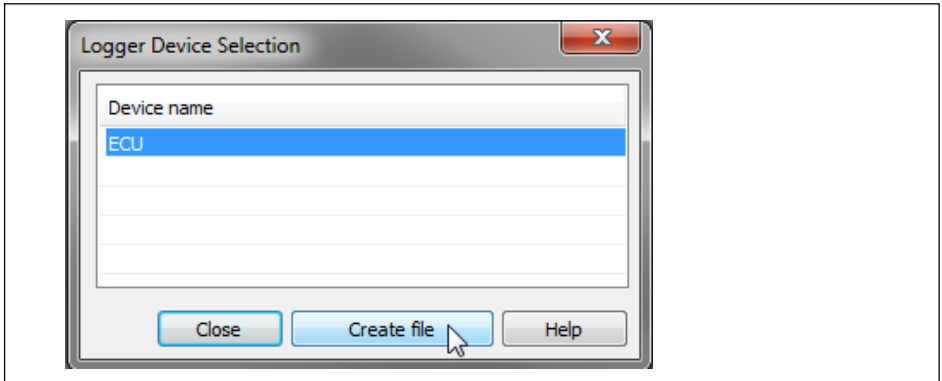


- Set “Consistency mode” to “ODT” and “Identification field type” to “Absolute”



Step 3 - Export Measurement configuration to a *.dbc file using CANape:

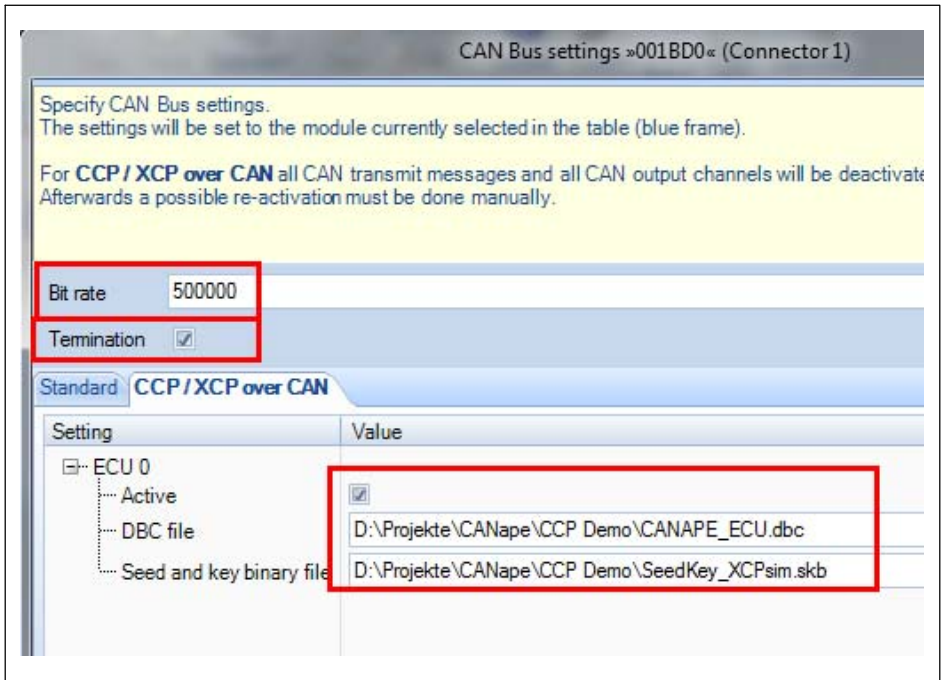
- Select ECU in Logger Configuration (Tools -> Logger configuration) and click on “Create file” button.



- => *.dbc file is created in your CANape project directory

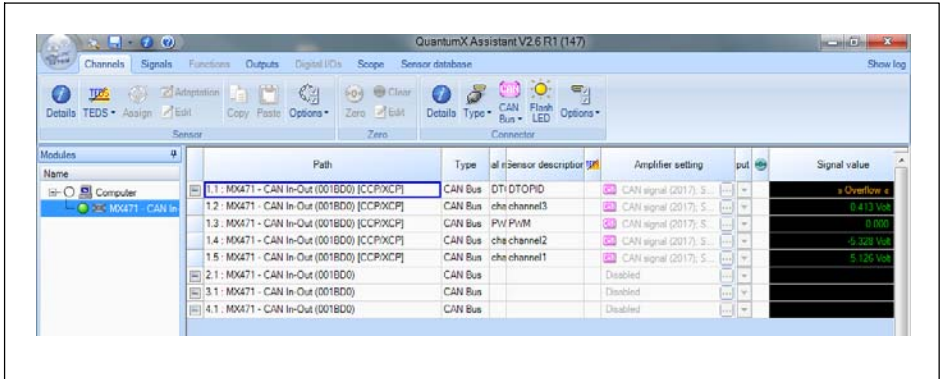
Step 4 – Configure MX471B using *.dbc file and MX Assistant:

- Open MX Assistant and connect to your MX471B
- Select Channel 1 of the MX471B and open CAN Bus settings
- Enter Bit rate of your CAN network
- Activate Termination if necessary
- Select the *.dbc file created in step 3
- If ECU is locked by Seed & Key mechanism you have to select the *skb file.
- If ECU is not locked, leave field blank
- Click OK button



If the check box next to Active is set, processing starts immediately. If the check box is not selected, processing does not begin until a control item is written.

Result: All configured signals are now visible on CAN Bus 1



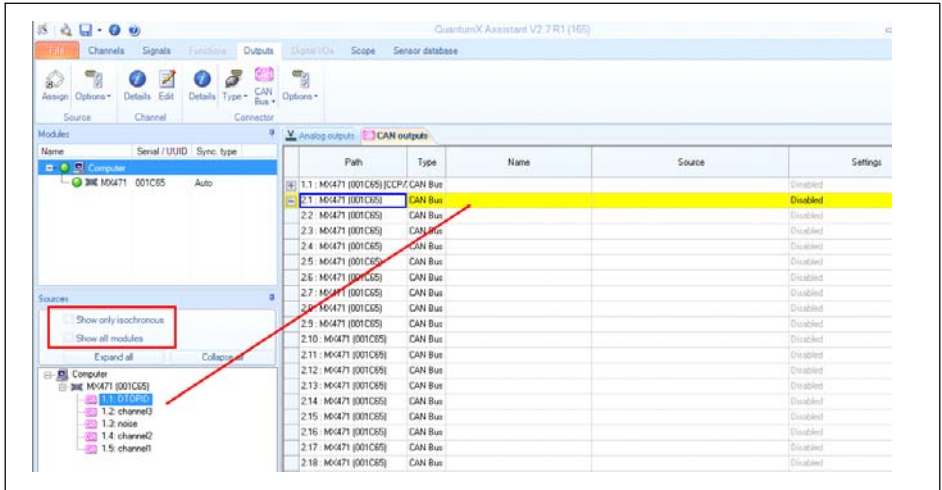
Step 5 – You can now open Catman and start measurement

Configuration of a Gateway Mode

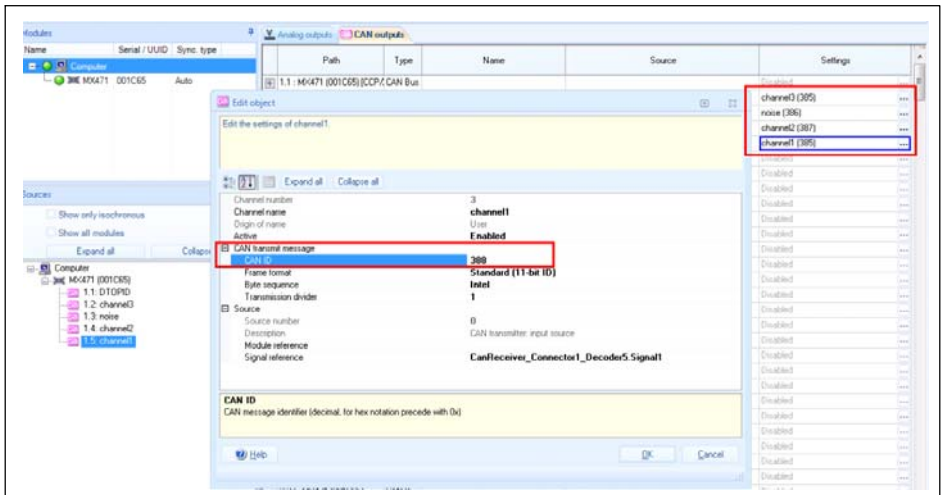
The MX471B can also be used as a gateway transferring XCP-on-CAN or CCP signals to CAN. In this mode XCP or CCP Messages are received on one port of the MX471B and are sent out as standard CAN messages on another port.

Configuration steps in MX Assistant

1. Switch to Outputs tab
2. Uncheck checkbox “Show only isochronous”
3. Drag and drop signals from Sources section to a CAN Output.



- Adjust CAN IDs in that way, that every signal has its own unique ID. In this example 385, 386, 387, 388:
(An automatic assignment of consecutive IDs can be done via CAN bus settings -> Assign message IDs).



8 Glossary

A2L

Extension of a Device Description File of an ECU. Standardized by ASAM.

ASAM

Association for Standardisation of Automation and Measuring Systems

CCP

CAN Calibration Protocol. Standardized by ASAM.

DBC

Data Base CAN: File format for CAN communication

ECU

Electronic Control Unit

Seed & Key

Mechanism to lock the ECU against unauthorized access

XCP

Universal Calibration Protocol. Can be used on different networks: CAN, FlexRay, Ethernet. Standardized by ASAM.



QUANTUM^X / SOMAT^{XR}

CANBus

Empfangen / Senden

1	Sicherheitshinweise	5
2	Verwendete Kennzeichnungen	12
2.1	In dieser Anleitung verwendete Kennzeichnungen	12
3	QuantumX / SomatXR-Dokumentation	14
4	Der CANBus	15
5	QuantumX / SomatXR und CAN	16
5.1	Übersicht	16
5.2	Anschluss CANbus	17
5.2.1	Zustandsanzeige LEDs der Geräte	20
5.2.2	CAN-Nachrichten empfangen	22
6	Funktionsbeschreibung	24
6.1	Globale Parameter	24
6.1.1	Bitraten	24
6.1.2	Terminierung des CANBus	24
6.1.3	Fehlerbehandlung	25
6.2	Fehlerereignisse	26
6.2.1	Erfassen von Fehlern des Sende- und Empfangsweges	26
6.2.2	Verhalten der LED und des Fehlerstatus	26
6.2.3	Mögliche Fehlerursachen im CAN-Betrieb	27
6.2.3.1	CANBus Warning, CANBus Error, CANBus OFF	27
6.2.3.2	CAN „Receiver Overrun“	27
6.2.3.3	CAN „Transmitter Overrun“	28
6.2.3.4	CAN Dekoder „Timeout“	28
6.2.3.5	CAN Dekoder „Loss of Signal“	28
6.2.3.6	Modul-Ressourcen	29
6.3	Zustandsanzeige per LED	29
6.3.1	MX840B	29
6.3.2	MX471B	30
6.4	CAN-Dekoder: Empfang von CAN-Daten	32

6.4.1	Remote-Frames (RTR)	32
6.4.2	Benutzerdefinierte Auswahl der Datentypen	32
6.4.3	Datentypen der Rechenvorschrift	33
6.4.4	Fließkomma-Skalierung	33
6.4.5	CAN-Rohdaten-Empfang	37
6.4.5.1	CAN-Rohdaten im Signal	37
6.4.5.2	Parametrierung	37
6.4.6	Ganzzahl-Skalierung	38
6.5	CAN-Encoder (Mappable CAN-Sende-Nachrichten)	41
6.5.1	Motivation	41
6.5.2	Definition der Signalquellen	41
6.5.3	Skalierung des Messwertes	42
6.5.4	Datentypen und Bit-Positionen eines Messwertes	43
6.5.5	Sendedaten im Fehlerfall	44
6.5.6	Parameter der CAN-Nachricht	45
6.5.6.1	Datenlänge der CAN-Nachricht	45
6.5.7	Beispiel für verschiedene Signalquellen innerhalb einer einzigen CAN-Nachricht	46
6.5.8	Transmission-Type	49
6.5.8.1	Control	49
6.5.8.2	Timer	49
6.5.8.3	SourceChange	50
6.5.8.4	IsoEvent	50
6.5.9	Einschränkungen für MX840B	51
6.6	Zählweisen der Datenbit gemäß Vector-DBC-Format	52
6.6.1	Zählweisen, in QuantumX / SomatXR-Parametrierung genutzt ...	52
6.6.1.1	Format INTEL-Standard	53
6.6.1.2	Format MOTOROLA Forward MSB	54
6.6.2	Weitere Zählweisen, nicht in QuantumX / SomatXR-Parametrierung genutzt	55
6.6.2.1	Format MOTOROLA Forward LSB	56
6.6.2.2	Format MOTOROLA Backward	56
6.6.2.3	Format INTEL Sequential	58
6.6.2.4	Format MOTOROLA Sequential	59

7	QuantumX / SomatXR und CCP / XCP-on-CAN	60
7.1	Einführung in CCP und XCP	60
7.2	MX471B und CAN / XCP-on-CAN	61
7.3	Initialisierung per XML	63
7.4	Start und Stopp per Control „CANECU“	64
7.5	Allgemeiner Arbeitsablauf	65
8	Glossar	76

1 Sicherheitshinweise

Hinweis

Die hier aufgeführten Sicherheitshinweise gelten auch für das Netzteil NTX001 und die Modulträger BPX001 und BPX002.

Bestimmungsgemäße Verwendung

Ein Modul welches über den jeweiligen Anschluss an den CANbus integriert wird, ist ausschließlich für Mess- und Testaufgaben zu verwenden. Jeder darüber hinausgehende Gebrauch gilt als nicht bestimmungsgemäß.

Zur Gewährleistung eines sicheren Betriebes darf das Modul nur nach den in der allgemeinen Bedienungsanleitung und den in diesem Dokument dargestellten Angaben betrieben werden. Bei der Verwendung sind zusätzlich die für den jeweiligen Anwendungsfall erforderlichen Rechts- und Sicherheitsvorschriften zu beachten. Sinngemäß gilt dies auch bei Verwendung von Zubehör.

Vor jeder Inbetriebnahme der Module ist eine Projektierung und Risikoanalyse vorzunehmen die alle Sicherheitsaspekte der Automatisierungstechnik berücksichtigt. Dies betrifft vor allem den Personen- und Anlagenschutz.

Bei Anlagen, die aufgrund einer Fehlfunktion größere Schäden, Datenverlust oder sogar Personenschäden verursachen können, müssen zusätzliche Sicherheitsvorkehrungen getroffen werden. Im Fehlerfall stellen diese Vorkehrungen einen sicheren Betriebszustand her.

Dies kann z. B. durch mechanische Verriegelungen, Fehlersignalisierung, Grenzwertschalter usw. erfolgen.

Hinweis

Ein Modul darf nicht unmittelbar an ein Stromversorgungsnetz angeschlossen werden. Die Versorgungsspannung darf 10 V ... 30 V (DC) betragen.

Allgemeine Gefahren bei Nichtbeachten der Sicherheitshinweise

Jedes Modul entspricht dem Stand der Technik und ist betriebssicher. Von dem Modul können Restgefahren ausgehen, wenn es von ungeschultem Personal unsachgemäß eingesetzt und bedient wird. Jede Person, die mit Aufstellung, Inbetriebnahme, Wartung oder Reparatur des Modules beauftragt ist, muss die Bedienungsanleitungen und insbesondere die sicherheitstechnischen Hinweise gelesen und verstanden haben.

Der Leistungs- und Lieferumfang der Module deckt nur einen Teilbereich der Messtechnik ab. Sicherheitstechnische Belange der Messtechnik sind zusätzlich vom Anlagenplaner/Ausrüster/Betreiber so zu planen, zu realisieren und zu verantworten, dass Restgefahren minimiert werden. Jeweils existierende Vorschriften sind zu beachten. Auf Restgefahren im Zusammenhang mit der Messtechnik ist hinzuweisen. Nach Einstellungen und Tätigkeiten, die mit Paßwort geschützt sind, ist sicherzustellen, dass evtl. angeschlossene Steuerungen in einem sicheren Zustand verbleiben, bis das Schaltverhalten des Moduls geprüft ist.

Wartung und Reinigung

Die Module sind wartungsfrei. Beachten Sie bei der Reinigung des Gehäuses folgende Punkte:

- Trennen Sie vor der Reinigung die Verbindung zu allen Anschlüssen.
- Reinigen Sie das Gehäuse mit einem weichen und leicht angefeuchteten (nicht nassen!) Tuch. Verwenden Sie auf *keinen Fall* Lösungsmittel, da diese die Beschriftung oder das Gehäuse angreifen könnten.
- Achten Sie beim Reinigen darauf, dass keine Flüssigkeit in das Modul oder an die Anschlüsse gelangt.

Busanbindung und Ausgänge

Bei Anbindung an den CANbus sowie beim Senden von CANbus-Nachrichten muss in besonderem Maß auf die Sicherheit geachtet werden. Stellen Sie sicher, dass die reine Einbindung, Status- oder Steuersignale keine Aktionen vornehmen, die zu einer Gefahren für Mensch oder Umwelt führen.

Produkthaftung

In den folgenden Fällen kann die vorgesehene Sicherheit des Gerätes beeinträchtigt sein. Die Haftung für die Gerätefunktion geht dann auf den Betreiber über:

- Das Gerät wird nicht entsprechend der Bedienungsanleitung benutzt.
- Das Gerät wird außerhalb des in diesem Kapitel beschriebenen Anwendungsbereichs eingesetzt.
- Am Gerät werden vom Betreiber unautorisiert Änderungen vorgenommen.

Warnzeichen und Gefahrensymbole

Wichtige Hinweise für Ihre Sicherheit sind besonders gekennzeichnet. Beachten Sie diese Hinweise unbedingt, um Unfälle und Sachschäden zu vermeiden.

Sicherheitshinweise sind wie folgt aufgebaut:





Art der Gefahr

Folgen bei Nichtbeachtung

Gefahrenabwehr

- **Warnzeichen:** macht auf die Gefahr aufmerksam
- **Signalwort:** gibt die Schwere der Gefahr an (siehe folgende Tabelle)
- **Art der Gefahr:** benennt die Art oder Quelle der Gefahr
- **Folgen:** beschreibt die Folgen bei Nichtbeachtung
- **Abwehr:** gibt an, wie man die Gefahr vermeidet/umgeht

Gefahrenklassen nach ANSI

Warnzeichen, Signalwort	Bedeutung
 WARNUNG	Diese Kennzeichnung weist auf eine <i>mögliche</i> gefährliche Situation hin, die – wenn die Sicherheitsbestimmungen nicht beachtet werden – Tod oder schwere Körperverletzung zur Folge <i>haben kann</i> .
 VORSICHT	Diese Kennzeichnung weist auf eine <i>mögliche</i> gefährliche Situation hin, die – wenn die Sicherheitsbestimmungen nicht beachtet werden – leichte oder mittlere Körperverletzung zur Folge <i>haben kann</i> .
Hinweis	Diese Kennzeichnung weist auf eine Situation hin, die – wenn die Sicherheitsbestimmungen nicht beachtet werden – Sachschäden zur Folge <i>haben kann</i> .

Sicherheitsbewussten Arbeiten

Der Versorgungsanschluss, sowie Signal- und Fühlerleitungen müssen so installiert werden, dass elektromagnetische Einstrahlungen keine Beeinträchtigung der Gerätefunktionen hervorrufen (Empfehlung HBM "Greenline-Schirmungskonzept", Internetdownload <http://www.hbm.com/Greenline>).

Geräte und Einrichtungen der Automatisierungstechnik müssen so verbaut werden, dass sie gegen unbeabsichtigte Betätigung ausreichend geschützt bzw. verriegelt sind (z. B. Zugangskontrolle, Passwortschutz o.ä.).

Bei Geräten die in einem Netzwerk arbeiten sind diese Netzwerke so auszulegen, dass Störungen einzelner Teilnehmer erkannt und abgestellt werden können.

Es müssen hard- und softwareseitig Sicherheitsvorkehrungen getroffen werden, damit ein Leitungsbruch oder andere Unterbrechungen der Signalübertragung, z. B. über Busschnittstellen, nicht zu undefinierten Zuständen oder Datenverlust in der Automatisierungseinrichtung führen.

Fehlermeldungen dürfen nur quittiert werden, wenn die Ursache des Fehlers beseitigt ist und keine Gefahr mehr existiert.

Umbauten und Veränderungen

Das Modul darf ohne unsere ausdrückliche Zustimmung weder konstruktiv noch sicherheitstechnisch verändert werden. Jede Veränderung schließt eine Haftung unsererseits für resultierende Schäden aus.

Insbesondere sind jegliche Reparaturen, Lötarbeiten an den Platinen (Austausch von Bauteilen) untersagt. Bei Austausch gesamter Baugruppen sind nur Originalteile von HBM zu verwenden.

Das Modul wurde ab Werk mit fester Hard- und Softwarekonfiguration ausgeliefert. Änderungen sind nur im Rahmen der in den Handbüchern dokumentierten Möglichkeiten zulässig.

Qualifiziertes Personal



Wichtig

Dieses Gerät ist nur von qualifiziertem Personal ausschließlich entsprechend der technischen Daten in Zusammenhang mit den nachstehend aufgeführten Sicherheitsbestimmungen und Vorschriften einzusetzen bzw. zu verwenden.

Qualifizierte Personen sind Personen, die mit Aufstellung, Montage, Inbetriebsetzung und Betrieb des Produktes vertraut sind und über die ihrer Tätigkeit entsprechende Qualifikationen verfügen. Dieses Modul ist nur von qualifiziertem Personal ausschließlich entsprechend der technischen Daten in Zusammenhang mit den Sicherheitsbestimmungen und Vorschriften einzusetzen bzw. zu verwenden.

Dazu zählen Personen, die mindestens eine der drei folgenden Voraussetzungen erfüllen:

- Die Sicherheitskonzepte der Automatisierungstechnik werden als bekannt vorausgesetzt und sie sind als Projektpersonal damit vertraut.
- Als Bedienungspersonal der Automatisierungsanlagen sind sie im Umgang mit den Anlagen unterwiesen und mit der Bedienung der in dieser Dokumentation beschriebenen Modulen und Technologien vertraut.
- Als Inbetriebnehmer oder im Service eingesetzt haben sie eine Ausbildung absolviert, die Sie zur Repa-

ratur der Automatisierungsanlagen befähigt. Sie haben zusätzlich die Berechtigung, Stromkreise und Geräte gemäß den Normen der Sicherheitstechnik in Betrieb zu nehmen, zu erden und zu kennzeichnen.

Bei der Verwendung sind zusätzlich die für den jeweiligen Anwendungsfall erforderlichen Rechts- und Sicherheitsvorschriften zu beachten. Sinngemäß gilt dies auch bei Verwendung von Zubehör.





2 Verwendete Kennzeichnungen

catman[®] ist ein eingetragenes Warenzeichen der Hottinger Baldwin Messtechnik GmbH.

Alle in diesem Dokument verwendeten Warenzeichen oder Marken weisen nur auf das jeweilige Produkt oder den Inhaber des Warenzeichens oder der Marke hin. Hottinger Baldwin Messtechnik GmbH erhebt damit keinen Anspruch auf andere als die eigenen Warenzeichen oder Marken.

2.1 In dieser Anleitung verwendete Kennzeichnungen

Wichtige Hinweise für Ihre Sicherheit sind besonders gekennzeichnet. Beachten Sie diese Hinweise unbedingt, um Schäden zu vermeiden.

Symbol	Bedeutung
	Diese Kennzeichnung weist auf eine Situation hin, die – wenn die Sicherheitsbestimmungen nicht beachtet werden – Sachschäden zur Folge <i>haben kann</i> .
	Diese Kennzeichnung weist auf eine <i>mögliche</i> gefährliche Situation hin, die – wenn die Sicherheitsbestimmungen nicht beachtet werden – leichte oder mittlere Körperverletzung zur Folge <i>haben kann</i> .
 Wichtig	Diese Kennzeichnung weist auf <i>wichtige</i> Informationen zum Produkt oder zur Handhabung des Produktes hin.
 Tipp	Diese Kennzeichnung weist auf Anwendungstipps oder andere für Sie nützliche Informationen hin.

Gerät -> Neu	Fette Schrift kennzeichnet Menüpunkte sowie Dialog- und Fenstertitel in Programmoberflächen. Pfeile zwischen Menüpunkten kennzeichnen die Reihenfolge, in der Menüs und Untermenüs aufgerufen werden.
<i>Bitrate, 500</i>	Fett-kursive Schrift kennzeichnet Eingaben und Eingabefelder in Programmoberflächen.
<i>Hervorhebung Siehe ...</i>	Kursive Schrift kennzeichnet Hervorhebungen im Text und kennzeichnet Verweise auf Kapitel, Bilder oder externe Dokumente und Dateien.

3 QuantumX / SomatXR-Dokumentation

Die Dokumentation der QuantumX / SomatXR-Familie besteht aus

- der QuantumX and SomatXR-Bedienungsanleitungen im PDF-Format
- den Datenblättern im PDF-Format
- der Bedienungsanleitung des Datenrekorders CX22B-W
- der Bedienungsanleitung des EtherCAT®¹⁾ / Ethernet-Gateways CX27/B im PDF-Format
- den Produktbeschreibungen für Zubehörteile im PDF-Format
- einer umfangreichen Online-Hilfe mit Index und komfortabler Suchmöglichkeit, die nach Installation eines Softwarepaketes (z. B. MX-Assistent, catman®AP) zur Verfügung steht. Hier finden Sie auch Hinweise zur Konfiguration der Module und Kanäle.

Sie finden diese Dokumente

- auf der mit dem Gerät gelieferten QuantumX / SomatXR-System-CD
- nach Installation des MX-Assistenten auf der Festplatte ihres PCs, zu erreichen über Windows-Start
- immer aktuell auf unseren Internetseiten unter www.hbm.com/

¹⁾ EtherCAT® ist eine eingetragene Marke und patentierte Technologie, lizenziert durch die Beckhoff Automation GmbH, Deutschland

4 Der CANBus

Diese Anleitung unterstützt Sie beim Anschließen Ihres QuantumX- oder SomatXR-Systems an einen CAN-Bus mit den folgenden Modulen:

- MX471B(-R)
- MX840B(-R)

Im Folgenden werden diese Module als MX471B und MX840B bezeichnet. Die SomatXR-Module werden nur im Falle wesentlicher Unterschiede speziell genannt.

Bei Verwendung des Datenrekorders CX23-R gibt es einige Einschränkungen hinsichtlich der CAN-Bus-Optionen (siehe CX23-R-Anleitung).

Die Konfiguration eines CAN-Ports kann über die Software catman[®]EASY oder MX-Assistent vorgenommen werden, die über eine ausführliche Onlinehilfe verfügen.

Diese Anleitung zeigt Ihnen:

- Wie Sie einen CANbus-Knoten parametrieren.

Zusätzlich steht folgende Dokumentation zur Verfügung:

- Allgemeine Bedienungsanleitung
- Datenblätter MX840B oder MX471B
- Onlinehilfen in der Software catman[®]EASY und MX-Assistent

5 QuantumX / SomatXR und CAN

5.1 Übersicht

Das Modul MX471B bietet vier unabhängige CANbus-Knoten, die alle untereinander und zur Stromversorgung potentialgetrennt sind. Das Modul MX840B bietet am Kanal 1 einen CANbus-Knoten, der galvanisch getrennt ist.

Bei der Datenübertragung auf einem CANbus werden keine Teilnehmer direkt adressiert. Ein eindeutiger Identifier kennzeichnet den Inhalt einer Nachricht (z.B. Drehzahl oder Motortemperatur).

Der Identifier steht auch für die Priorität der Nachricht.

Nachricht = Identifier + Signal + Zusatzinformation

Teilnehmer am Bus = Knoten

Jeder Knoten kann entweder als Empfänger oder als Sender (Gateway) parametrierbar werden. Die Parametrierung im Detail wird in der jeweiligen Online-Hilfe des Softwarepakets dargestellt.

Hinweis

Um einen störungsfreien Betrieb zu gewährleisten, muss der CANbus an beiden Enden und nur dort mit einem entsprechenden Abschlusswiderstand terminiert werden. Ein 120 Ohm Abschlusswiderstand kann individuell im Modul per Software zugeschaltet werden. Eine Terminierung ist auch schon bei kurzen Leitungen mit niedrigen Bitraten erforderlich.

Im Datenblatt wird der Zusammenhang zwischen Bitrate und maximale Leitungslänge des Busses dargestellt.

Die Konfiguration eines Knotens bleibt auch nach dem Ab- und Anschalten der Module bestehen.

Sollten Sie Signale mit einer Rate größer als 2000/s dekodieren, richten Sie bitte die Signaleingänge 1 bis 8 auf dem MX471B ein. Bei diesen Signaleingängen wurden hierfür die Signalpuffer vergrößert.

5.2 Anschluss CANbus

CAN-Signale empfangen:

- MX471B, MX840B (Kanal 1)

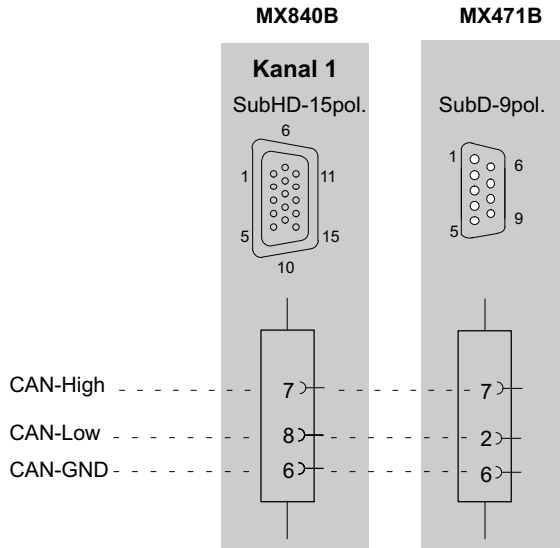
CAN-Signale senden:

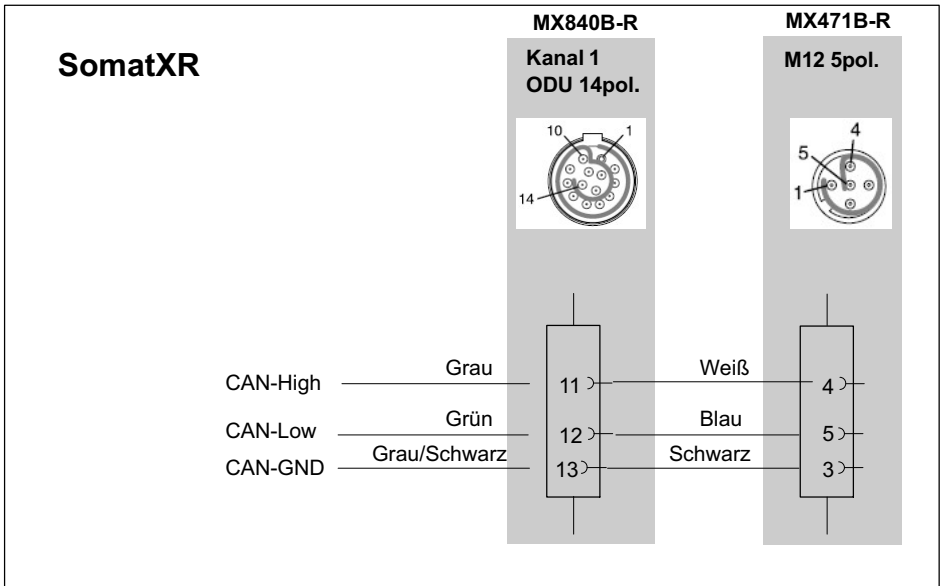
- MX471B
- MX840B (nur modul-interne Messsignale)
- die Software MX-Assistent kann aus der Liste aller versendeten Nachrichten ein DBC-file generieren

CCP oder XCP-over-CAN-Signale empfangen:

- nur MX471B

QuantumX





Hinweis

Sorgen Sie für eine korrekte Terminierung mit Abschlusswiderständen, wie in Abb. 5.1 dargestellt. Das QuantumX-Modul MX840B enthält keine Terminierung, der MX471B und der SomatXR MX840B-R enthalten eine interne Terminierung, die über Software aktiviert werden kann.

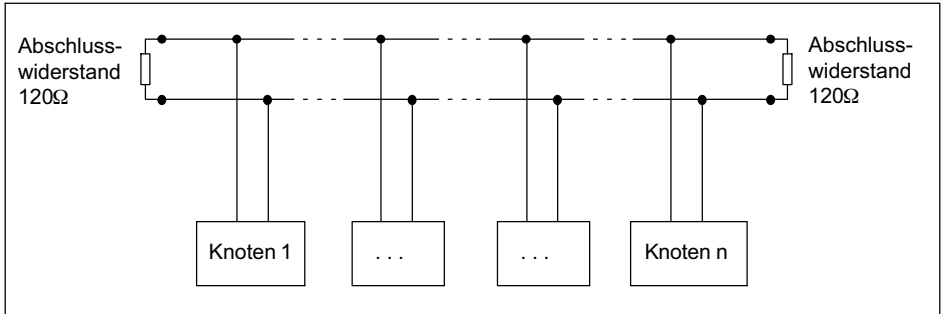


Abb. 5.1 Busabschlusswiderstände

Zum Anschluss der D-SUB-15HD-Gerätebuchsen des QuantumX MX840B an handelsübliche D-SUB-Stecker mit standardisierter CiA-Belegung dient das Adapterkabel 1-Kab418.

5.2.1 Zustandsanzeige LEDs der Geräte

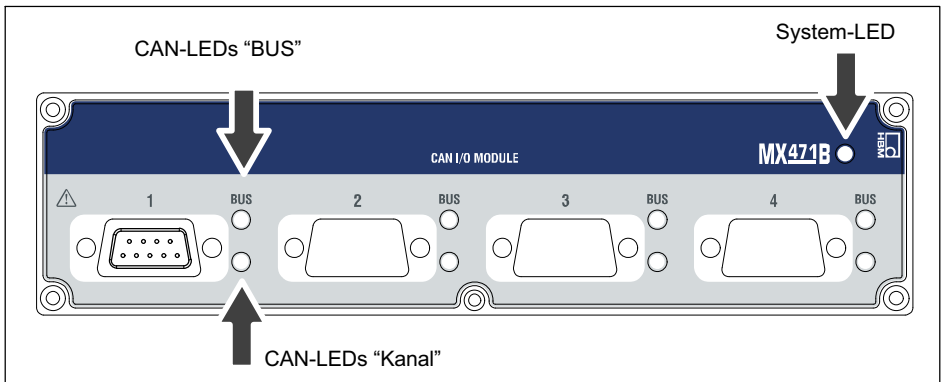


Abb. 5.2 QuantumX Frontansicht MX471B

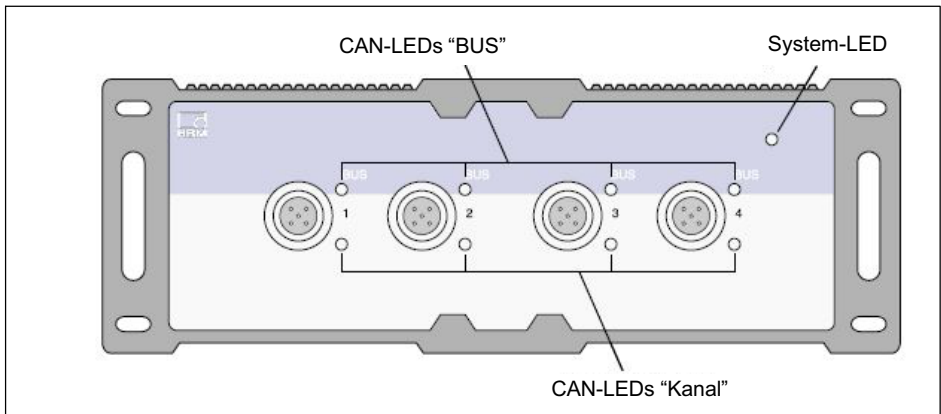


Abb. 5.3 SomatXR Frontansicht MX471B-R

Modulbezogene Zustandsanzeigen (Fehlermeldungen)
siehe auch Kapitel 6.3.

System-LED

Grün	Fehlerfreier Betrieb
Gelb	System ist nicht bereit, Bootvorgang läuft
Gelb blinkend	Download aktiv, System ist nicht bereit
Rot	Fehler, Synchronisation fehlerhaft

CAN-LEDs (BUS)

Grün flackert	Bus ist fehlerfrei und Aktivität auf CAN
Grün dauerhaft an	Bus ist fehlerfrei und keine Aktivität auf CAN
Gelb flackert	Bus ist zeitweise fehlerhaft (Warning) und Aktivität auf CAN
Gelb dauerhaft an	Bus ist zeitweise fehlerhaft (Warning) und keine Aktivität auf CAN
Rot an	Bus ist fehlerhaft, das CAN-Interface ist im Zustand "Bus-OFF"

CAN-LEDs (Kanal)

Grün dauerhaft an	Kanal ist betriebsbereit
Gelb blinkend	Firmware1-Download aktiv
Gelb an	Bootvorgang läuft
Rot an	Kanal ist fehlerhaft

Ethernet-LED (nur QuantumX)

Grün an	Ethernet Linkstatus ist in Ordnung
Gelb blinkt	Ethernet Datenübertragung läuft

5.2.2 CAN-Nachrichten empfangen

Um CAN-Nachrichten empfangen zu können, müssen die relevanten Nachrichten dem Knoten bekannt gemacht werden. Das kann direkt auf dem Knoten oder wiederholbar über vorher angelegte Nachrichten in der Sensordatenbank erfolgen. Aus der Sensordatenbank können einzelne Nachrichten per drag & drop mit dem Knoten verbunden werden.

Es können auch CAN-Datenbasen vom Typ *.dbc in die Sensordatenbank eingelesen werden. Steht keine CAN-Datenbasis zur Verfügung, kann diese auch selbst erstellt werden. Unterschiedliche Firmen bieten hierzu Editoren an.

Im Messbetrieb werden die empfangenen CAN-Nachrichten sofort "zeitgestempelt". Damit ist im Gesamtsystem eine parallele und synchrone Erfassung und Analyse von direkt erfassten Messgrößen und von CAN-Nachrichten möglich.

6 Funktionsbeschreibung

6.1 Globale Parameter

6.1.1 Bitraten

Seit Firmware-Version 4.6 wird die Parametrierung der CAN-Bitrate erweitert. In einer Ansicht kann man eine gewünschte Bitrate, sowie den Abtastpunkt und die Synchronisationsweite (SJW) wählen.

Zur Parametrierung muss eine Bitrate (Wert in bit/s) angegeben werden. Zusätzlich wählt man mit „SamplePointRatio“ ein gewünschten Abtastpunkt und mit „SJW“ eine Synchronisationsweite zwischen 1 und 4 Time-Quanta.

Nachdem die Parametrierung abgesendet wurde, wird diese im Modul geprüft und eine Einstellung vorgenommen, die den gewünschten Parametern möglichst nahe kommt. Man kann nun die Parametrierung zurücklesen und aus den entsprechenden XML-Tags mit vorangestelltem „Active“ die derzeit tatsächlich eingestellten Werte auslesen. Es steht dem Anwender frei, diese zurück gelesenen Werte nach eigenen Anforderungen zu bewerten.

Werksseitig ist gewählt: Bitrate = 1000 kbit/s, SamplePointRatio = 87,5 %, SJW = 4.

6.1.2 Terminierung des CANBus

Alle Varianten von MX471B, sowie MX840B-R können eine Terminierung des CAN-Bus per Parametrierung zu- oder abschalten. MX840, MX840A und MX840B unterstützen keine interne Terminierung.

6.1.3 Fehlerbehandlung

Es gibt Anwendungen, bei denen einzelne Fehlerereignisse kritisch sind und dauerhaft angezeigt werden müssen, auch wenn die Fehlerursache längst behoben ist. In anderen Anwendungen wünscht man sich, dass die LED und der Fehlerstatus den momentanen Zustand des CAN-Bus und Moduls anzeigt und Fehler automatisch gelöscht werden, wenn deren Ursache beseitigt ist.

Im MX-Assistent kann man daher in den „CAN-Bus-Einstellungen“ unter „Fehlermodus“ wählen, wie sich das Modul beim Auftreten kurzzeitiger Fehler verhalten soll: Wird die Methode **„automatisch, verzögert“** gewählt, wird der Fehler nach der einstellbaren Verzögerungszeit automatisch aus dem Fehlerstatus gelöscht und die LED wechselt in den Normalzustand, nachdem die Fehlerursache beseitigt wurde. Das Modul zeigt demnach mit der eingestellten Verzögerung des aktuellen Zustand des CAN-Bus.

Wird die Methode **„beim Lesen“** gewählt, bleibt ein Fehlerstatus so lange bestehen und wird auch über die LED signalisiert, bis der Fehlerstatus aus dem Modul ausgelesen wurde. Sofern die Ursache des Fehlers nicht mehr besteht, wird also der Fehlerstatus und die LED erst dann in den Normalzustand wechseln, wenn der Fehlerstatus ausgelesen wurde. Auf diese Weise können auch einzelne Fehlerereignisse sicher erfasst werden, ohne dass der Fehlerstatus während einer Messung dauerhaft beobachtet werden muss. Allerdings gibt diese Methode nicht immer den aktuellen Zustand des CAN-Bus wider.

6.2 Fehlerereignisse

6.2.1 Erfassen von Fehlern des Sende- und Empfangsweges

Fehler in Senderichtung (vom Modul in den CAN-Bus) werden nur dann überwacht, wenn in der Parametrierung des Moduls Daten zum Versenden konfiguriert wurden. Umgekehrt werden nur dann Fehler in Empfangsrichtung (CAN-Bus in das Modul) detektiert, wenn mind. ein CAN-Empfangsdekoder aktiv konfiguriert ist.

6.2.2 Verhalten der LED und des Fehlerstatus

Die LED leuchtet gelb, wenn BUS-Warning auftritt, rot in allen anderen Fehlerfällen.

Timeout und „Loss of signal“ werden mit gelber LED signalisiert, sofern die Überwachung aktiviert wurde. Dies erfolgt damit, dass in der Parametrierung eines Dekoders der Wert „MaxRepetitionTime“ größer „0“ gewählt wurde.

Das Fehlerregister kann per MX-Assistent ausgelesen werden. Die LED folgt dabei den Statusmeldungen im Fehlerregister, siehe Kapitel 3.

Fehlerereignisse treten oft nur sporadisch und kurzzeitig auf. Wenn die Anwendung fordert, dass auch einmalige Ereignisse sicher erkannt werden können, kann in der Parametrierung (siehe Kapitel 1.3) der „Fehlermodus“ so eingestellt werden, dass Fehler erst mit dem Auslesen des Fehlerstatus gelöscht werden.

6.2.3 Mögliche Fehlerursachen im CAN-Betrieb

6.2.3.1 CANBus Warning, CANBus Error, CANBus OFF

Im CAN-Controller des Moduls zählt ein interner Zähler Bus-Fehler entsprechend der CAN-Norm. Beim Überschreiten der 1. Schwelle wird „Warning“ gesetzt, danach „Error“. Wenn die Fehlerursache rechtzeitig behoben wird, werden Warning und Error automatisch wieder gelöscht.

Bleibt der Fehler weiterhin bestehen, wird schließlich „BUS-OFF“ gesetzt. Damit wird der Bus inaktiv/abgeschaltet. Nach Beheben der Fehlerursache wird der Zustand im Modul durch externen BUS-Reset, Parametrierung oder Neustart des Moduls gelöscht. Ein BUS-Reset kann von allen Varianten MX471B und MX840B, jedoch nicht von MX840 und MX840A ausgelöst werden.

Fehler auf dem CAN-Bus werden z. B. dann im CAN-Controller gezählt, wenn beim Senden keine Gegenseite korrekt antwortet (weil es keinen Abnehmer gibt, der mit Acknowledges antwortet), oder der Bus durch Kurzschluss, falschen Busabschluss, gemischten Betrieb verschiedener Bitraten oder andere (meist) elektrische Eigenschaften gestört ist.

6.2.3.2 CAN „Receiver Overrun“

Wenn die Daten nicht schnell genug vom CAN-Controller im Modul ausgelesen werden können, gehen Daten verloren, die auf dem CAN-BUS anliegen und das Flag „Overrun“ wird gesetzt. Zu diesem Zeitpunkt ist nicht bekannt, ob die verloren gegangenen Daten relevant

sind, d. h. der CAN-Identifizier zu dieser Nachricht im Dekoder parametrier ist.

Nachdem die Daten vom CAN-Bus gelesen wurden, werden sie in einen Zwischenpuffer gelegt, aus dem dann regelmäßig Nachrichten gelesen und dekodiert werden. Im Mittel können mit allen Varianten des MX840A-Moduls 20000 Messwerte pro Sekunde aus CAN-Daten dekodiert werden. Wird dieser Wert überschritten, läuft der Zwischenpuffer über, was zu einer Fehlermeldung führt.

6.2.3.3 CAN „Transmitter Overrun“

Wenn Sende-Daten nicht korrekt verschickt werden können, weil z. B. der Bus gestört oder überlastet ist, wird dieser Fehler in den Fehlerstatus eingetragen

6.2.3.4 CAN Dekoder „Timeout“

Über den Parameter „MaxRepetitionTime“ wird ein Timeout festgelegt. Immer wenn eine CAN-Nachricht dekodiert wurde, wird die Zeit zum vorherigen Empfang dieser Nachricht bestimmt. Ist sie größer als dieser Timeout, wird ein ungültig markierter Messwert in den Signalstrom eingefügt und ein Fehler signalisiert.

6.2.3.5 CAN Dekoder „Loss of Signal“

Im Gegensatz zum „Timeout“ folgt beim „Loss of signal“ keine weitere dekodierte CAN-Nachricht. Für MX840B) gilt: Wenn das Signal innerhalb 128 ms völlig ausfällt, wird dies detektiert und ein ungültig markierter Messwert in den Signalstrom eingefügt. Sobald die nächste Nachricht dekodiert wurde, wird dieser Fehler wieder gelöscht.

6.2.3.6 Modul-Ressourcen

Für MX471B gilt, dass in Summe über alle CAN-Anschlüsse pro Sekunde bis zu 100.000 Signale dekodiert und CAN-Nachrichten zum Versenden erzeugt werden können. Wird dieser Wert überschritten, wird das Dekodieren und Erzeugen der CAN-Nachrichten solange unterbrochen, bis wieder weniger Signale empfangen werden bzw. erzeugt werden sollen. Damit wird sichergestellt, dass das Modul auch im Überlastungsfall ansprechbar bleibt. Im Modul-Fehlerstatus wird ein entsprechender Fehlercode eingetragen, und alle Status-LED leuchten dauerhaft rot.

6.3 Zustandsanzeige per LED

Unterschieden werden Fehlermeldungen allgemeiner Art (siehe Kapitel 5.2.1), wie z.B. fehlerhafte Bitrateneinstellung, und Fehler, die abhängig von Empfangs- oder Sende-Parametrierung sind. Ist kein Transmit-Kanal parametriert, werden Fehler in Senderichtung ignoriert. Umgekehrt erfolgt keine Fehlersignalisierung aus dem CAN-Empfang, wenn in Empfangsrichtung keine CAN-Signale parametriert sind.

Wenn die Anschluss-LED im Normalbetrieb nicht grün leuchtet, sind in der Regel Fehlermeldungen vorhanden, die über den Fehlerstatus ausgelesen werden können.

6.3.1 MX840B

Für MX840B ist das Verhalten der Kanal-LED im CAN-Betriebsmode beschrieben. Das Flackern der LED bei Aktivität auf dem CAN-Bus ist nicht mit einzelnen Nachrichten synchronisiert.

Anschluss-LED	Funktion
dauerhaft grün	CAN-BUS aktiviert, keine Fehler oder Ausfälle
flackernd grün	CAN-BUS aktiviert, keine Fehler, Aktivität auf CAN. (Seit Firmware 4.0)
dauerhaft orange	CAN-Controller im Zustand „BUS WARNING“. Zeitüberwachung eines Empfangssignals meldet dauerhaften oder kurzzeitigen Ausfall der externen CAN-Signalquelle.
orange flackernd	CAN-Controller zeitweise fehlerhaft („BUS WARNING“). Aktivität auf CAN. (Seit Firmware 4.0)
orange blinkend	Firmware wird aktualisiert. Modul erst nach Aufforderung durch das Update-Programm ausschalten!
dauerhaft rot	CAN-Controller im Zustand „BUS ERROR“ oder Datenverlust im Empfänger aufgrund von Pufferüberlauf. Sendedaten gehen verloren, weil kein CAN-Abnehmer die CAN-Nachrichten abnimmt oder aufgrund anderer Störungen des CAN-BUS.
rot blinkend	CAN-Controller meldet „BUS OFF“, kein Empfang oder Versand von CAN-Nachrichten möglich. Bitrate aller angeschlossenen Teilnehmer, Verpolung oder Kurzschlüsse prüfen. CAN-Controller wird evtl. erst nach einem Rücksetzen wieder aktiviert. Blinken zusätzlich die übrigen Kanal-LED und die System-LED rot, liegt ein Hardware-Defekt vor.

6.3.2 MX471B

BUS-LED	Funktion
dauerhaft grün	CAN-BUS aktiviert, keine Fehler, keine Aktivität auf CAN
flackernd grün	CAN-BUS aktiviert, keine Fehler, Aktivität auf CAN
dauerhaft orange	CAN-Controller zeitweise fehlerhaft („BUS WARNING“). Keine Aktivität auf CAN.

BUS-LED	Funktion
orange flackernd	CAN-Controller zeitweise fehlerhaft („BUS WARNING“). Aktivität auf CAN.
dauerhaft rot	CAN-Controller meldet „BUS OFF“, kein Empfang oder Versand von CAN-Nachrichten möglich.

Anschluss-LED	Funktion
dauerhaft grün	CAN-BUS aktiviert, keine Fehler oder Ausfälle
dauerhaft orange	CAN-Controller im Zustand „BUS WARNING“. Zeitüberwachung eines Empfangssignals meldet dauerhaften oder kurzzeitigen Ausfall der externen CAN-Signalquelle.
orange flackernd	CAN-Controller zeitweise fehlerhaft („BUS WARNING“). Aktivität auf CAN. (Seit Firmware 4.0)
orange blinkend	Firmware wird aktualisiert. Modul erst nach Aufforderung durch das Update-Programm ausschalten!
dauerhaft rot	CAN-Controller im Zustand „BUS ERROR“ oder Datenverlust im Empfänger aufgrund von Pufferüberlauf. Sendedaten gehen verloren, weil kein CAN-Abnehmer die CAN-Nachrichten abnimmt oder aufgrund anderer Störungen des CAN-BUS. Rechen-Ressource im Modul überschritten: Zu viele Signale müssen dekodiert werden und/oder zu viele CAN-Nachrichten sollen versendet werden.
rot blinkend	CAN-Controller meldet „BUS OFF“, kein Empfang oder Versand von CAN-Nachrichten möglich. Bitrate aller angeschlossenen Teilnehmer, Verpolung oder Kurzschlüsse prüfen. CAN-Controller wird evtl. erst nach einem Rücksetzen wieder aktiviert. Blinken zusätzlich die übrigen Kanal-LED und die System-LED rot, liegt ein Hardware-Defekt vor.

6.4 CAN-Dekoder: Empfang von CAN-Daten

6.4.1 Remote-Frames (RTR)

Remote-Frames sendet ein CAN-Master, um Daten zeitnah anzufordern. Kein MX-Modultyp unterstützt Remote-Frames (RTR). Diese Nachrichten werden beim Empfang verworfen.

6.4.2 Benutzerdefinierte Auswahl der Datentypen

Im CAN-Dekoder wird durchgängig unterschieden, in welchem Datenformat die Daten aus der CAN-Nachricht extrahiert und in welchem Datenformat sie als Signal im QuantumX / SomatXR-System verfügbar sind.. Je nach Kombination beider Datentypen verhält sich der CAN-Dekoder unterschiedlich. Das mag zunächst verwirrend sein, bietet aber die Möglichkeit, dass Integer-Werte durchgängig als solche behandelt werden können, um die Verarbeitung von Digital-IO-Daten nicht zu verfälschen, und Fließkomma-Berechnungen mit bestmöglicher Genauigkeit erfolgen.

Nachdem die Daten-Byte vom CAN-Bus empfangen wurden, werden diese so behandelt, wie es in der Parametrierung als Rohdatentyp („RawValueFormat“) angegeben wurde. Das Signal, das später systemweit verwendet wird, wird durch das Ausgabeformat („SignalFormat“) bestimmt. Zudem werden die Integer-Datentypen auch darauf geprüft, ob diese vorzeichenbehaftet (UINT32 und UINT64) sind oder nicht (INT32 und INT64).

Der Datentyp zum Interpretieren des Mode-dependent-Signal wird immer als UINT64 angenommen. Die Auswahl der Mode-Länge kann dabei zwischen 1 und 64 gewählt werden. Die Skalierung des „ModeValue“ ist nicht möglich; und erscheint auch nicht sinnvoll.

Ist die gewählte Kombination aus Startbit und Länge ungültig, wird der CAN-Dekoder nicht auf die Umrechnungsvorschrift umgestellt und eine Fehlermeldung gesetzt. Der CAN -Dekoder arbeitet solange mit der alten Vorschrift weiter, bis eine gültige übertragen wurde.

6.4.3 Datentypen der Rechenvorschrift

Seit Firmware-Version 4.3.1 gelten die in der nachfolgenden Tabelle beschriebenen Regeln für die Umrechnung der Datentypen unter Berücksichtigung der Skalierrechnung:

```
signal_value = ( raw_value * factor ) +  
offset
```

Damit die Berechnung möglichst genau bleibt und auch bei der Verwendung einer Skalierung Integer-Werte nicht verfälscht werden, wird die Skalierrechnung in zwei Verfahren aufgeteilt. Dabei werden die Datentypen für „raw“, „factor“ und „offset“ unterschiedlich verrechnet.

6.4.4 Fließkomma-Skalierung

Sofern wenigstens einer der Werte von „factor“ oder „offset“ mit einem Fließkommawert parametrisiert wurde, wird das CAN-Rohsignal „raw_value“ zunächst in einen Fließkommawert (REAL64) umgewandelt. „factor“ und „offset“ sind in diesem Fall immer vom Fließkomma-

Datentyp (REAL64). Das Zwischenergebnis wird schließlich in den Datentyp umgewandelt, der als „SignalFormat“ angegeben wurde. Rundungsfehler sind hierbei durch die Anwendung von Fließkommazahlen prinzipbedingt.

Ist der Datentyp für „SignalFormat“ 64-bit Ganzzahl, werden die Nachkommastellen des Fließkommawerts vor der abschliessenden Umwandlung von REAL64 nach UINT64 bzw. INT64 abgeschnitten. In allen anderen Fällen wird der Fließkommawert zunächst auf eine Ganzzahl gerundet.

Nachfolgende Tabelle gilt ausschließlich für die hier beschriebene „Fließkomma-Skalierung“:

Signalformat	Datenformat der Rohdaten	Rohdaten werden umgewandelt in Rechenformat	Verwendetes Datenformat für „factor“ und „offset“
REAL32¹⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
REAL64¹⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
INT32¹⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
UINT32¹⁾	REAL32	REAL64	REAL64
	REAL64		
	INT32		
	INT64		
	UINT32		
	UINT64		

INT64²⁾	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
UINT64²⁾	REAL32	REAL64	REAL64
	REAL64		
	INT32		
	INT64		
	UINT32		
	UINT64		

- 1) Bei der Umrechnung vom Zwischenwert (REAL64) in das 64-bit große Datenformat des Signalformats wird auf die nächste Ganzzahl gerundet.
- 2) Bei der Umrechnung vom Zwischenwert (REAL64) in das 64-bit große Datenformat des Signalformats werden die Nachkommastellen abgeschnitten.

REAL32: single precision floating point, size 32 bit,
gemäß IEEE 754

REAL64: double precision floating point, size 64 bit,
gemäß IEEE 754

Umsetzung des farbig hervorgehobenen Beispiels im
Modul:

```
C:    int32_t raw;
        double factor, offset;
        int32_t signal = (int32_t)
        (round((double)raw * factor) +
        offset) );
```

```
Pascal: var
        raw: LongInt;
        factor, offset: Double;
        signal: LongInt
    begin
        signal:= LongInt(Round ((Int64(raw)
        * factor) + offset) );
```

6.4.5 CAN-Rohdaten-Empfang

Hiermit stehen alle empfangenen CAN-Nachrichten ungefiltert als Signal zur Verfügung. Das Signal kann derzeit per Streaming-Protokoll (<DaqAvailable>) versendet werden, aber nicht innerhalb des Modulverbands über Firewire (<RtAvailable>).

Ist der Rohdaten-Empfang aktiviert, erscheint in der Signalliste das Signal für Connector 2 mit Referenz: „CanRawReceiver_Connector2.Signal1“.

6.4.5.1 CAN-Rohdaten im Signal

Das Signal stellt folgende Daten zur Verfügung:

CAN-Id, wobei Bit 31 das Frame-Format bestimmt:

Bit 31 = 0: Base Format; Bit 0010: CAN-Identifizier.

Bit 31 = 1: Extended Format; Bit 0028: CAN-Identifizier

DLC: Anzahl der Daten-Byte; Wert zwischen 0 und 8

Daten: Abhängig von „DLC“ werden 0 bis 8 Byte Daten im Signal gesendet.

Der **Zeitstempel** beim Datenempfang ist Teil der übergeordneten Streaming-Daten.

Details zur Codierung des Signals sind dem Dokument zur Definition des Streaming-Protokolls zu entnehmen.

6.4.5.2 Parametrierung

Die Parametrierung schaltet den Empfang der CAN-Rohdaten individuell für jeden CAN-Connector ein und aus (<Active>). Außerdem enthält das Signal wie viele andere Signale im QuantumX / SomatXR-System einen Namen

(<ChannelName>) und eine Identifikation, ob dieser Name durch den Anwender gesetzt wurde oder aus der Werkseinstellung stammt (<OriginOfName>).

6.4.6 Ganzzahl-Skalierung

Wenn sowohl der Wert für „factor“ als auch „offset“ mit einer Ganzzahl parametrierung wurde, wird die Integer-Genauigkeit besonders berücksichtigt. Ziel ist es, Rundungsfehler zu vermeiden, die kritisch sind, wenn die Integer-Werte boolesche Zustände darstellen.

Zunächst werden ebenfalls die Rohdaten „raw_value“ im Rohdatenformat („RawValueFormat“) vom CAN-Bus genommen werden, danach aber abhängig vom Datentyp von „signal_value“ („SignalFormat“) in ein Rechenformat umgewandelt, wie es in nachfolgender Tabelle genannt wird. Ebenso werden „factor“ und „offset“ mit dem in der Tabelle genannten Datentyp gerechnet. Das Zwischenergebnis wird schließlich in das Datenformat umgewandelt, das als „SignalFormat“ angegeben wurde.

Nachfolgende Tabelle gilt ausschließlich für die hier beschriebene „Ganzzahl-Skalierung“.

Signalformat	Datenformat der Rohdaten	Rohdaten werden umgewandelt in Rechenformat	Verwendetes Datenformat für „factor“ und „offset“
REAL32	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
REAL64	REAL32	REAL64	REAL64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
INT32	REAL32	REAL32	INT32
	REAL64	INT64	
	UINT32	INT32	
	INT32		
	UINT64	INT64	
	INT64		
UINT32	REAL32	REAL32	INT32
	REAL64	REAL64	
	INT32	INT32	
	INT64	INT64	
	UINT32	UNIT32	UNIT32
	UINT64	UNIT64	

INT64	REAL32	INT64	INT64
	REAL64		
	UINT32		
	INT32		
	UINT64		
	INT64		
UINT64	REAL32	INT64	INT64
	REAL64	INT32	
	INT32	INT64	
	INT64	UNIT64	UNIT64
	UINT32		
	UINT64		

REAL32: single precision floating point, size 32 bit, gemäß IEEE 754

REAL64: double precision floating point, size 64 bit, gemäß IEEE 754

Umsetzung des farbig hervorgehobenen Beispiels im Modul:

```
C:    double raw;
        int32_t factor, offset;
        int32_t signal = (int32_t)
        ( ((int64_t)raw * factor) +
        offset );
```

```
Pascal: var
        raw: Double;
        factor, offset: LongInt;
        signal: LongInt
    begin
        signal := LongInt((Int64(raw)
        * factor) + offset );
```

6.5 CAN-Encoder (Mappable CAN-Sende-Nachrichten)

6.5.1 Motivation

Seit Firmware-Version 4.6 können mit MX471B mehrere Signale innerhalb einer CAN-Nachricht versendet werden und Messwerte auch in ein Ganzzahl-Format umgewandelt werden, wobei die resultierende Signallänge 1 bis 64 Bit betragen kann.

Mit dem allgemein definierten CAN-Encoder ist es möglich, bis zu 128 Signalquellen in bis zu 128 verschiedenen CAN-Nachrichten zu versenden, wobei mehrere verschiedene Signalquellen innerhalb einer einzelnen CAN-Nachricht „gemappt“ werden können. Pro CAN-Nachricht können somit bis zu 64 Signalquellen versendet werden. Die Anzahl der Signalquellen ist aufgrund Kompatibilität zur vorherigen <CanTransmit>-Implementierung bestimmt und durch systeminterne Ressourcen begrenzt.

Für jede CAN-Nachricht wird durch den Anwender ein individueller CAN-Identifizier <Identifizier> parametrisiert, ebenso das Frame-Format <ExtendedFrame> (Base-/Extended-Identifizier) und ein Name <Channel-Name> zur Anzeige in den Oberflächen der PC-Software.

6.5.2 Definition der Signalquellen

Was wird wo gesendet?

In jeder CAN-Nachricht können Daten aus mehreren verschiedenen Quellen gesendet werden. Da auch Daten mit 1 bit Länge „gemappt“ werden können, sind bis zu 64

verschiedene Quellen in einer einzigen CAN-Nachricht denkbar. Für jede Quelle wird die Signalquelle im System (z. B. „ModuleReference“ = „UUID = 1234“, „SignalReference“ = „AnalogIn_Connector1.Signal2“) definiert, das Datenformat <DataFormat> (z. B. Real32, Integer32) und die Art der Codierung <BitSequence> (Intel/Motorola). Die Quellen können Messwerte aus dem QuantumX / SomatXR-System sein, die per FireWire übertragen wurden, aber auch dekodierte Daten, die vom CAN-Bus empfangen wurden.

Eine Signalquelle wird typischerweise einem anderen MX-Modul entnommen. Dazu muss „ModuleReference“ gesetzt sein. Zeigt die „ModuleReference“ auf das eigene Modul oder ist deren Angabe leer, können Daten, die auf diesem oder einem anderen CAN-Bus-Anschluss innerhalb dieses Moduls empfangen werden, als Sende-Datenquelle dienen. Das Modul verhält sich damit als „Gateway“.

Für jede Quelle wird festgelegt, an welche Stelle und mit wie vielen Bits in der CAN-Nachricht die Daten geschrieben werden, siehe Kap. 5.4.

In welcher CAN-Nachricht diese so definierten Daten versendet werden, wird mit den Parametern <Identifier> und <ExtendedFrame> festgelegt. Diese Parameter finden sich ebenso im XML-Sub-Tree <CanMessage>, der die CAN-Nachricht und deren Versand beschreibt. Auf diese Weise muss die <CanMessage> nur einmal parametrisiert werden. Verschiedene Signal-Quellen weisen direkt auf die jeweilige CAN-Nachricht.

6.5.3 Skalierung des Messwertes

Analog zum Dekoder des CAN-Empfängers können die Signale von der Quelle mit <Factor> und <Offset>

skaliert werden, bevor sie gesendet werden. Dies ist vor allem wichtig, will man einen Fließkommawert als Integer übertragen, was in der CAN-Welt weit verbreitet ist. Will man z. B. den Fließkommawert mit einer Genauigkeit von 3 Nachkommastellen als Integer versenden, denkt man sich einen Dezimalpunkt an der 3. Stelle des Integer-Werts und multipliziert das Quellsignal mit 1000. („1.2345“ wird zu „1234“.)

Die Skalierung erfolgt mit Double-Genauigkeit (REAL64), d.h. dass der resultierende Ganzzahlwert in den unteren Bits gerundet sein wird.

Die Wahl des Datentyps UINT64 garantiert eine eins-zu-eins-Übertragung aller Bits, wobei auf die Skalierung verzichtet wird. Die Werte in <Factor> und <Offset> werden ignoriert.

6.5.4 Datentypen und Bit-Positionen eines Messwertes

Messwerte liegen im System üblicherweise als REAL32 (float) vor. Sollen die Messwerte auch als Fließkommawert gesendet werden, ist das Datenformat <DataFormat> entsprechend zu wählen. Die Anzahl der zu sendenden Bits ist damit in <SignalLength> auf 32 (float) bzw. 64 (double) festgelegt.

Es ist auch möglich, eine Typkonvertierung vorzunehmen, sodass die Daten in der CAN-Nachricht als Ganzzahlen versendet werden, deren Länge beliebig zwischen 1 und 64 Bit betragen kann. Die Auflösung des Messwerts als Ganzzahl ist also variabel. Dies leistet eine differenzierte Datentyp-Umwandlung.

Zuerst wird der Messwert als Fließkommawert skaliert. Wenn das Sende-Datenformat <DataFormat> als Integerwert gewählt ist, erfolgt erst jetzt die Umrechnung

in einen 32- bzw. 64-Bit großen Integer. Beginnend mit dem MSB des resultierenden Integer-Werts werden dann so viele Bits in die CAN-Nachricht gemappt, wie in `<SignalLength>` bestimmt wird. Es werden also immer die signifikantesten Bits eines Messwerts übertragen.

Der Parameter `<StartBit>` schließlich definiert, an welcher Stelle die resultierenden Daten in die CAN-Nachricht eingefügt werden und `<BitSequence>`, welche Regel dazu angewandt werden soll (INTEL bzw. MOTOROLA, siehe Kap. 6).

Die Parameter `<DataFormat>`, `<StartBit>`, `<SignalLength>` und `<BitSequence>` haben im CAN-Decoder entsprechende Bedeutung, weshalb auch hier keine anderen Bezeichnungen für die Parameter gewählt werden.

6.5.5 Sendedaten im Fehlerfall

Wenn die Signalquelle noch keinen Wert geliefert hat, oder die Signalquelle einen Fehlerwert sendet (z. B. aufgrund Übersteuerung oder nicht gestecktem Sensor), muss ein definierter Wert in die CAN-Nachricht eingetragen werden. Der Anwender kann diesen Wert mit den XML-Parametern zu `<ValueOnError>` wählen.

„Internal“: Die Auswahl des Fehlerwerts ist durch das Modul festgelegt. Bestimmt `<DataFormat>` einen Fließkommawert, wird im Fehlerfall „2,0E15“ gesendet. Ist `<Type>` auf „BitArray“ gesetzt oder `<DataFormat>` ist auf einen Integer oder Boolean parametrieren, wird „0“ gesendet. Dies entspricht dem Verhalten des Moduls, als `<ValueOnError>` noch nicht existierte und ist die Werkseinstellung.

„Float“: Diese Auswahl ist nur zulässig, wenn <Data-Format> einen Fließkommawert bestimmt. Dann wird im Fehlerfall der Wert aus <Value> gesendet.

„Integer“: Diese Auswahl ist nur zulässig, wenn <Data-Format> einen Integer-Wert bestimmt. Dann wird im Fehlerfall der Wert aus <Value> gesendet.

„Hex“: Diese Auswahl ist für alle Werte von <Data-Format> gültig. Im Fehlerfall wird der Hex-String unter Berücksichtigung von <BitSequence> als hexadezimalen Zahlenwert gesendet. Die Anzahl der Bits ist dabei von der <SignalLength> der Quelle bestimmt. Gesendet werden die Bits des Fehlerwerts beginnend mit Bit 0 des Fehlerwerts.

6.5.6 Parameter der CAN-Nachricht

Wie werden die CAN-Nachrichten gesendet?

Im Sub-Tree <Source> werden die *Inhalte* einer CAN-Nachricht definiert, also welche Quelldaten an welcher Stelle in welcher CAN-Nachricht „gemappt“ werden. Im Sub-Tree <CanMessage> wird der Header der CAN-Nachricht definiert.

Die Parameter <Identifizier> und <ExtendedFrame> im Sub-Tree <Source> legen fest, in welche <CanMessage> diese Signalquelle „gemappt“ werden soll.

6.5.6.1 Datenlänge der CAN-Nachricht

Daten in CAN-Nachrichten sind immer Vielfache Byte. Die Länge der CAN-Nachricht entspricht der Anzahl Byte, die erforderlich ist, um alle in diese CAN-Nachricht aktiven „gemappten“ Signalquellen zu versenden.

Sind z. B. nur die ersten 18 Datenbits einer CAN-Nachricht gemappt, beträgt die Datenlänge der CAN-Nachricht `<ByteCount>` 3 Byte.

Grundsätzlich sind „Lücken“ in der CAN-Nachricht möglich und erlaubt. Deren unbestimmte Bits werden mit „0“ gesendet.

Der Parameter `<ByteCount>` dient zur Information, wie lang die aktuell parametrisierte CAN-Nachricht ist. Er kann nur ausgelesen und nicht gesetzt werden.

Will man testweise die Last auf dem CAN-Bus reduzieren, kann man temporär das `<Active>`-Tag einer `<Source>` auf „false“ setzen. Die übrigen Parameter zu dieser Quelle bleiben dabei erhalten, sind aber nicht wirksam. Wenn die Quelle später mit dem Setzen des `<Active>`-Tag auf „true“ wieder aktiviert wird, sind alle zuvor bestimmten Parameter dieser Quelle wieder aktiv. Wenn die inaktive Quelle am Ende einer CAN-Nachricht „gemappt“ ist, wird sich die Anzahl der Datenbyte der CAN-Nachricht entsprechend verkürzen. Ist sie dagegen zwischen weiteren Quellen platziert, wird sich die Datenlänge der CAN-Nachricht nicht ändern, die entsprechenden Bits dieser Quelle werden mit „0“ aufgefüllt.

6.5.7 Beispiel für verschiedene Signalquellen innerhalb einer einzigen CAN-Nachricht

Innerhalb einer einzigen CAN-Nachricht sollen Daten aus verschiedenen Quellen gesendet werden.

Als erstes soll ein Messwert als Float-Wert in die CAN-Nachricht kodiert werden. Als nächstes soll ein Float-Wert einer Mathematikeinheit als 18 bit langer Signed-Integer unter Berücksichtigung von 2 Nachkommastellen an der Bit-Position 33 gesendet werden. Schließlich sollen ab Bit-Position 52 drei Bit mit Schaltzuständen

gesendet werden, die zuvor im gleichen Modul per CAN-Bus empfangen wurden.

Erforderliche Parameter, die der Anwender festlegt. **Fettkursiv** sind die Werte gesetzt, die zwingend erforderlich sind, um obige Bedingungen zu erfüllen:

Für alle Sources gelten der identische Wert für <Identifizier> und <Frameformat>.

Source 1:

Input: ModuleReference = „UUID = 001234“, Signal-Reference = „AnalogIn_Connector1.Signal2“
 Type = **MeasVal**
 DataFormat = **REAL32**
 Factor = 1.0
 Offset = 0.0
 BitSequence = INTEL
 StartBit = **0**
 (SignalLength wird automatisch auf 32 gesetzt aufgrund DataFormat = REAL32)

Source 2:

Input: ModuleReference = „UUID = 001256“, Signal-Reference = „Math_Index2.Signal1“
 Type = **MeasVal**
 DataFormat = **Signed Integer32**
 Factor = **100.0** (verschiebt den Messwert der Signalquelle um 2 „gedachte“ Nachkommastellen)
 Offset = 0.0
 StartBit = **33**
 SignalLength = **18**
 BitSequence = INTEL
 (Nachdem der Float-Wert mit 100 multipliziert und danach in einen Signed-Integer32-Wert konvertiert wurde, werden aus diesem Integer-Wert die Bits 31 bis 8 INTEL-codiert gesendet)

Source 3:

Input: ModuleReference = „“, SignalReference = „CanReceiver_Connector2_Decoder1.Signal1“

Type = **MeasVal**

DataFormat = **Unsigned Integer32**

Factor = 1.0

Offset = 0.0

StartBit = **52**

SignalLength = **3**

BitSequence = INTEL

Daraus ergibt sich diese CAN-Nachricht

Bit 55 (1 bit)	Bit 54 ... 52 (3 bit)	Bit 51 ... 49 (3 bit)	Bit 48 ... 33 (24 bit)	Bit 32 (1 bit)	Bit 31 ... 0 (32 bit)
0 (nicht genutzt)	Signal vom CAN- Dekoder (Source 3)	0 0 0 (nicht genutzt)	Messwert aus Mathe- Einheit (Source 2) als 18-bit-In- teger-Wert	0 (nicht genutzt)	Messwert vom Analogeingang (Source 1) als Float-Wert (REAL32)

Da auf dem CANbus immer nur Pakete in Byte-Größe übertragen werden können, hat diese CAN-Nachricht demnach eine Datenlänge von 7 Byte.

Mit der CAN-Receive-Funktionalität (CAN-Dekoder <CanInChannel>) im MX840B oder eines anderen CAN-Kanals eines MX471B kann diese CAN-Nachricht direkt wieder zu vier verschiedenen Signalen dekodiert werden, indem <DataFormat>, <StartBit>, <BitSequence> und <SignalLength> aus obiger Parametrierung übernommen werden.

6.5.8 Transmission-Type

Wann wird gesendet?

<TransmissionType> legt für eine CAN-Nachricht fest, wann die Nachricht auf dem CAN-Bus versendet werden soll. Hierzu kann man verschiedene Modi wählen. Gesendet wird immer der Wert, der zu diesem Zeitpunkt aktuell vorliegt. Ist noch kein Wert zu einer Quelle bekannt, wird der Wert gesendet, der in Kap. 5.5 parametrierung wurde.

6.5.8.1 Control

Eine CAN-Nachricht wird nur dann versendet, wenn ein Control mit Angabe des Connectors und des Index der CAN-Nachricht abgesetzt wurde. Ein Control kann z. B. per Menü oder Schaltfläche des MX-Assistenten abgesetzt werden.

6.5.8.2 Timer

Der Anwender definiert ein Zeitintervall (in Mikrosekunden). In diesem zeitlichen Abstand werden die CAN-Nachrichten selbstständig gesendet. Wird „Interval = 0“ gesetzt, wird keine CAN-Nachricht versendet.

Der Versand einer CAN-Nachricht kann zusätzlich per Control (siehe Kap. 5.8.1) ausgelöst werden.

Die Daten in der CAN-Nachricht entsprechen dem Momentanwert und sind nicht synchronisiert mit der Datenquelle.

Im MX471B realisieren wir die Timer-Auflösung durch Abzählen der isochronen Ereignisse (1200 Hz). Die Peri-

odendauer der CAN-Nachrichten entspricht somit Vielfachen von 1/1200 Sekunden.

Im MX840B erfolgt die Auflösung in Schritten von 1 ms, wobei die kürzeste Periodendauer 10 ms je CAN-Nachricht ist.

6.5.8.3 SourceChange

Ändert sich der Wert der Quelle zu dem letzten gesendeten Wert um den Betrag $\langle \text{Delta} \rangle$, wird die CAN-Nachricht gesendet. $\langle \text{Delta} \rangle$ wird in den Parametern der jeweiligen Quelle festgelegt. Wenn mehrere Messwerte in einer CAN-Nachricht „gemappt“ sind, wird bei der Änderung einer Quelle die gesamte CAN-Nachricht gesendet.

Der Versand einer CAN-Nachricht kann zusätzlich per Control (siehe Kap. 5.8.1) ausgelöst werden.

6.5.8.4 IsoEvent

Dies entspricht dem Verhalten, wie es im MX471B bis einschließlich Firmware-Version 4.4 implementiert war: Die CAN-Nachricht wird dann verschickt, wenn eine der „gemappten“ Signalquellen (per Firewire) isochron empfangen wurde. Zusätzlich kann mittels $\langle \text{Divisor} \rangle$ festgelegt werden, dass nur alle n empfangenen isochronen Ereignisse gesendet wird. Damit kann die Auslastung auf dem CAN-Bus verringert werden, wenn z. B. Messwerte mit niedriger Dynamik (z. B. Temperatur) gesendet werden sollen.

Der Versand einer CAN-Nachricht kann zusätzlich per Control (siehe Kap. 5.8.1) ausgelöst werden.

6.5.9 Einschränkungen für MX840B

MX840B ist in der Lage, Messwerte, die innerhalb des Moduls gewonnen wurden, per CAN-Nachricht zu verschicken. Die Einschränkungen sind hierbei aus Performance-Gründen deutlich. CAN-Mapping ist nicht möglich.

Um Kompatibilität zu den neuen XML-Bäumen im MX471B zu erhalten, wird auch MX840B ausschließlich die neuen XML-Bäume nutzen. Damit können beide Modultypen mit der gleichen Software-Schnittstelle parametrisiert werden.

Für MX840B gilt dabei weiterhin, dass nur maximal 10 verschiedene CAN-Nachrichten versendet werden können und dass jede CAN-Nachricht nur 1 Source-Quelle haben kann, die einem analogen Messwert innerhalb des Moduls entspricht.

Diese Parameter können bei MX840B daher nicht frei gewählt werden und sind fix wie angegeben:

Type = MeasVal
DataFormat = REAL32
Factor = 1.0
Offset = 0.0
StartBit = 0 bzw. 7 (je nach „BitSequence“)
SignalLength = 32
BitSequence = INTEL oder MOTOROLA
CanMessage-TransmissionType = „Control“ oder „Timer“

Eine empfangene CAN-Nachricht kann nicht per CAN weitergesendet werden. Auch muss die Signalquelle innerhalb des Moduls entnommen werden, also:

ModuleReference (leer bzw. „UUID=[eigene UUID]“),
SignalReference = „AnalogIn_Connector[2...8].Signal1“

Die Parameter aus <ValueOnError> werden unterstützt.

Für den Timer-gesteuerten Versand gilt, dass ein Messwert nur alle 10 ms oder seltener versendet werden kann. Zu beachten ist, dass der Parameter <Period> bis einschließlich Firmware-Version 4.4 in Millisekunden angegeben wurde. Um diesen Parameter künftig identisch zum MX471B zu nutzen, muss <Period> künftig auch im MX840B in Microsekunden angegeben werden, wobei die Auflösung 1000 µs beträgt.

6.6 Zählweisen der Datenbit gemäß Vector-DBC-Format

6.6.1 Zählweisen, in QuantumX / SomatXR-Parametrierung genutzt

Die Daten für die Messwerte können in unterschiedlichen Zählweisen interpretiert werden.

Die Formate „INTEL Standard“, „MOTOROLA Forward“ und „MOTOROLA Backward“ nutzen folgende „Sägezahn“-Zählweise der Datenbit:

Byte 0 (aus CAN-Controller)							
Bit 7	6	5	4	3	2	1	0

Byte 1							
15	14	13	12	11	10	9	8

Die Formate „INTEL Sequential“ und „MOTOROLA Sequential“ nutzen folgende sequentielle Zählweise der Datenbit. Sie entspricht der Reihenfolge, wie die Bits vom CAN-Bus empfangen werden:

Byte 0 (aus CAN-Controller)							
Bit 0	1	2	3	4	5	6	7

Byte 1							
0	1	2	3	4	5	6	7

6.6.1.1 Format INTEL-Standard

Für Signale im „INTEL Standard“-Format wird als „Startbit“ die Position des Least-Significant-Bit („lsb“) des Messwerts angegeben und dann ab dem Startbit nach links mit aufsteigender Bit-Wertigkeit gezählt, wie im Beispiel unten gezeigt.

Für den Datenempfang im CAN-Dekoder gilt: Ist der Datentyp vorzeichenbehaftet, wird dies immer im „msb“ angenommen und beim Rechtsschieben in den Zwischenwert mit 1 aufgefüllt. Diese Dekodierung wird sowohl für den Messwert als auch die Mode-Information gleichermaßen verwendet, wenn im Element „BitSequence“ bzw. „ModeBitSequence“ der Wert „1“ steht.

Beispiel „INTEL Standard“-Format, Startbit = 9, Länge = 12:

Bit-Nr. innerhalb des Daten-Byte								0	1	2	3	4	5	6	7	
7	6	5	4	3	2	1	0									
7	6	5	4	3	2	1	0	Daten-In- dex	0	1	2	3	4	5	6	7
15	14	13	12	11	10	9 Startbit/ lsb	8									
23	22	21	20 msb ←	19 ←	18 ←	17 ←	16 ←									
31	30	29	28	27	26	25	24									
39	38	37	36	35	34	33	32									
47	46	45	44	43	42	41	40									
55	54	53	52	51	50	49	48									
63	62	61	60	59	58	57	56									

6.6.1.2 Format MOTOROLA Forward MSB

Für Signale im CANdb-internen „MOTOROLA Forward MSB“-Format wird als „Startbit“ die Position des Most-Significant-Bit („msb“) des Messwerts angegeben und mit nach rechts absteigender Bit-Wertigkeit gezählt, wie im Beispiel unten gezeigt.

Für den Datenempfang im CAN-Dekoder gilt: Ist der Datentyp vorzeichenbehaftet, wird dies immer im „msb“ angenommen und beim Empfang der CAN-Nachrichten beim Rechtsschieben in den Zwischenwert mit 1 aufgefüllt. Diese Dekodierung wird sowohl für den Messwert als auch die Mode-Information gleichermaßen verwendet, wenn im Element „BitSequence“ bzw. „ModeBitSequence“ der Wert „0“ steht.

Beispiel CANdb-internes „MOTOROLA Forward MSB“-Format, Startbit = 13, Länge = 12:

Bit-Nr. innerhalb des Daten-Byte								
7	6	5	4	3	2	1	0	
7	6	5	4	3	2	1	0	0
15	14	13 Startbit/ msb	12 ←	11 ←	10 ←	9 ←	8 ←	1
23 ←	22 ←	21 ←	20 ←	19 ←	18 ← lsb	17	16	2
31	30	29	28	27	26	25	24	3
39	38	37	36	35	34	33	32	4
47	46	45	44	43	42	41	40	5
55	54	53	52	51	50	49	48	6
63	62	61	60	59	58	57	56	7

Daten-In-
dex

6.6.2 Weitere Zählweisen, nicht in QuantumX / SomatXR-Parametrierung genutzt

Um weitere Zählweisen in ein Format überführen zu können, das zur Parametrierung von MX genutzt wird, dienen nachfolgende Tabellen zur Information.

6.6.2.1 Format MOTOROLA Forward LSB

Beispiel CANdb-internes „MOTOROLA Forward LSB“-Format, Startbit = 18, Länge = 12:

Bit-Nr. innerhalb des Daten-Byte								
7	6	5	4	3	2	1	0	
7	6	5	4	3	2	1	0	0
15	14	13 msb ←	12 ←	11 ←	10 ←	9 ←	8 ←	1
23 ←	22 ←	21 ←	20 ←	19 ←	18 Startbit/ lsb	17	16	2
31	30	29	28	27	26	25	24	3
39	38	37	36	35	34	33	32	4
47	46	45	44	43	42	41	40	5
55	54	53	52	51	50	49	48	6
63	62	61	60	59	58	57	56	7

Daten-Index

6.6.2.2 Format MOTOROLA Backward

Bei diesem Format wird ab dem letzten zu versendenden Bit gezählt. Somit ist das Startbit abhängig von der Anzahl der zu versendenden Byte.

Beispiel CANdb-internes „MOTOROLA Backward“-Format, Datenlänge = 8, Startbit = **42**, Länge = 12:

Bit-Nr. innerhalb des Daten-Byte								
7	6	5	4	3	2	1	0	
63	62	61	60	59	58	57	56	0
55	54	53 msb ←	52 ←	51 ←	50 ←	49 ←	48 ←	1
47 ←	46 ←	45	44 ←	43 ←	42 Startbit/ lsb	41	40	2
39	38	37	36	35	34	33	32	3
31	30	29	28	27	26	25	24	4
23	22	21	20	19	18	17	16	5
15	14	13	12	11	10	9	8	6
7	6	5	4	3	2	1	0	7

Daten-In-
dex

Die gleiche Nutzinformation, gesendet in 3 Daten-Byte:

CANdb-internes „MOTOROLA Backward“-Format,
Datenlänge = 3, Startbit = **2**, Länge = 12:

Bit-Nr. innerhalb des Daten-Byte								
7	6	5	4	3	2	1	0	
23	22	21	20	19	18	17	16	0
15	14	13 msb ←	12 ←	11 ←	10 ←	9 ←	8 ←	1
7 ←	6 ←	5	4 ←	3 ←	2 Startbit/ lsb	1	0	2

Daten-In-
dex

6.6.2.3 Format INTEL Sequential

Beispiel CANdb-internes „INTEL Sequential“-Format,
 Startbit = **14**, Länge = 12:

Bit-Nr. innerhalb des Daten-Byte									
0	1	2	3	4	5	6	7		
0	1	2	3	4	5	6	7	0	
8	9	10	11	12	13	14 Startbit/ lsb	15	1	
←		←	←	←	←			Daten-In- dex	
16	17	18	19 msb ←	20 ←	21 ←	22 ←	23 ←		2
24	25	26	27	28	29	30	31		3
32	33	34	35	36	37	38	39		4
40	41	42	43	44	45	46	47		5
48	49	50	51	52	53	54	55		6
56	57	58	59	60	61	62	63		7

6.6.2.4 Format MOTOROLA Sequential

Beispiel CANdb-internes „MOTOROLA Sequential“-Format, Startbit = 10, Länge = 12:

Bit-Nr. innerhalb des Daten-Byte								
0	1	2	3	4	5	6	7	
0	1	2	3	4	5	6	7	0
8	9	10 msb ←	11 ←	12 ←	13 ←	14 ←	15 ←	1
16 ←	17 ←	18	19 ←	20 ←	21 Startbit/ lsb	22	23	2
24	25	26	27	28	29	30	31	3
32	33	34	35	36	37	38	39	4
40	41	42	43	44	45	46	47	5
48	49	50	51	52	53	54	55	6
56	57	58	59	60	61	62	63	7

Daten-In-
dex

7 QuantumX / SomatXR und CCP / XCP-on-CAN

7.1 Einführung in CCP und XCP

Moderne Steuergeräte (Electronic Control Units, ECUs) in Fahrzeugen kommunizieren untereinander oder mit Abstimmungs- und Analysewerkzeugen über verschiedene Protokolle. Das "Universal Measurement and Calibration Protocol" oder kurz XCP ist ein modernes Protokoll, das hauptsächlich zur Parametrierung / Kalibrierung oder Feineinstellung von Parametern und Datenfeldern in Steuergeräten dient. Außerdem kann das XCP zur Erfassung der Signalinformationen von Steuergeräten genutzt werden, die normalerweise nicht Bestandteil der Standardkommunikation zwischen Steuergeräten (ECU-2-ECU) sind – beispielsweise von internen Signalisierungs- oder Sensorinformationen des Steuergeräts.

Die Hardware-Verbindung zwischen Messgerät und Steuergerät verläuft weiterhin über den gleichen Datenbus, doch das "Mess- und Kalibrierprotokoll" wird von unterschiedlichen Geräten aktiviert, beispielsweise Datenrekordern oder Geräten für die Motoreinstellung. XCP wurde von einem Arbeitskreis der ASAM (Association for Standardization of Automation and Measuring Systems) als weltweiter Standard festgelegt.

Das "X" steht dabei für die variable und austauschbare Transportschicht, die als CANbus (XCP-on-CAN), Flex-Ray (XCP-on-FlexRay) oder Ethernet (XCP-on-Ethernet) ausgeführt sein kann. XCP ist das Nachfolgeprotokoll von CCP (CAN Calibration Protocol), das mit der konzeptionellen Idee entwickelt worden war, einen Lese- und Schreibzugriff auf interne Daten des Steuergeräts nur über CAN zu gewähren.

XCP bietet die Möglichkeit, Messwerte “ereignissynchron” zu Prozessen in Steuergeräten zu erfassen. Dies sichert die Konsistenz der Daten untereinander.

7.2 MX471B und CAN / XCP-on-CAN

Der MX471B bietet 4 einzeln konfigurierbare CAN-Schnittstellen, die für den Betrieb mit CCP oder XCP-on-CAN und damit für das Lesen der Signalwerte von Steuergeräten parametrierbar werden können. In diesem Modus arbeitet MX471B als Logger. Nachdem die Kommunikation mit dem Steuergerät aufgebaut wurde, sendet das Steuergerät Daten in einer zyklischen Periode.

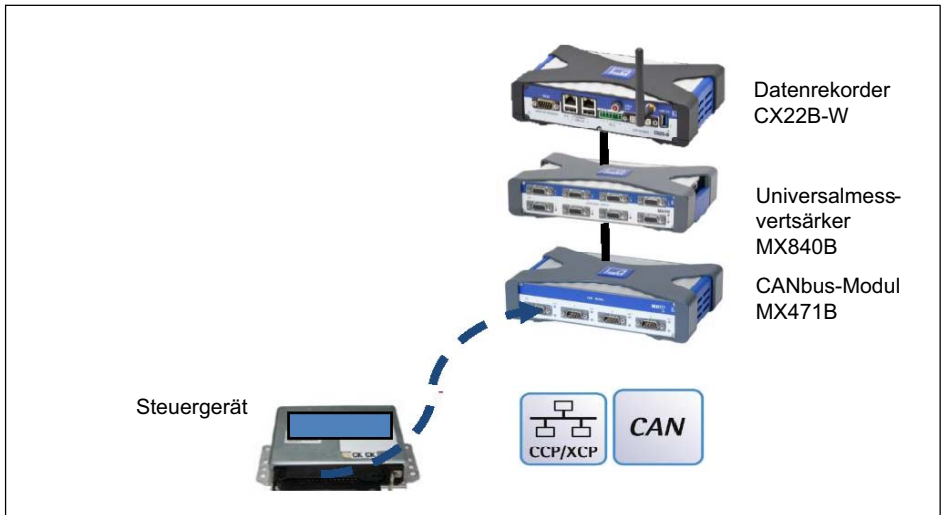


Abb. 7.1 Beispielkonfiguration von MX471B und Kommunikation über CCP / XCP-on-CAN

Voraussetzungen

- Das Steuergerät unterstützt die folgenden Protokolle:
- CCP Version 2.1 oder höher
- XCP-on-CAN Version 1.1 oder höher
- Die Parameterbeschreibungsdatei *.a2l vom Hersteller des Steuergeräts ist verfügbar.
- Wenn das Steuergerät durch ein Schlüsselaustauschverfahren ("Seed & Key") gesperrt ist, muss die zugehörige *.skb-Datei verfügbar sein.
- CANape Version 10.0 oder höher ist installiert (Betrieb mit früheren Versionen eventuell möglich, wurde aber nicht getestet).
- MX471B mit neuester Firmware ist verfügbar.
- Datenrekorder CX22B-W oder ein PC mit der aktuellen Version des MX-Assistenten ist verfügbar.

Hardware Setup

- Verbinden Sie eine der Schnittstellen des MX471B mit dem CAN-Netzwerk.
- Verbinden Sie die Ethernet-Schnittstellen des MX471B mit einem Laptop.

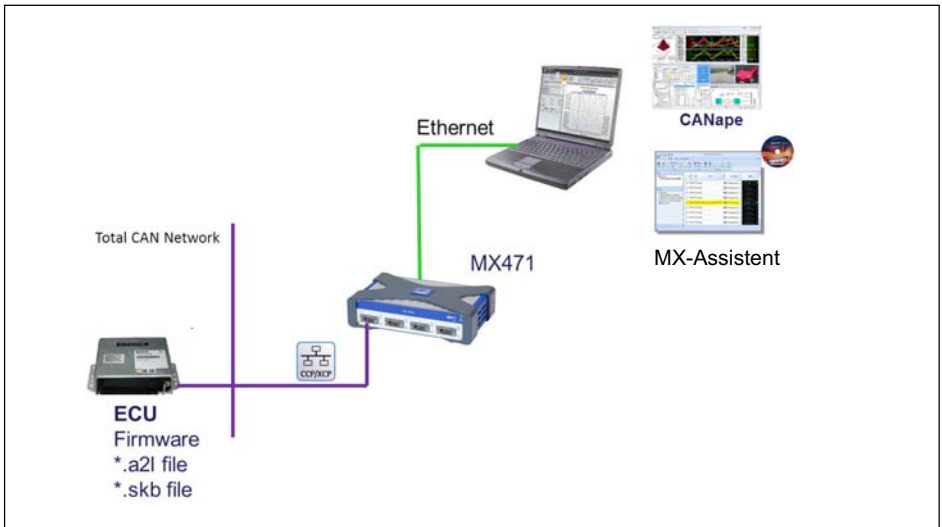


Abb. 7.2 Hardware Setup

7.3 Initialisierung per XML

Die Initialisierung erfolgt über XML-Parameter in der Datei „AddConnParam.xml“, ab XPath:

```
...<CanBus><ECU>.
```

In das Tag <DBC> muss der Inhalt einer Vector-DBC-Datei als String geschrieben werden, sofern eine

„seed and key binary“-Datei verwendet werden soll, deren Inhalt als String ins Tag <SKB>. Mit dem Schreiben dieser Daten wird der CAN-Bus für das CCP/XCP-Protokoll initialisiert.

Wurde das XML-Tag <Active> auf „1“ gesetzt, wird die Verarbeitung direkt gestartet, ansonsten startet die Verarbeitung erst mit dem Schreiben eines Controls.

Nach dem Neustart des Moduls werden die Daten automatisch aktiviert, die zuvor in die XML-Tags eingetragen wurden.

Die PC-Software „MXAssistant“ bietet im Reiter „Kanal“ den Dialog „CAN-Bus“ - „Einstellungen...“. Im folgenden Dialog „CCP / XCP über CAN“ können die erforderlichen Parameter gesetzt werden.

7.4 Start und Stopp per Control „CANECU“

Die Steuerung per Control ist erst mit Firmware ab Versionen 4.8 möglich.

Wenn das Tag <Active> auf „0“ gesetzt wurde, startet die Verarbeitung erst mit dem Schreiben des Controls „CANECU“. Voraussetzung ist, dass die die Protokoll-Verarbeitung durch Setzen der Parameter <DBC> und evtl. <SKB> initialisiert wurde.

Anhand dieses Controls kann die Verarbeitung jederzeit gestoppt und wieder gestartet werden. Das Control „CANECU“ nutzt diese Parameter zum Starten und Stoppen:

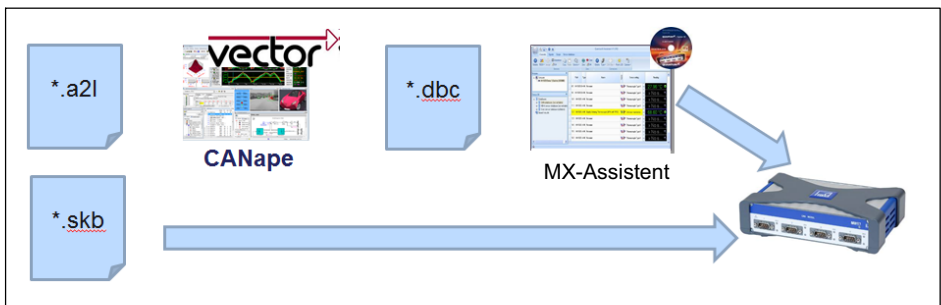
Connector	Nummer des CAN-Bus-Anschlusses („1...4“, „0“: alle)
Index	Index der ECU, gegenwärtig immer „1“.
Mode	Auswahl der Funktionalität: „1“: Start/Stop
Value	Auswahl zwischen Start („1“) und Stopp („0“)

Die PC-Software „catman“ bildet die Steuerung per Control in einem entsprechenden Dialog ab.

7.5 Allgemeiner Arbeitsablauf

Starten Sie CANape von Vector Informatik, und lesen Sie die *.a2l-Datei ein. Erstellen Sie nun eine Messkonfiguration mit allen relevanten Signalen, die Sie mit MX471B erfassen möchten. Wandeln Sie die Messkonfiguration anschließend in eine *.dbc-Datei um.

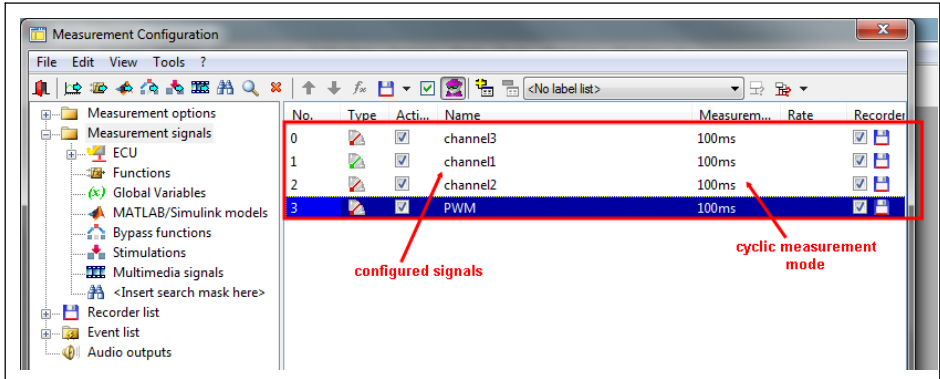
Starten Sie den MX-Assistenten, und lesen Sie die *.dbc- und *.skb-Dateien ein (sofern zutreffend). Laden Sie die Dateien danach in das Gerät. Die CCP- oder XCP-on-CAN-Kommunikation zwischen dem Steuergerät und MX471B wird daraufhin sofort gestartet. Dieser Dienst kann auch über ein in catman ausgeführtes Skript (in Arbeit) aktiviert und deaktiviert werden.



Schritt 1 – Erstellen der Messkonfiguration mit CANape

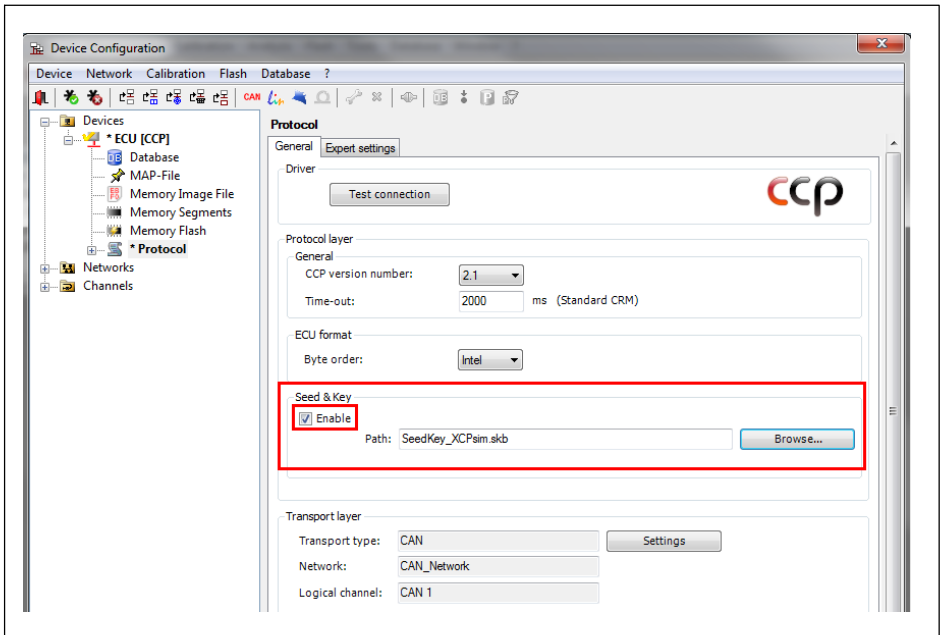
- ·Erstellen Sie ein neues CANape-Projekt.
- ·Erstellen Sie ein neues CCP- oder XCP-Gerät, je nachdem, welches Protokoll im Steuergerät implementiert ist.
- ·Erstellen Sie eine Messkonfiguration mit allen Signalen, die Sie mit MX471B aufzeichnen möchten. Beachten Sie, dass es sich um einen zyklischen

Messmodus handelt. Der Polling-Modus wird nicht unterstützt.

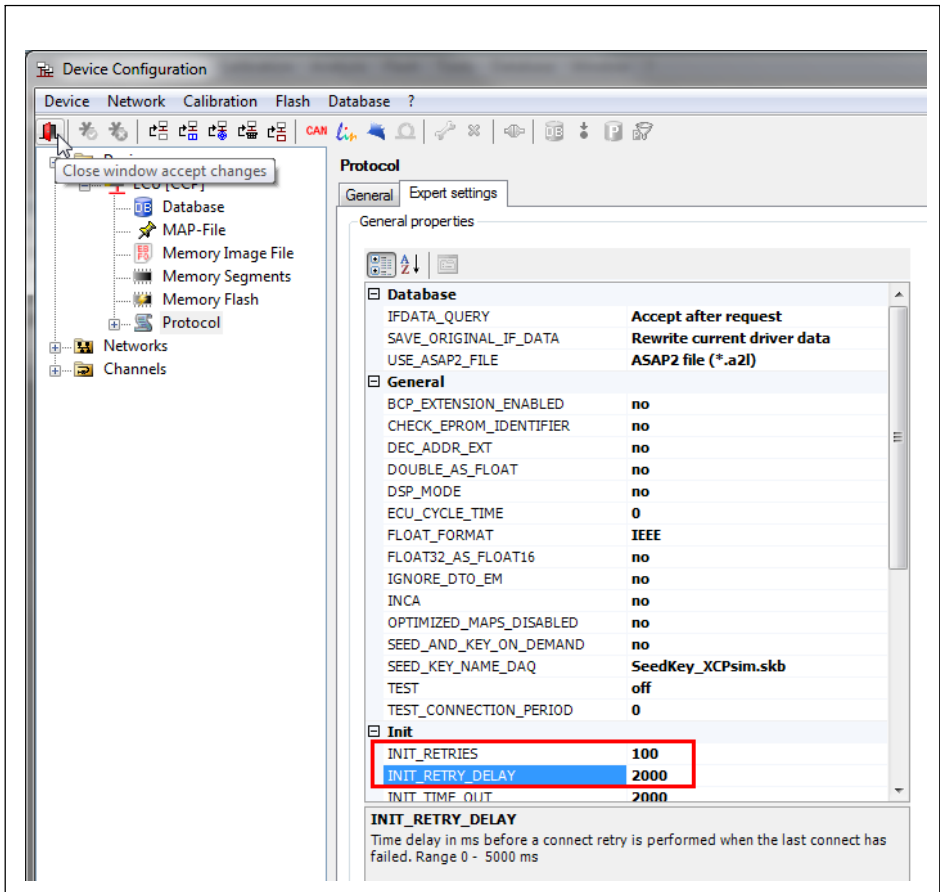


Schritt 2 – Konfigurieren der Schlüsselaustausch-Option “Seed & Key” und Anpassen der “Init”-Einstellungen mit CANape

- Wenn das Steuergerät die Schlüsselaustausch-Option “Seed & Key” nicht unterstützt, müssen Sie sie in den “Protocol Settings” (Protokolleinstellungen) unter “Device Configuration” (Gerätekonfiguration) deaktivieren.
- Wenn das Steuergerät “Seed & Key” unterstützt, müssen Sie die *.skb-Datei auswählen.

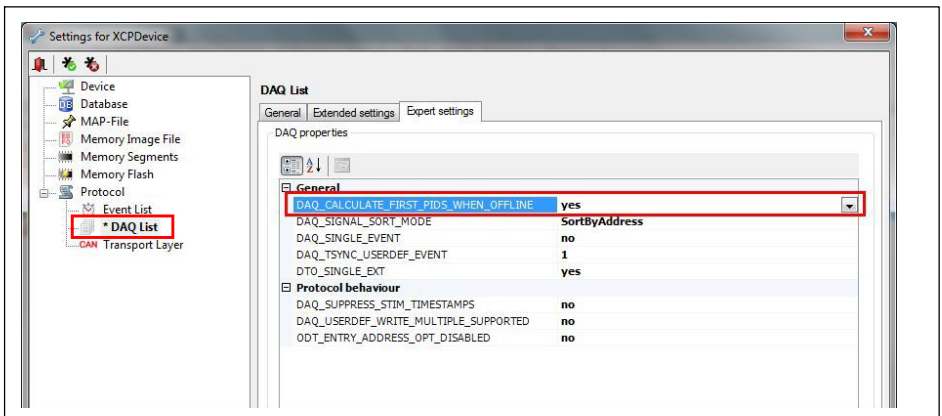


- Wechseln Sie zur Registerkarte “Expert settings” (Erweiterte Einstellungen). Ändern Sie die Einstellungen in INIT_RETRIES = 100 und INIT_RETRY_DELAY = 2000 (empfohlene Werte). MX471 B versucht dann 100 Mal, die Kommunikation mit dem Steuergerät aufzubauen, und wiederholt die Versuche alle ms. Dies ist wichtig, wenn das Steuergerät nach dem MX471B eingeschaltet wird.

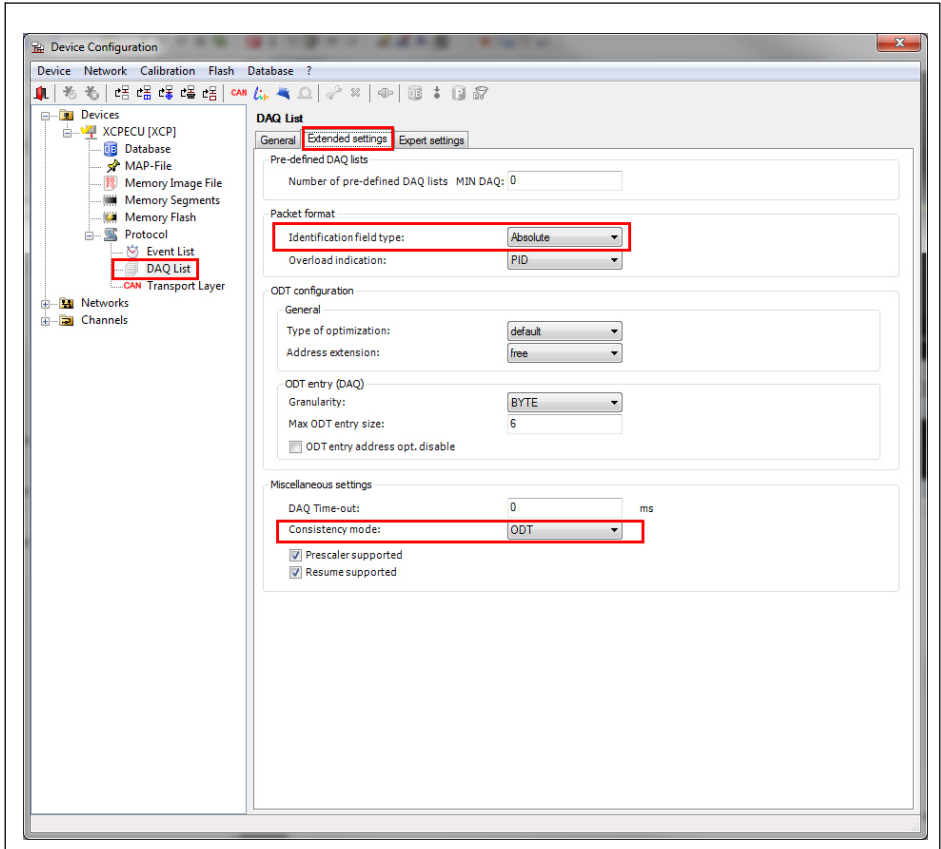


Schritt 2b – Nur für XCP-on-CAN – Anpassen der Protokolleinstellungen in der Gerätekonfiguration (für CCP nicht notwendig)

- Setzen Sie den Parameter “DAQ_CALCULATE_FIRST_PIDS_WHEN_OFFLINE” auf “yes” (ja).

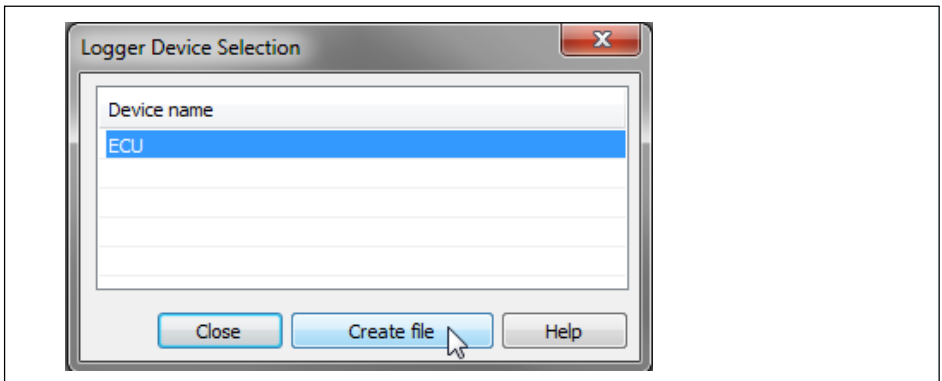


- ·Setzen Sie “Consistency mode” (Konsistenzmodus) auf “ODT” und “Identification field type” (Typ des Identifizierungsfelds) auf “Absolute” (Absolut).



Schritt 3 – Exportieren der Messkonfiguration in eine *.dbc-Datei mit CANape

- Wählen Sie in “Logger Configuration” (Logger-Konfiguration) (aufgerufen über “Tools” (Extras) -> “Logger configuration” (Logger-Konfiguration)) die Option “ECU” (Steuergerät), und klicken Sie auf die Schaltfläche “Create file” (Datei erstellen).

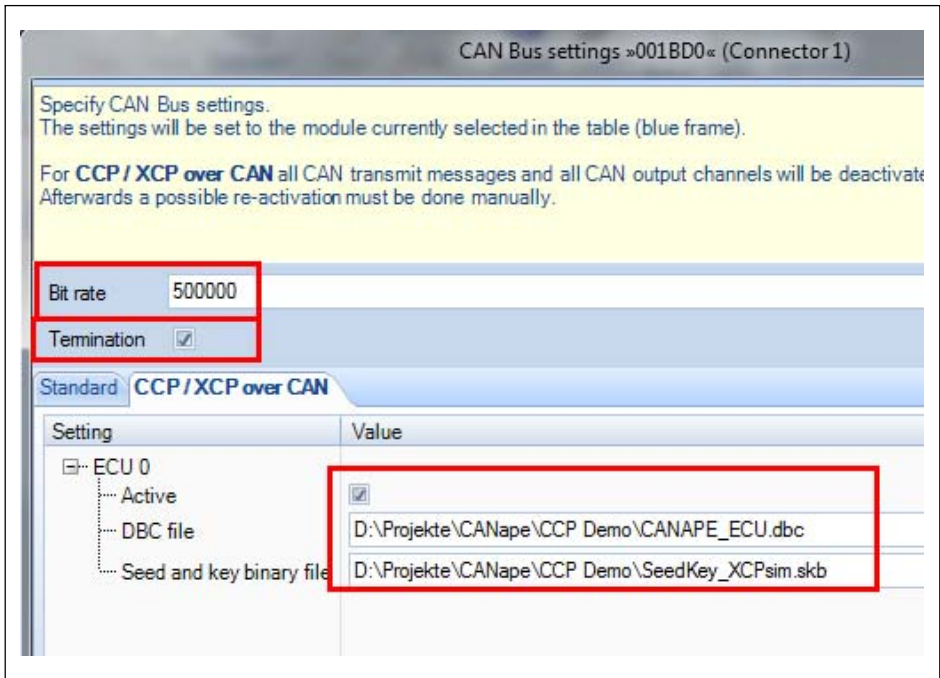


- => Die *.dbc-Datei wird in Ihrem CANape-Projektverzeichnis erstellt.

Schritt 4 – Konfigurieren von MX471B mit der *.dbc-Datei und MX-Assistent:

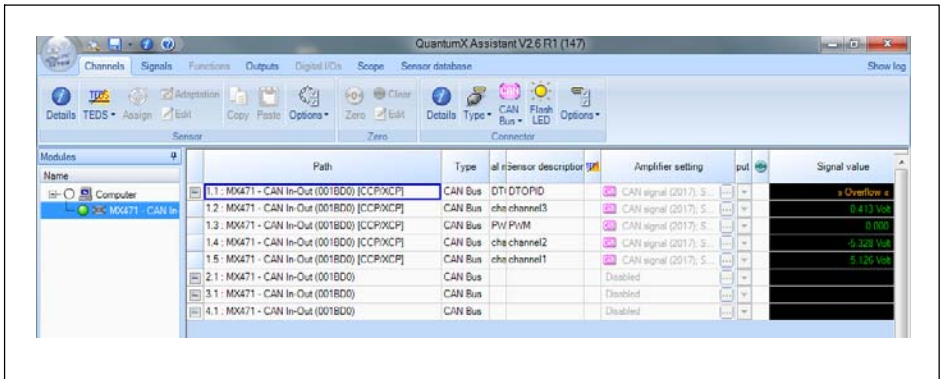
- Öffnen Sie den MX-Assistent, und verbinden Sie den Assistenten mit Ihrem MX471B.
- Wählen Sie “Channel 1” (Kanal 1) des MX471B, und öffnen Sie “CAN Bus settings” (CAN-Bus-Einstellungen).
- Geben Sie die Bitrate Ihres CAN-Netzwerks ein.
- Aktivieren Sie bei Bedarf die Option “Termination” (Abschlusswiderstand).
- Wählen Sie die in Schritt 3 erstellte *.dbc-Datei aus.

- Wenn das Steuergerät mit einem Schlüsselaustauschverfahren ("Seed & Key") gesperrt ist, müssen Sie die *skb-Datei auswählen.
- Wenn das Steuergerät nicht gesperrt ist, lassen Sie das Feld leer.
- Klicken Sie auf die Schaltfläche "OK".



Wird das Häkchen bei Active gesetzt, startet die Verarbeitung sofort. Ist das Häkchen nicht gesetzt, wird auf den Start per Control gewartet.

Ergebnis: Alle konfigurierten Signale werden nun im CAN-Bus 1 angezeigt.



Schritt 5 – Nun können Sie Catman öffnen und die Messung starten.

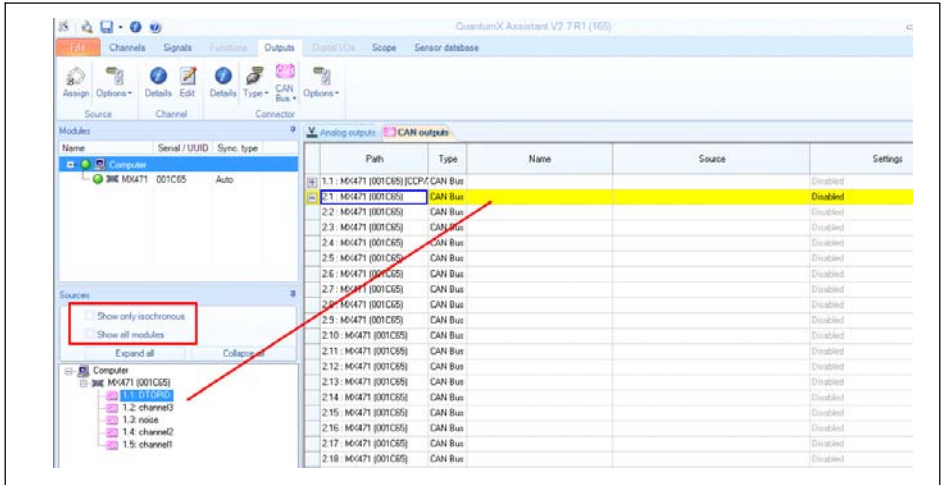
Konfiguration eines Gateway-Modus

MX471B kann auch als Gateway genutzt werden, das XCP-on-CAN- oder CCP-Signale an das CAN überträgt. In diesem Modus werden die XCP- oder CCP-Nachrichten an einer Schnittstelle des MX471B empfangen und an einer anderen Schnittstelle als CAN-Standardnachrichten weitergesendet.

Konfigurationsschritte im MX-Assistenten

5. Wechseln Sie zur Registerkarte "Outputs" (Ausgänge).
6. Deaktivieren Sie das Kontrollkästchen "Show only isochronous" (Nur isochrone anzeigen).

- Ziehen Sie Signale per Drag & Drop aus dem Bereich "Sources" (Quellen) auf einen "CAN Output" (CAN-Ausgang).



- Passen Sie die CAN-IDs so an, dass jedes Signal seine eigene eindeutige ID hat, in diesem Beispiel 385, 386, 387, 388. (Eine automatische Zuweisung fortlaufender IDs wird über "CAN bus settings -> Assign message IDs" (CAN-Bus-Einstellungen -> Nachrichten-IDs zuweisen) konfiguriert.)

8 Glossar

A2L

Dateierweiterung der Gerätebeschreibungsdatei eines Steuergeräts. Wurde von der ASAM standardisiert.

ASAM

“Association for Standardisation of Automation and Measuring Systems” (Vereinigung zur Standardisierung von Automatisierungs- und Messsystemen)

CCP

“CAN Calibration Protocol” (CAN-Kalibrierungsprotokoll). Wurde von der ASAM standardisiert.

DBC

“Data Base CAN” (Datenbasis-CAN): das Dateiformat für die CAN-Kommunikation

ECU

“Electronic Control Unit” (Steuergerät)

Seed & Key

Schlüsselaustauschverfahren, mit dem das Steuergerät gegen unbefugten Zugriff gesperrt werden kann.

XCP

“Universal Calibration Protocol” (Universal-Kalibrierungsprotokoll). Kann in verschiedenen Netzwerken verwendet werden: CAN, FlexRay, Ethernet. Wurde von der ASAM standardisiert.

HBM Test and Measurement

Tel. +49 6151 803-0

Fax +49 6151 803-9100

info@hbm.com

measure and predict with confidence



A4461-2.0 7-2002.4461 HBM: public

www.hbm.com