

# X20(c)AO2438

Data sheet  
1.50 (January 2025)



## **Publishing information**

B&R Industrial Automation GmbH

B&R Strasse 1

5142 Eggelsberg

Austria

Telephone: +43 7748 6586-0

Fax: +43 7748 6586-26

[office@br-automation.com](mailto:office@br-automation.com)

## **Disclaimer**

All information in this document is current as of its creation. The contents of this document are subject to change without notice. B&R Industrial Automation GmbH assumes unlimited liability in particular for technical or editorial errors in this document only (i) in the event of gross negligence or (ii) for culpably inflicted personal injury. Beyond that, liability is excluded to the extent permitted by law. Liability in cases in which the law stipulates mandatory unlimited liability (such as product liability) remains unaffected. Liability for indirect damage, consequential damage, business interruption, loss of profit or loss of information and data is excluded, in particular for damage that is directly or indirectly attributable to the delivery, performance and use of this material.

B&R Industrial Automation GmbH notes that the software and hardware designations and brand names of the respective companies used in this document are subject to general trademark, brand or patent protection.

Hardware and software from third-party suppliers referenced in this document is subject exclusively to the respective terms of use of these third-party providers. B&R Industrial Automation GmbH assumes no liability in this regard. Any recommendations made by B&R Industrial Automation GmbH are not contractual content, but merely non-binding information for which no liability is assumed. When using hardware and software from third-party suppliers, the relevant user documentation of these third-party suppliers must additionally be consulted and, in particular, the safety guidelines and technical specifications contained therein must be observed. The compatibility of the products from B&R Industrial Automation GmbH described in this document with hardware and software from third-party suppliers is not contractual content unless this has been separately agreed in individual cases; in this respect, warranty for such compatibility is excluded in any case, and it is the sole responsibility of the customer to verify this compatibility in advance.

## **Version history**

B&R makes every effort to keep documents as current as possible. The most current versions are available for download on the B&R website ([www.br-automation.com](http://www.br-automation.com)).

# 1 General information

## 1.1 Other applicable documents

For additional and supplementary information, see the following documents.

### Other applicable documents

Document name	Title
MAX20	<a href="#">X20 System user's manual</a>

## 1.2 Coated modules

Coated modules are X20 modules with a protective coating for the electronics component. This coating protects X20c modules from condensation and corrosive gases.

The modules' electronics are fully compatible with the corresponding X20 modules.



For simplification purposes, only images and module IDs of uncoated modules are used in this data sheet.

The coating has been certified according to the following standards:

- Condensation: BMW GS 95011-4, 2x 1 cycle
- Corrosive gas: EN 60068-2-60, method 4, exposure 21 days



## 1.3 Order data


Order number	Short description	Figure
	<b>Analog outputs</b>	
X20AO2438	X20 analog output module, 2 outputs, 4 to 20 mA / 0 to 20 mA or 0 to 24 mA, 16-bit converter resolution, single-channel galvanically isolated, supports the HART protocol, NetTime function	
X20cAO2438	X20 analog output module coated, 2 outputs, 4 to 20 mA / 0 to 20 mA or 0 to 24 mA, 16-bit converter resolution, single-channel galvanically isolated, supports the HART protocol, NetTime function	
	<b>Required accessories</b>	
	<b>Bus modules</b>	
X20BM11	X20 bus module, 24 VDC keyed, internal I/O power supply connected through	
X20BM15	X20 bus module, with node number switch, 24 VDC keyed, internal I/O power supply connected through	
X20cBM11	X20 bus module, coated, 24 VDC keyed, internal I/O power supply connected through	
	<b>Terminal blocks</b>	
X20TB12	X20 terminal block, 12-pin, 24 VDC keyed	

Table 1: X20AO2438, X20cAO2438 - Order data

## 1.4 Module description

The module is equipped with 2 current outputs with 16-bit digital converter resolution. It supports the HART communication standard for data transfer, parameter configuration and diagnostics.

The 2 channels are electrically isolated from each other. The user can select between the 3 output ranges 4 to 20 mA, 0 to 20 mA and 0 to 24 mA.

Functions:

- [Analog outputs](#)
- [HART](#)
- [OSP mode](#)
- [NetTime Technology](#)
- [Flatstream communication](#)

### Analog outputs

The module is equipped with analog outputs with a configurable current signal.

### HART

Highway Addressable Remote Transducer (HART) is a protocol for communicating with intelligent field devices. The procedure was designed to make more efficient use of infrastructures for the transfer of analog signals.

In exceptional cases, error messages may occur when connecting specific DTM devices. For details, see ["Limitations" on page 13](#).

### OSP mode

In mode "OSP" (Operator Set Predefined), the user defines an analog value or digital pattern. This OSP value is output as soon as the communication between the module and master is aborted.

### NetTime timestamp of the HART image

For many applications, not only the HART values are important, but also the exact time of receipt. For this purpose, the module has a NetTime timestamp function that provides the reception time with a timestamp with microsecond accuracy.

### Flatstream communication

"Flatstream" was designed for X2X and POWERLINK networks and allows data transfer to be adapted to individual demands. This allows data to be transferred more efficiently than with standard cyclic polling.

## 2 Technical description

### 2.1 Technical data

Order number	X20AO2438	X20cAO2438
Short description		
I/O module	2 analog outputs 4 to 20 mA, 0 to 20 mA or 0 to 24 mA	
General information		
B&R ID code	0xB3AA	0xE211
Status indicators	I/O function per channel, operating state, module status, HART	
Diagnostics		
Module run/error	Yes, using LED status indicator and software	
Outputs	Yes, using LED status indicator and software	
HART link	Yes, using LED status indicator and software	
HART error	Yes, using LED status indicator and software	
Power consumption		
Bus	0.05 W	
Internal I/O	1.65 W	
Additional power dissipation caused by actuators (resistive) [W]	-	
Certifications		
CE	Yes	
UKCA	Yes	
ATEX	Zone 2, II 3G Ex nA nC IIA T5 Gc IP20, Ta (see X20 user's manual) FTZÜ 09 ATEX 0083X	
UL	cULus E115267 Industrial control equipment	
HazLoc	cCSAus 244665 Process control equipment for hazardous locations Class I, Division 2, Groups ABCD, T5	
DNV	Temperature: <b>B</b> (0 to 55°C) Humidity: <b>B</b> (up to 100%) Vibration: <b>B</b> (4 g) EMC: <b>B</b> (bridge and open deck)	
CCS	Yes	-
LR	ENV1	
KR	Yes	
ABS	Yes	
BV	<b>EC33B</b> Temperature: 5 - 55°C Vibration: 4 g EMC: Bridge and open deck	
KC	Yes	-
Analog outputs		
Output	4 to 20 mA, 0 to 20 mA or 0 to 24 mA configurable using software	
Digital converter resolution	16-bit	
Settling time on output change over entire range	2 ms to 20 s, configurable using software	
Data output rate		
With HART	210 ms (default)	
Analog	1 ms without ramp	
Max. error <sup>1)</sup>		
Gain		
4 to 20 mA	±0.025% <sup>2)</sup>	
0 to 20 mA	±0.022% <sup>2)</sup>	
0 to 24 mA	±0.02% <sup>2)</sup>	
Offset		
4 to 20 mA	±0.025% <sup>3)</sup>	
0 to 20 mA	±0.022% <sup>3)</sup>	
0 to 24 mA	±0.02% <sup>3)</sup>	
Output protection	Short-circuit proof, overvoltage protection (up to 30 VDC)	
Open-circuit detection	Yes, using hardware and software	
Data format	INT	

Table 2: X20AO2438, X20cAO2438 - Technical data

## Technical description


Order number	X20AO2438	X20cAO2438
Output format		
4 to 20 mA	INT 0x0000 to 0x7FFF / 1 LSB = 0x0001 = 488.281 nA	
0 to 20 mA	INT 0x0000 to 0x7FFF / 1 LSB = 0x0001 = 610.352 nA UINT 0x0000 to 0xFFFF / 1 LSB = 0x0001 = 305.176 nA	
0 to 24 mA	INT 0x0000 to 0x5DC0 / 1 LSB = 0x0001 = 1000 nA	
Load per channel	Max. 600 Ω	
Short-circuit proof	Yes, continuous	
Output filter	Active second-order low-pass filter / Cutoff frequency 19 Hz Configurable slew rate	
Max. gain drift		
4 to 20 mA	±0.0055%/°C <sup>2)</sup>	
0 to 20 mA	±0.005%/°C <sup>2)</sup>	
0 to 24 mA	±0.005%/°C <sup>2)</sup>	
Max. offset drift		
4 to 20 mA	±0.0035%/°C <sup>3)</sup>	
0 to 20 mA	±0.002%/°C <sup>3)</sup>	
0 to 24 mA	±0.002%/°C <sup>3)</sup>	
Error caused by load change <sup>4)</sup>		
4 to 20 mA	0.14%	
0 to 20 mA	0.1%	
0 to 24 mA	0.1%	
Nonlinearity	<0.003% <sup>5)</sup>	
Test voltage		
Channel - Channel	1000 VAC	
Channel - Bus	1000 VAC	
Channel - Ground	1000 VAC	
HART		
Transfer rate	1200 bit/s	
Operating frequencies	1200 Hz / 2200 Hz	
Burst operation possible	Yes	
Multi-drop operation		
Possible	Yes	
Stations	Up to 15	
Transmission amplitude		
Minimum	400 mV <sub>pp</sub>	
Typical	500 mV <sub>pp</sub>	
Maximum	600 mV <sub>pp</sub>	
Receiving amplitude		
Minimum	120 mV <sub>pp</sub>	
Maximum	1500 mV <sub>pp</sub>	
Electrical properties		
Electrical isolation	Channel isolated from channel and bus	
Operating conditions		
Mounting orientation		
Horizontal	Yes	
Vertical	Yes	
Installation elevation above sea level		
0 to 2000 m	No limitation	
>2000 m	Reduction of ambient temperature by 0.5°C per 100 m	
Degree of protection per EN 60529	IP20	
Ambient conditions		
Temperature		
Operation		
Horizontal mounting orientation	-25 to 60°C	
Vertical mounting orientation	-25 to 50°C	
Derating	See section "Derating".	
Storage	-40 to 85°C	
Transport	-40 to 85°C	
Relative humidity		
Operation	5 to 95%, non-condensing	Up to 100%, condensing
Storage	5 to 95%, non-condensing	
Transport	5 to 95%, non-condensing	
Mechanical properties		
Note	Order 1x terminal block X20TB12 separately. Order 1x bus module X20BM11 separately.	Order 1x terminal block X20TB12 separately. Order 1x bus module X20cBM11 separately.
Pitch	12.5 <sup>+0.2</sup> mm	

Table 2: X20AO2438, X20cAO2438 - Technical data

- 1) At 25°C
- 2) Based on the current output value.
- 3) Based on the respective output range.
- 4) Load change from 1  $\Omega$  → 600  $\Omega$ , resistive
- 5) Based on the entire output range.

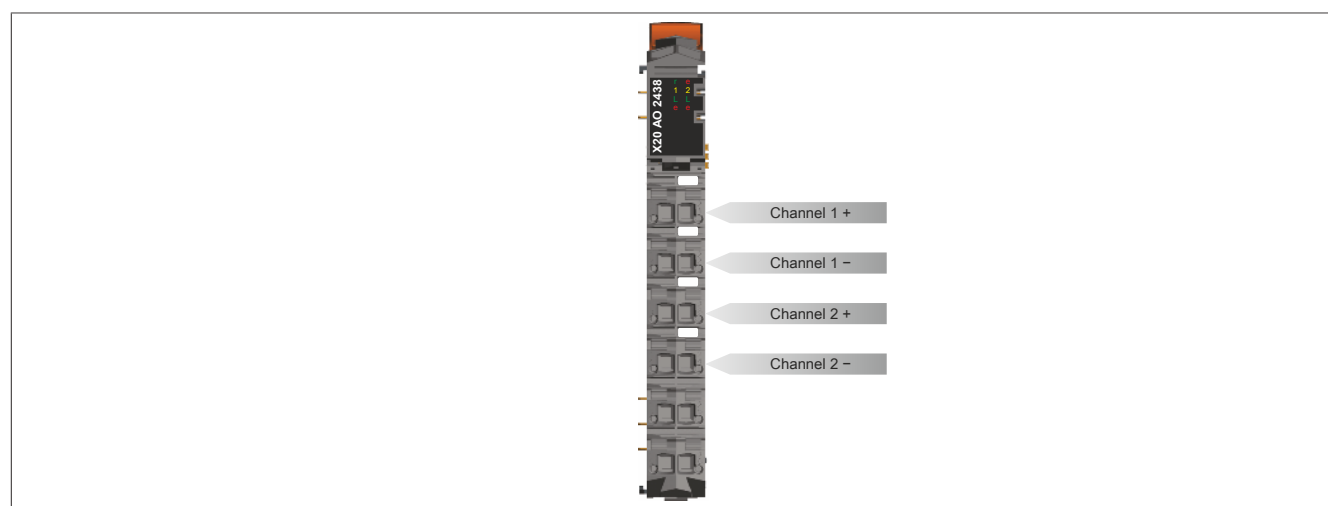
## 2.2 LED status indicators

For a description of the various operating modes, see section "Additional information - Diagnostic LEDs" in the X20 System user's manual.

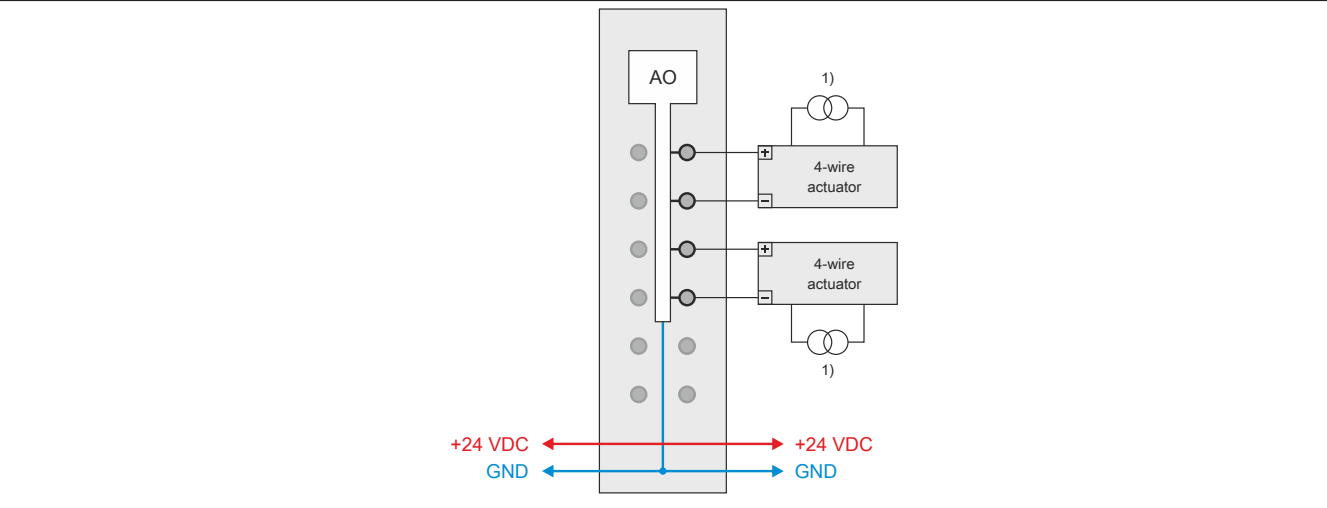
Figure	LED	Color	Status	Description
	Operating status			
	r	Green	Off	No power to module
			Single flash	UNLINK mode
			Double flash	BOOT mode (during firmware update) <sup>1)</sup>
			Blinking quickly	SYNC mode
			Blinking slowly	PREOPERATIONAL mode
			On	RUN mode
			Flickering (approx. 10 Hz)	Module is in OSP mode
	Module status			
	e	Red	Off	No power to module or everything OK
			Single flash	A conversion error has occurred. When an error occurs, the LED of the faulty analog output channel begins to double flash and this status is output.
			On	Error or reset status
	Analog output			
	1 - 2	Orange	Off	Indicates one of the following cases: <ul style="list-style-type: none"><li>No power to module</li><li>Channel disabled</li></ul>
			Single flash	Open line
			Double flash	A conversion error has occurred. A single flash is output on the red "e" module status LED.
			On	Digital/analog converter running, value OK
	HART link			
	L	Green	Off	Indicates one of the following cases: <ul style="list-style-type: none"><li>No power to module</li><li>HART disabled for the respective channel</li></ul>
			Flickering	Carrier signal active (DCD or RTS)
	HART error			
	e	Red	Off	Indicates one of the following cases: <ul style="list-style-type: none"><li>Communication taking place without errors</li><li>No power to module</li><li>HART disabled for the respective channel</li></ul>
			On	Communication error

1) Depending on the configuration, a firmware update can take up to several minutes.

## 2.3 Pinout



2.4 Connection example

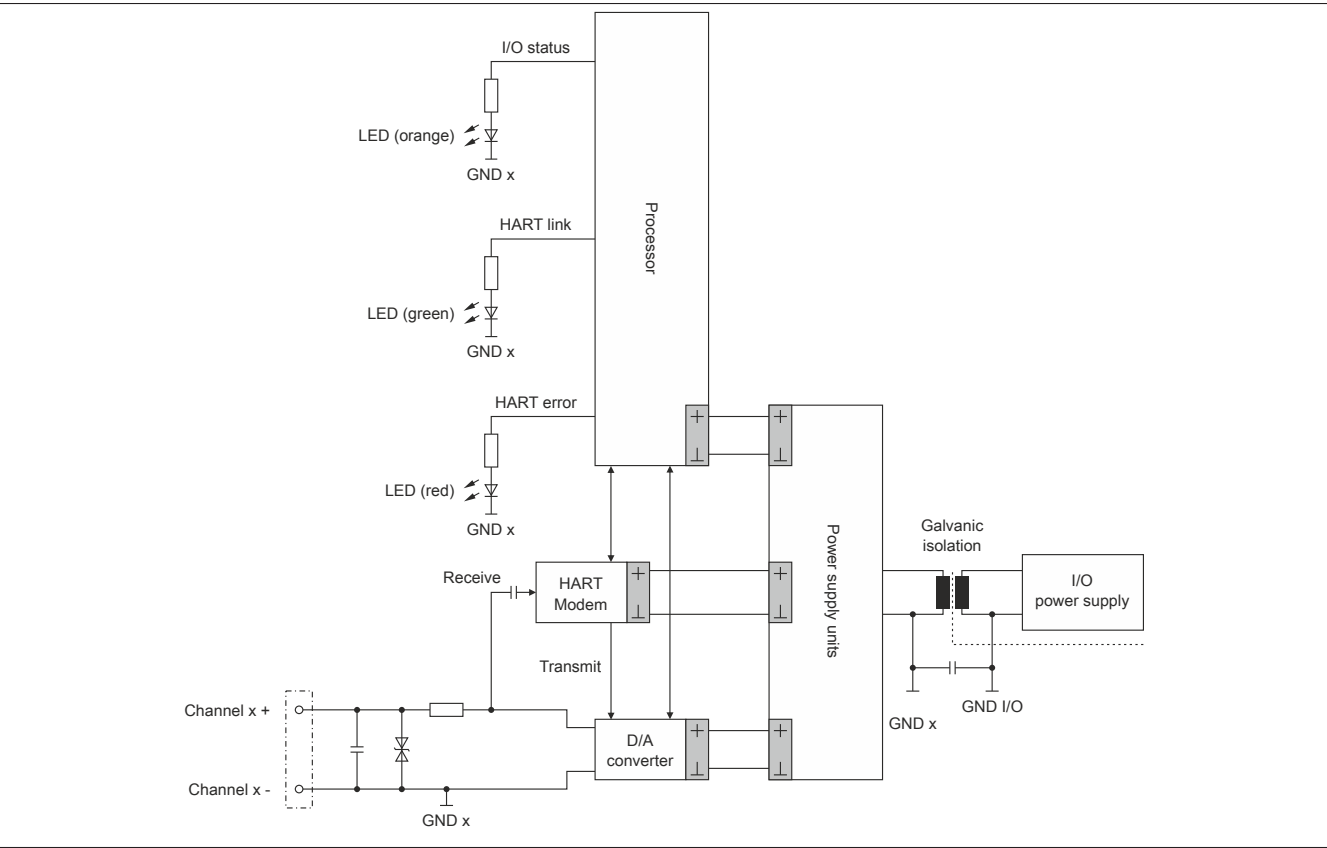


1) With external power supply.

2.5 OSP hardware requirements

In order to use OSP mode sensibly, it should be ensured when setting up the application that the power supply of the output module and controller are designed to be independent of each other.

2.6 Output circuit diagram





## 2.7 Operation

### 2.7.1 Derating

To ensure proper operation, the derating values listed below must be adhered to:

#### Horizontal installation

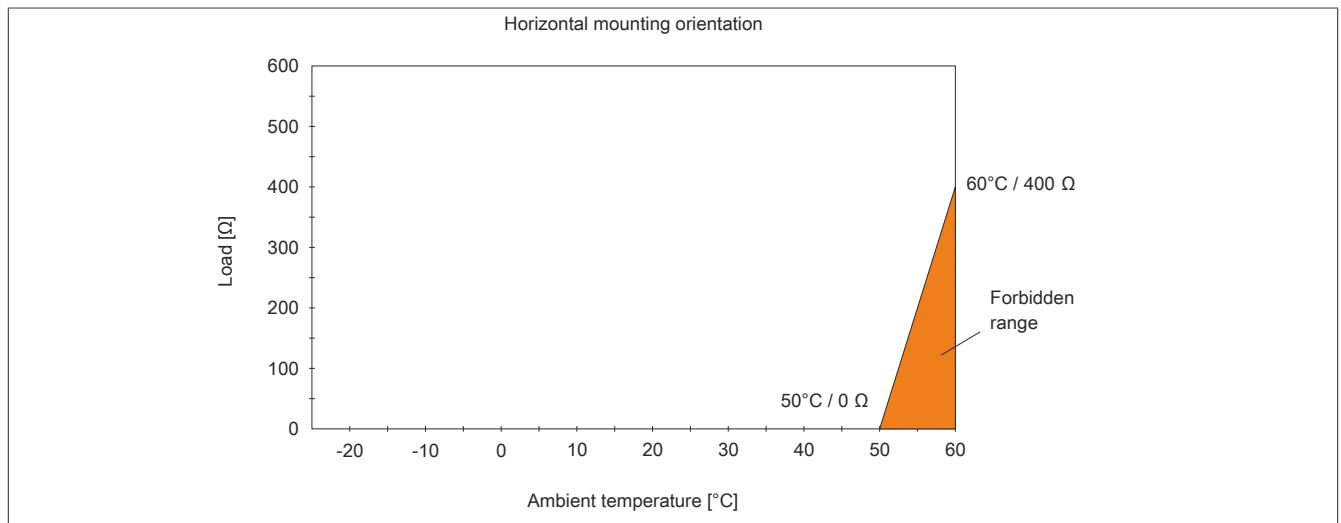


Figure 1: Derating the load with horizontal mounting

#### Vertical installation

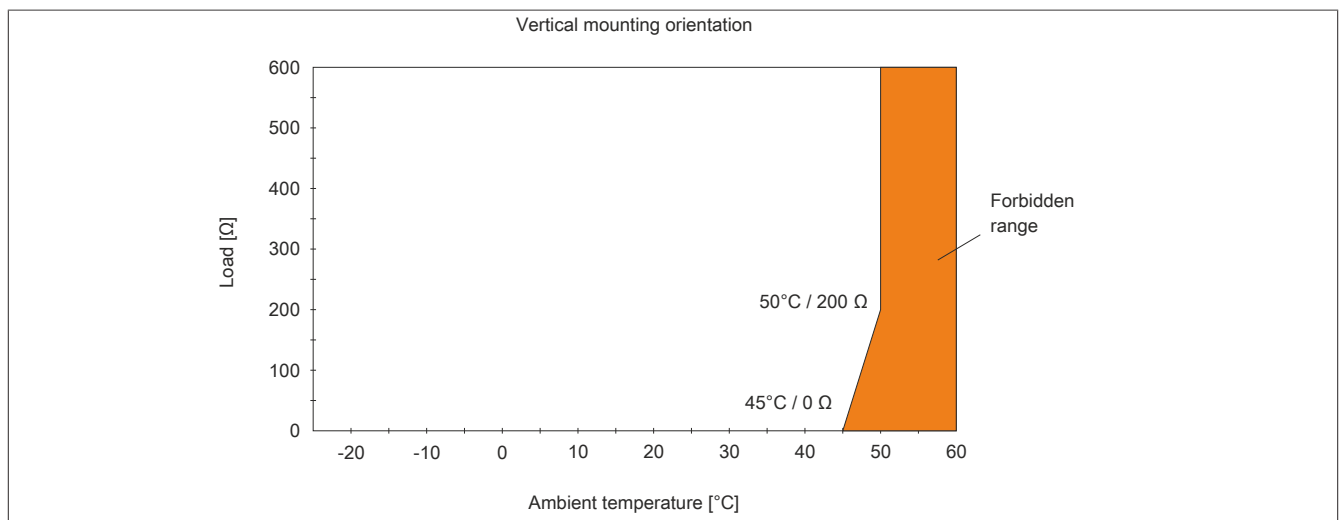


Figure 2: Derating the load with vertical mounting

### 2.7.2 Usage after the X20IF1091-1

If this module is operated after X2X Link module X20IF1091-1, delays may occur during the Flatstream transfer. For detailed information, see section "Data transfer on the Flatstream" in X20IF1091-1.

### 2.7.3 HART communication standard

This module supports the HART communication standard for data transfer, parameter configuration and diagnostics. The HART standard is used for the current range 4 to 20 mA. Be aware that the load is not permitted to fall below 230  $\Omega$  .

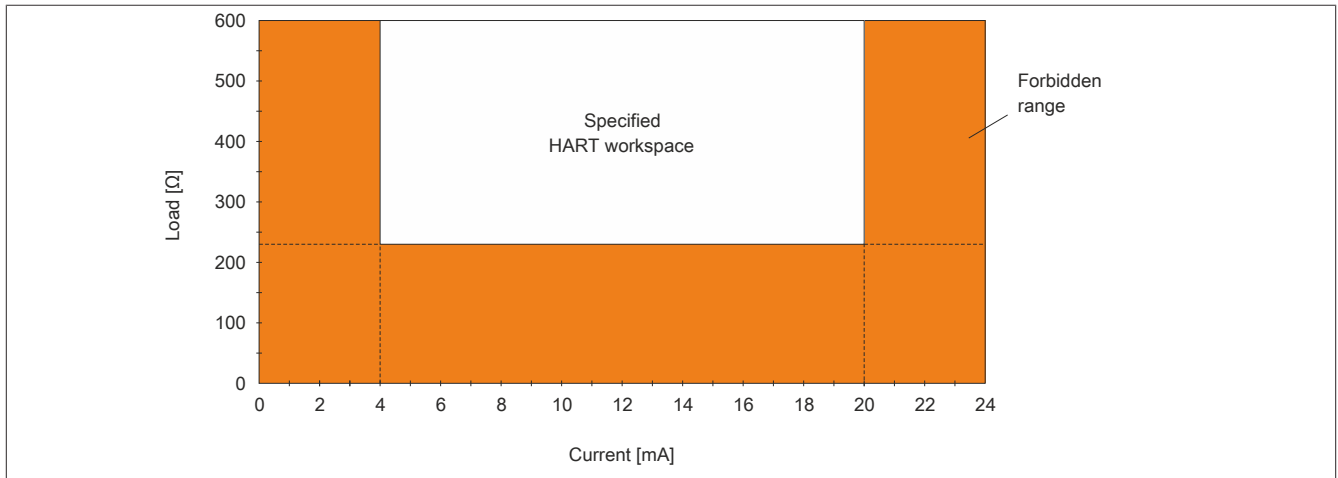


Figure 3: Specified HART operational range

Both current ranges 0 to 20 mA and 0 to 24 mA are supported by this module. HART communication can also be used in these ranges as well. It is important to make sure, however, that the output current lies within the specified HART operational range.

## 3 Function description

### 3.1 Analog outputs

The module has 2 independent electrically isolated channels with integrated HART modems. Both channels can be used to output an analog signal and handle HART communication. 2 registers need to be configured for one analog signal. The 2 channels operate independently, so 2 registers must be configured per channel to be used.

#### Setting the current signal

In order to output the desired current signal (default: 4 to 20 mA), the module must be provided with a normalized output value (default: 0 to 32767). In this way, the module can be used as a conventional output module.



#### Information:

Selecting operating mode "Scaling 0 to 20 mA (resolution 0 to 65535)" means that the corresponding "AnalogOutput" register is no longer interpreted internally as INT, but as UINT. A rebuild of the entire program is required to change the data type. The data type cannot be modified at runtime (e.g. via library).

#### Configurable output ramp

The slew rate of the analog signal (DACslewrate) can be limited. In this way, a kind of upper cutoff frequency can be defined.

The following formula applies:  $f(\text{analog}) = f(\text{output clock}) * \text{Permissible change} / \text{Max. } \Delta(\text{normalized output value})$

For error-free communication, it must be ensured that the frequency range of the digital HART signal is not influenced by the analog output. HART communicates in the frequency range from 950 to 2500 Hz.

Example (standard):  $f(\text{Analog}) = 152440 \text{ Hz} * 4 / (32767 - 0)$

Conclusion:  $f(\text{Analog}) = \sim 20 \text{ Hz} \ll 950 \text{ Hz} = f(\text{HART})$



#### Information:

The registers are described in ["Analog signal - Configuration" on page 52.](#)

## 3.2 HART

HART (Highway Addressable Remote Transducer) is a protocol for communicating with intelligent field devices. It was developed in order to more efficiently use the infrastructure for transferring analog signals. The digital HART notifications are modulated to the analog signal using Frequency Shift Keying (FSK). HART can thus use the same physical line as the analog signal without influencing the original function.

HART slaves are able to determine different process data independently and prepare HART concordantly. This protocol supports polling of the value of a process variable as well as its unit and status. Field devices usually supply their information after the master requests it. In newer revisions, it is also possible to transfer configuration data.

There are 2 different types of HART networks. In a point-to-point network, only one slave is connected to a HART master. Here, the analog signal and the HART signal can be transferred over the same line. Managing several slaves with HART requires what is known as a multidrop network. Here, each HART slave is assigned and identified by a unique address. Classic analog signals cannot be clearly traced in bus systems. As a result, the HART protocol does not support analog information transfers in multidrop networks up to and including HART Revision 5.



### Information:

#### Split range operation with HART AO modules

**Beginning with HART revision 6, bus stations that use an analog signal according to the split range method are written to separately. The HART protocol supports multidrop addressing as well as the use of analog signals for these applications.**

The module was designed based on HART-Revision 5. Only single-channel FSK scheme is available for transmitting the signals.

Since all HART frames are generated and evaluated in the application when using the FlatStream interface, information that isn't specified until later revisions can also be read.



### Information:

**The registers are described in "HART" on page 55.**

### 3.2.1 HART modem

The integrated HART modems physically use the same lines as the analog outputs. Digital information can be retrieved from the memory of the HART slave using additional higher frequency signals.

**A distinction is made between the following connection variants for each channel.**

- Point-to-point (connection of a HART node to the channel):  
→ Evaluation of the analog signal  
and  
→ Recording of up to 4 pieces of HART information
- Multidrop (connection of up to 15 HART nodes to the channel):  
→ Recording of one piece of HART information per connected node

### Specific features

- Galvanically isolated per channel
- Up to 4 or 15 HART input variables per channel
- Configurable output ramp (DAC slew rate) to transfer HART and analog signals without impairment (default: 210 ms full scale)
- Selectable error strategy (static replacement value or retention of the last permissible value)
- Cyclic "HART status" polling (HART command 0). The status information received is made available for channel diagnostics.
- Compatible with additional secondary master in the HART network (module acts as primary master)
- "HART communication error bit" (indicates termination of the HART connection if the connection was previously established successfully)
- Optional: BURST mode for one node per channel
- Optional: Cyclic "HART variables" polling (HART command 3 or 9)
- Optional: Flatstream functionality (module as a bridge for HART packets)

#### 3.2.1.1 Limitations

In exceptional cases, the following error message may appear in Automation Studio when connecting specific DTM devices:

- "Error connecting DTM device <Device name>. (The persistency operation InvokePersistSave() failed!)"

This error can be corrected by adding the registry entries required to find the necessary COM interface. To do this, a new text file with file extension ".reg" must be created on the affected PC and the following content added:

```
(Windows Registry Editor Version 5.00)

[HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{0000000c-0000-0000-C000-000000000046}]
@="IStream"
[HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{0000000c-0000-0000-C000-000000000046}\InprocServer32]
@="combase.dll"
"ThreadingModel"="Both"
```

After saving, the values must be imported into the registry by double-clicking on the .reg file. A confirmation prompt must be answered with "Yes" or "OK". Automation Studio must then be restarted.

### 3.2.2 HART - Configuration

HART modules are analog modules equipped with a HART modem. For each channel, a separate HART network can be managed by the module, which acts as a primary master. Once configured successfully, the HART information is stored in the module where it can then be used by the PLC.

The number of HART slaves must be specified in the configuration.

If only one slave is connected to the HART channel, then it is part of a point-to-point network. The module can then prepare up to 4 process variables from the connected slave.

Multidrop mode allows up to 15 HART slaves to be connected. The primary process variable from each slave is then retrieved.

In addition to the type of network, the user can choose between 2 different communication behaviors.

- **Polling**

The principles of polling are used in conventional HART communication. The module queries the data of the HART slave individually and receives the corresponding information from the slave in response.

- **Burst mode**

If a HART node should be queried at short intervals, the user can configure burst mode for one node per channel. In this case, the slave sends the information from this node cyclically without a new prompt from the master.

#### Extended configuration

The additional configuration registers are pre-assigned with values when the module is started. In many systems, the user does not have to make any adjustments to them. The register values should only be changed if communication in the HART network is not running satisfactorily.



#### **Information:**

The registers are described in "[HART configuration](#)" on page 55.

### 3.2.3 HART - Communication

After the configuration is completed, the information is retrieved automatically and transferred to the module registers. A separate register is implemented in the module for each piece of information. HART modules are designed to query up to 15 pieces of information per channel. The module reads in the data, stores it in temporary memory and prepares it for retrieval. When the X2X master accesses the module registers, it is irrelevant whether the HART data originates from a point-to-point or multidrop network.

#### Overview of internal module mapping

	Point-to-point network (1 HART slave)	Multidrop network (2 to 15 HART slaves)
(Pv)Input_01	Primary piece of information from HART node 1	Primary piece of information from HART node 1
(Pv)Input_02	Secondary piece of information from HART node 1	Primary piece of information from HART node 2
...	...	...
(Pv)Input_04	Quaternary piece of information from HART node 1	Primary piece of information from HART node 4
(Pv)Input_05	Reserved	Primary piece of information from HART node 5
...	...	...
(Pv)Input_15	Reserved	Primary piece of information from HART node 15

The HART specifications stipulates that information from a HART node be split into various pieces. The value of a process variable is stored to the respective "[PvInput](#)" on page 56 register and has a size of 4 bytes (REAL) per the HART specification. Due to the length limitation of 30 bytes on the X2X Link network, there are limitations to the number of possible cyclic variables. It is recommended to transfer a maximum of 2 "[PvInput](#)" on page 56 registers cyclically to the X2X master. All other information should be read in a different way. To access HART information, the user can choose between the following methods:

- **Acyclic** - If library AsIOAcc is used, information is queried acyclically only when it is needed, i.e. communication can be adapted to the program sequence of the X2X master. In this way, all of the necessary module registers on the X2X Link network can be queried despite the length limitation. This type of information exchange is not real-time capable.
- **Cyclic**: Data points configured for cyclic transfer are read once per bus cycle. This procedure allows real-time capable information exchange between the module and X2X master. The length limitation may prevent all data from being queried within one cycle, however.
- **Multiplexed** - A runtime driver can be used to transfer the HART data points in the I/O mapping. In this case, the HART process data is transmitted alternately (time multiplexed). Communication remains real-time capable. Multiple bus cycles are needed to update all data points, however.



#### Information:

**This mode cannot be used when using the module after a bus controller.**

**"Multiplexed" data transfer is used exclusively for HART data points.**

**Information from the analog inputs/outputs is always transferred cyclically (see above).**

- **Flatstream** - HART modules are equipped with a Flatstream interface. When using Flatstream communication, the module is used as a bridge between the X2X master and HART slave, i.e. the X2X master communicates directly with the HART slave (see "[Flatstream communication](#)" on page 22). Flatstream communication is also not real-time capable. It allows unrestricted access to the HART slave. The user must have sufficient knowledge of the HART protocol command set as well as the capabilities of the corresponding HART slave.



#### Information:

**The registers are described in "[HART communication](#)" on page 56.**

3.2.4 HART - Status information

The status information can be used to check whether a read-in value is valid. Per the HART specification, this type of status register consists of 2 parts. The "response code" is stored in the high byte; the "field device status" is stored in the low byte. This makes it possible to check the current state of a read process variable.


The status information can be checked before cached process information is processed further. If the current value is 0x0000, no errors were detected during the HART transfer and the information of the checked node can be used. If a different value is present, the situation in the HART network should be checked. The extension registers can be used for this purpose, for example.

HART-specific response code

0x82 ... Receive buffer overflow	If an error occurs during HART communication, the response code is written.
0x88 ... Incorrect checksum	
0x90 ... Invalid protocol structure	
0xA0 ... Overflow	
0xC0 ... Impermissible parity	
0xFF ... Timeout	

Retrieving the read-in information

After the node data has been successfully transferred to the module registers, the information can be retrieved from the module. Separate registers have been implemented in the module for each piece of information.



**Information:**

The register is described in ["Status of the process variables" on page 57](#).



### 3.2.5 HART to Flatstream

When using Flatstream communication, the module acts as a bridge between the X2X master and an intelligent field device connected to the module. Flatstream mode can be used for either point-to-point connections as well as for multidrop systems. Specific algorithms such as timeout and checksum monitoring are usually managed automatically. During normal operation, the user does not have access to these details.

HART is considered a master-slave network where half-duplex communication takes place asynchronously. Various features have been included to ensure that signals are transmitted without errors.

For example, the user can increase the length of the preamble, thus making the transmission more secure. However, this also has an effect on the percentage of payload data and overhead.

Additional information about HART can be found at [www.HARTcomm.org](http://www.HARTcomm.org).

#### How it works

The module has 2 independent channels. When using Flatstream, the channel number must therefore be specified. The general structure of a Flatstream frame is extended as follows.

Input/Output sequence		Tx/Rx bytes			
(unchanged)		Control byte (unchanged)	Channel number		HART frame (without preamble and checksum)
HART frame with Flatstream					
Startup	ADDR	CMD	BCNT	(STS)	(DATA)

Startup Start identification

ADDR Address within the HART network

CMD HART command

BCNT Byte counters (number of remaining bytes)

\*STS Status of the last command received. Information about the operating mode of the HART slave and communication errors (if supported, returns from the HART slave)

\*DATA Data (if necessary for the command)

#### Examples of HART commands

Command	Function
0x00	Read slave ID
0x03	Read current value and up to 4 variables
0x09	Read in up to 4 variables including status
0x21	Read variables

### 3.3 OSP mode

In function model "OSP" (Operator Set Predefined), the user defines an analog value or a digital pattern. This OSP value is output as soon as the communication between the module and master is aborted.

#### 3.3.1 Hardware requirements

In order to use OSP mode sensibly, it should be ensured when setting up the application that the power supply of the output module and controller are designed to be independent of each other.

#### 3.3.2 Functionality

The user has the choice between 2 OSP modes:

- Retain last valid value
- Replace with static value

In the first case, the module retains the last value recognized as a valid output status.

When selecting mode "Replace with static value", a plausible output value must be entered in the associated value register. When an OSP event occurs, this value is output instead of the value currently requested by the task.

If an OSP event occurs, e.g. communication between the module and master controller aborted, then bit OSPValid is reset on the module. The module enters the OSP state and output occurs according to the configuration in register OSPMode.



The following generally applies:

Even after regeneration of the communication channel, the OSP replacement value is still pending. The OSP state is only exited again when a set OSPValid bit is transferred.

When the master controller is restarted, bit OSPValid bit is reinitialized in the master controller. It must be set once more by the application and transferred via the bus.

In the event of brief communication errors between the module and master controller (e.g. due to EMC), the cyclic registers fail to refresh for several bus cycles. Within the module, bit OSPValid is reset; the set bit is retained in the controller, however. During the next successful transfer, the module-internal OSPValid bit is set again and the module automatically returns to normal mode.

If the task in the master controller needs the information about which output mode the module is currently in, bit ModulOK can be evaluated.



#### Warning!

**If bit OSPValid bit is reset to "0" by the module, the output status no longer depends on the responsible task in the master controller. Nevertheless, output is made depending on the configuration of the OSP replacement value.**



#### Information:

The registers are described in ["Function model "OSP"" on page 63.](#)

### 3.4 NetTime Technology

NetTime refers to the ability to precisely synchronize and transfer system times between individual components of the controller or network (controller, I/O modules, X2X Link, POWERLINK, etc.).

This allows the moment that events occur to be determined system-wide with microsecond precision. Upcoming events can also be executed precisely at a specified moment.



#### 3.4.1 Time information

Various time information is available in the controller or on the network:

- System time (on the PLC, Automation PC, etc.)
- X2X Link time (for each X2X Link network)
- POWERLINK time (for each POWERLINK network)
- Time data points of I/O modules

The NetTime is based on 32-bit counters, which are increased with microsecond resolution. The sign of the time information changes after 35 min, 47 s, 483 ms and 648  $\mu$ s; an overflow occurs after 71 min, 34 s, 967 ms and 296  $\mu$ s.

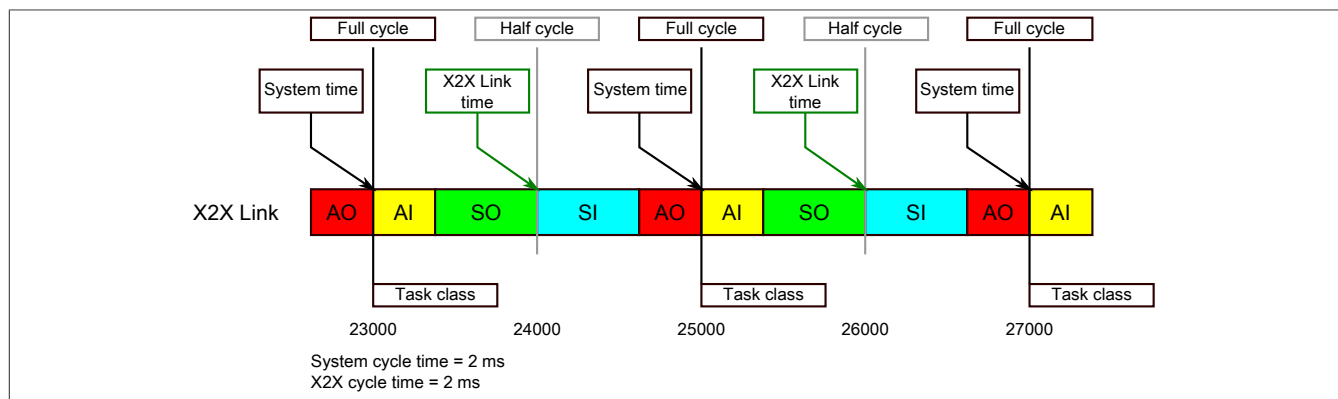
The initialization of the times is based on the system time during the startup of the X2X Link, the I/O modules or the POWERLINK interface.

Current time information in the application can also be determined via library AsIOTime.

##### 3.4.1.1 Controller data points

The NetTime I/O data points of the controller are latched to each system clock and made available.

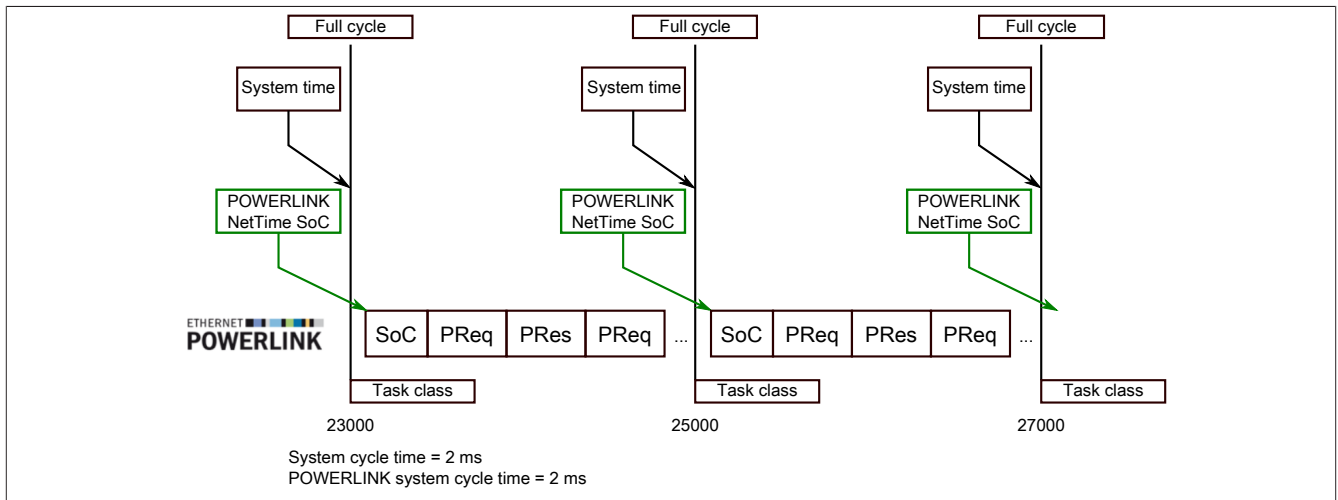
##### 3.4.1.2 X2X Link - Reference time point



The reference time point on the X2X Link network is always calculated at the half cycle of the X2X Link cycle. This results in a difference between the system time and the X2X Link reference time point when the reference time is read out.

In the example above, this results in a difference of 1 ms, i.e. if the system time and X2X Link reference time are compared at time 25000 in the task, then the system time returns the value 25000 and the X2X Link reference time returns the value 24000.

### 3.4.1.3 POWERLINK - Reference time point

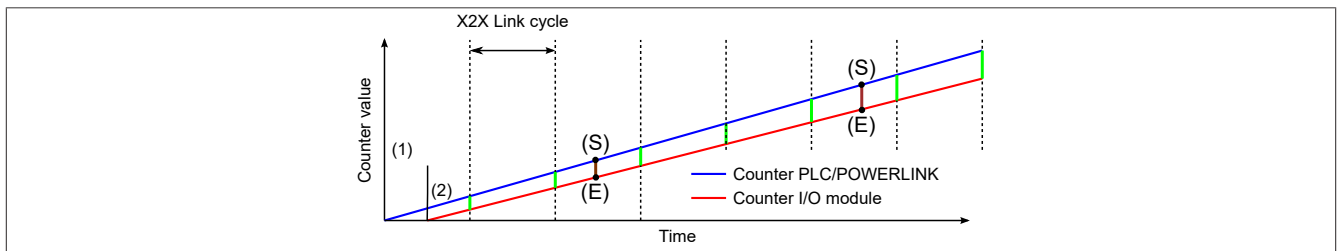


The POWERLINK reference time point is always calculated at the start of cycle (SoC) of the POWERLINK network. The SoC starts 20 µs after the system clock due to the system. This results in the following difference between the system time and the POWERLINK reference time:

POWERLINK reference time = System time - POWERLINK cycle time + 20 µs

In the example above, this means a difference of 1980 µs, i.e. if the system time and POWERLINK reference time are compared at time 25000 in the task, then the system time returns the value 25000 and the POWERLINK reference time returns the value 23020.

### 3.4.1.4 Synchronization of system time/POWERLINK time and I/O module



At startup, the internal counters for the controller/POWERLINK (1) and the I/O module (2) start at different times and increase the values with microsecond resolution.

At the beginning of each X2X Link cycle, the controller or POWERLINK network sends time information to the I/O module. The I/O module compares this time information with the module's internal time and forms a difference (green line) between the two times and stores it.

When a NetTime event (E) occurs, the internal module time is read out and corrected with the stored difference value (brown line). This means that the exact system moment (S) of an event can always be determined, even if the counters are not absolutely synchronous.

#### Note

The deviation from the clock signal is strongly exaggerated in the picture as a red line.

### 3.4.2 Timestamp functions

NetTime-capable modules provide various timestamp functions depending on the scope of functions. If a timestamp event occurs, the module immediately saves the current NetTime. After the respective data is transferred to the controller, including this precise moment, the controller can then evaluate the data using its own NetTime (or system time), if necessary.

For details, see the respective module documentation.

#### 3.4.2.1 Time-based inputs

NetTime Technology can be used to determine the exact moment of a rising edge at an input. The rising and falling edges can also be detected and the duration between 2 events can be determined.

**Information:**

**The determined moment always lies in the past.**

#### 3.4.2.2 Time-based outputs

NetTime Technology can be used to specify the exact moment of a rising edge on an output. The rising and falling edges can also be specified and a pulse pattern generated from them.

**Information:**

**The specified time must always be in the future, and the set X2X Link cycle time must be taken into account for the definition of the moment.**

#### 3.4.2.3 Time-based measurements

NetTime Technology can be used to determine the exact moment of a measurement that has taken place. Both the starting and end moment of the measurement can be transmitted.

## 3.5 Flatstream communication

### 3.5.1 Introduction

B&R offers an additional communication method for some modules. "Flatstream" was designed for X2X and POWERLINK networks and allows data transfer to be adapted to individual demands. Although this method is not 100% real-time capable, it still allows data transfer to be handled more efficiently than with standard cyclic polling.

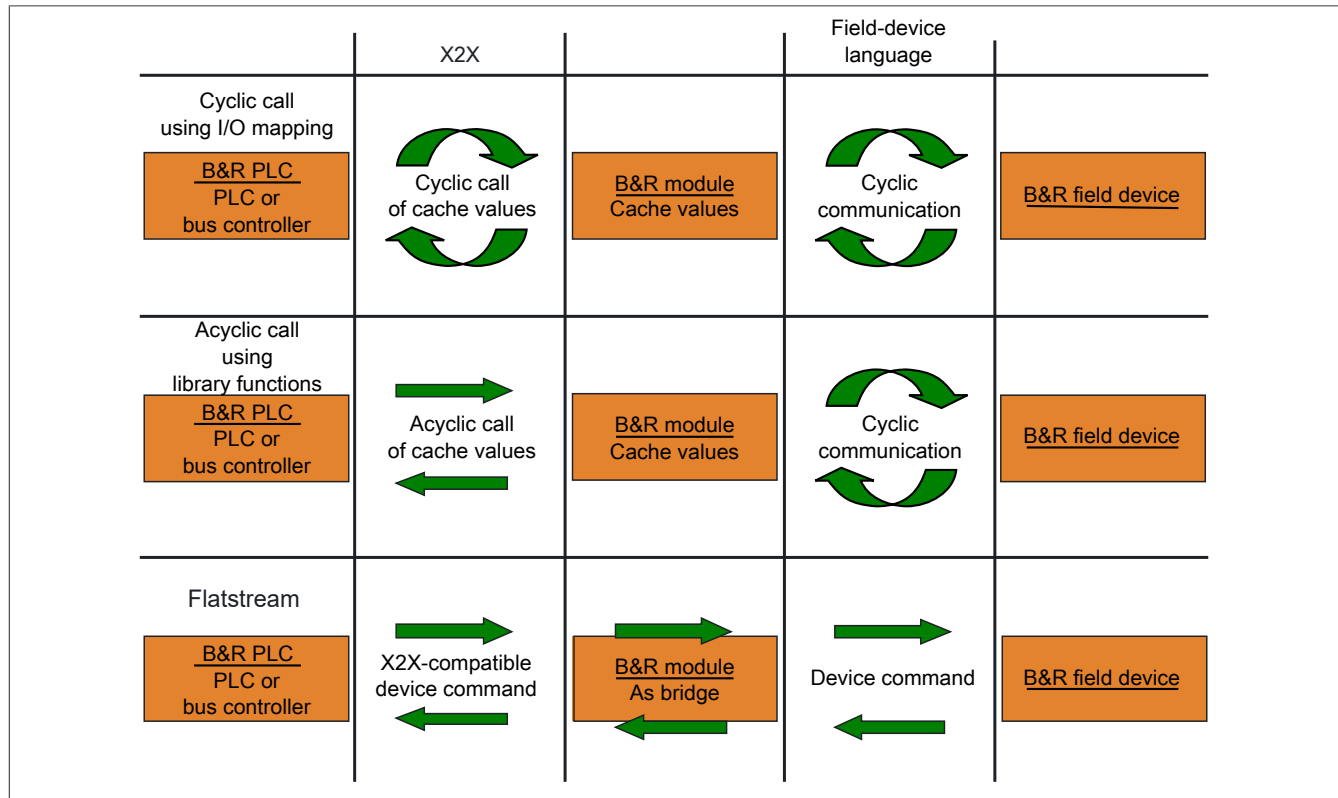


Figure 4: 3 types of communication

Flatstream extends cyclic and acyclic data queries. With Flatstream communication, the module acts as a bridge. The module is used to pass controller requests directly on to the field device.

### 3.5.2 Message, segment, sequence, MTU

The physical properties of the bus system limit the amount of data that can be transmitted during one bus cycle. With Flatstream communication, all messages are viewed as part of a continuous data stream. Long data streams must be broken down into several fragments that are sent one after the other. To understand how the receiver puts these fragments back together to get the original information, it is important to understand the difference between a message, a segment, a sequence and an MTU.

#### Message

A message refers to information exchanged between 2 communicating partner stations. The length of a message is not restricted by the Flatstream communication method. Nevertheless, module-specific limitations must be considered.

#### Segment (logical division of a message):

A segment has a finite size and can be understood as a section of a message. The number of segments per message is arbitrary. So that the recipient can correctly reassemble the transferred segments, each segment is preceded by a byte with additional information. This control byte contains information such as the length of a segment and whether the approaching segment completes the message. This makes it possible for the receiving station to interpret the incoming data stream correctly.

#### Sequence (how a segment must be arranged physically):

The maximum size of a sequence corresponds to the number of enabled Rx or Tx bytes (later: "MTU"). The transmitting station splits the transmit array into valid sequences. These sequences are then written successively to the MTU and transferred to the receiving station where they are lined up together again. The receiver stores the incoming sequences in a receive array, obtaining an image of the data stream in the process.

With Flatstream communication, the number of sequences sent are counted. Successfully transferred sequences must be acknowledged by the receiving station to ensure the integrity of the transfer.

#### MTU (Maximum Transmission Unit) - Physical transport:

MTU refers to the enabled USINT registers used with Flatstream. These registers can accept at least one sequence and transfer it to the receiving station. A separate MTU is defined for each direction of communication. OutputMTU defines the number of Flatstream Tx bytes, and InputMTU specifies the number of Flatstream Rx bytes. The MTUs are transported cyclically via the X2X Link network, increasing the load with each additional enabled USINT register.

#### Properties

Flatstream messages are not transferred cyclically or in 100% real time. Many bus cycles may be needed to transfer a particular message. Although the Rx and Tx registers are exchanged between the transmitter and the receiver cyclically, they are only processed further if explicitly accepted by register "InputSequence" or "OutputSequence".

#### Behavior in the event of an error (brief summary)

The protocol for X2X and POWERLINK networks specifies that the last valid values should be retained when disturbances occur. With conventional communication (cyclic/acyclic data queries), this type of error can generally be ignored.

In order for communication to also take place without errors using Flatstream, all of the sequences issued by the receiver must be acknowledged. If Forward functionality is not used, then subsequent communication is delayed for the length of the disturbance.

If Forward functionality is being used, the receiving station receives a transmission counter that is incremented twice. The receiver stops, i.e. it no longer returns any acknowledgments. The transmitting station uses SequenceAck to determine that the transfer was faulty and that all affected sequences must be repeated.

### 3.5.3 The Flatstream principle

#### Requirements

Before Flatstream can be used, the respective communication direction must be synchronized, i.e. both communication partners cyclically query the sequence counter on the remote station. This checks to see if there is new data that should be accepted.

#### Communication

If a communication partner wants to transmit a message to its remote station, it should first create a transmit array that corresponds to Flatstream conventions. This allows the Flatstream data to be organized very efficiently without having to block other important resources.

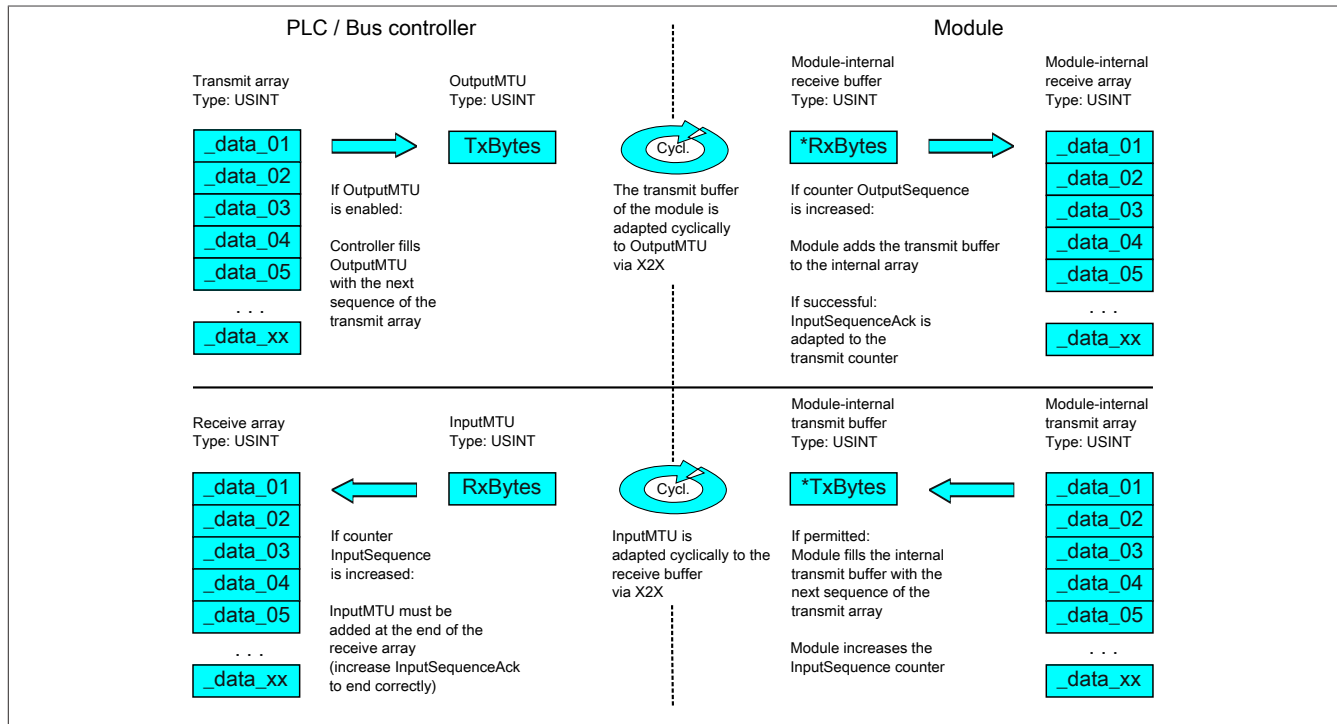


Figure 5: Flatstream communication

#### Procedure

The first thing that happens is that the message is broken into valid segments of up to 63 bytes, and the corresponding control bytes are created. The data is formed into a data stream made up of one control bytes per associated segment. This data stream can be written to the transmit array. The maximum size of each array element matches that of the enabled MTU so that one element corresponds to one sequence. If the array has been completely created, the transmitter checks whether the MTU is permitted to be re-filled. It then copies the first element of the array or the first sequence to the Tx byte registers. The MTU is transported to the receiver station via X2X Link and stored in the corresponding Rx byte registers. To signal that the data should be accepted by the receiver, the transmitter increases its SequenceCounter. If the communication direction is synchronized, the remote station detects the incremented SequenceCounter. The current sequence is appended to the receive array and acknowledged by SequenceAck. This acknowledgment signals to the transmitter that the MTU can now be refilled.

If the transfer is successful, the data in the receive array will correspond 100% to the data in the transmit array. During the transfer, the receiving station must detect and evaluate the incoming control bytes. A separate receive array should be created for each message. This allows the receiver to immediately begin further processing of messages that are completely transferred.



### 3.5.4 Registers for Flatstream mode

5 registers are available for configuring Flatstream. The default configuration can be used to transmit small amounts of data relatively easily.



#### Information:

The controller communicates directly with the field device via registers "OutputSequence" and "InputSequence" as well as the enabled Tx and RxBytes bytes. For this reason, the user must have sufficient knowledge of the communication protocol being used on the field device.

#### 3.5.4.1 Flatstream configuration

To use Flatstream, the program sequence must first be expanded. The cycle time of the Flatstream routines must be set to a multiple of the bus cycle. Other program routines should be implemented in Cyclic #1 to ensure data consistency.

At the absolute minimum, registers "InputMTU" and "OutputMTU" must be set. All other registers are filled in with default values at the beginning and can be used immediately. These registers are used for additional options, e.g. to transfer data in a more compact way or to increase the efficiency of the general procedure.

The Forward registers extend the functionality of the Flatstream protocol. This functionality is useful for substantially increasing the Flatstream data rate, but it also requires quite a bit of extra work when creating the program sequence.



#### Information:

In the rest of this description, the names "OutputMTU" and "InputMTU" do not refer to the registers names. Instead, they are used as synonyms for the currently enabled Tx or Rx bytes.



#### Information:

The registers are described in ["Flatstream registers" on page 60](#).

Registers are described in section "Flatstream communication" in the respective data sheets.

#### 3.5.4.2 Flatstream operation

When using Flatstream, the communication direction is very important. For transmitting data to a module (output direction), Tx bytes are used. For receiving data from a module (input direction), Rx bytes are used. Registers "OutputSequence" and "InputSequence" are used to control or secure communication, i.e. the transmitter uses them to give instructions to apply data and the receiver confirms a successfully transferred sequence.



#### Information:

The registers are described in ["Flatstream registers" on page 60](#).

Registers are described in section "Flatstream communication" in the respective data sheets.

##### 3.5.4.2.1 Format of input and output bytes

Name:

"Format of Flatstream" in Automation Studio

On some modules, this function can be used to set how the Flatstream input and output bytes (Tx or Rx bytes) are transferred.

- **Packed:** Data is transferred as an array.
- **Byte-by-byte:** Data is transferred as individual bytes.

### 3.5.4.2.2 Transporting payload data and control bytes

The Tx and Rx bytes are cyclic registers used to transport the payload data and the necessary control bytes. The number of active Tx and Rx bytes is taken from the configuration of registers "OutputMTU" and "InputMTU", respectively.

In the user program, only the Tx and Rx bytes from the controller can be used. The corresponding counterparts are located in the module and are not accessible to the user. For this reason, the names were chosen from the point of view of the controller.

- "T" - "Transmit" → Controller transmits data to the module.
- "R" - "Receive" → Controller receives data from the module.

#### 3.5.4.2.2.1 Control bytes

In addition to the payload data, the Tx and Rx bytes also transfer the necessary control bytes. These control bytes contain additional information about the data stream so that the receiver can reconstruct the original message from the transferred segments.

#### Bit structure of a control byte

Bit	Name	Value	Information
0 - 5	SegmentLength	0 - 63	Size of the subsequent segment in bytes (default: Max. MTU size - 1)
6	nextCBPos	0	Next control byte at the beginning of the next MTU
		1	Next control byte directly after the end of the current segment
7	MessageEndBit	0	Message continues after the subsequent segment
		1	Message ended by the subsequent segment

#### SegmentLength

The segment length lets the receiver know the length of the coming segment. If the set segment length is insufficient for a message, then the information must be distributed over several segments. In these cases, the actual end of the message is detected using bit 7 (control byte).



#### Information:

**The control byte is not included in the calculation to determine the segment length. The segment length is only derived from the bytes of payload data.**

#### nextCBPos

This bit indicates the position where the next control byte is expected. This information is especially important when using option "MultiSegmentMTU".

When using Flatstream communication with MultiSegmentMTUs, the next control byte is no longer expected in the first Rx byte of the subsequent MTU, but transferred directly after the current segment.

#### MessageEndBit

"MessageEndBit" is set if the subsequent segment completes a message. The message has then been completely transferred and is ready for further processing.



#### Information:

**In the output direction, this bit must also be set if one individual segment is enough to hold the entire message. The module will only process a message internally if this identifier is detected.**

**The size of the message being transferred can be calculated by adding all of the message's segment lengths together.**

Flatstream formula for calculating message length:

Message [bytes] = Segment lengths (all CBs without ME) + Segment length (of the first CB with ME)	CB	Control byte
	ME	MessageEndBit

### 3.5.4.2.3 Communication status

The communication status is determined via registers "OutputSequence" and "InputSequence".

- OutputSequence contains information about the communication status of the controller. It is written by the controller and read by the module.
- InputSequence contains information about the communication status of the module. It is written by the module and should only be read by the controller.

#### 3.5.4.2.3.1 Relationship between OutputSequence and InputSequence

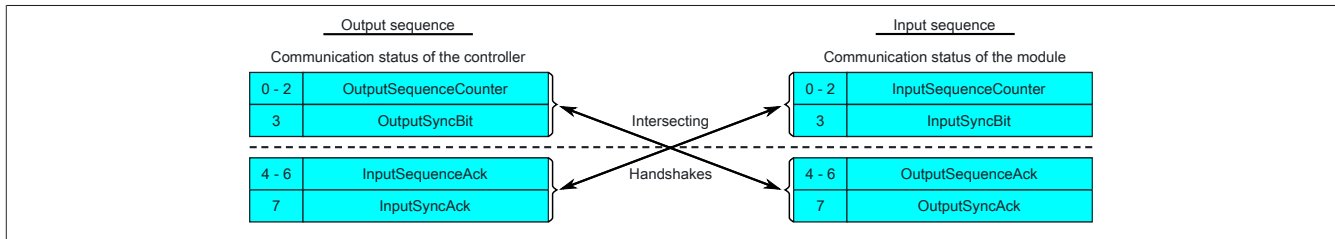


Figure 6: Relationship between OutputSequence and InputSequence

Registers OutputSequence and InputSequence are logically composed of 2 half-bytes. The low part indicates to the remote station whether a channel should be opened or whether data should be accepted. The high part is to acknowledge that the requested action was carried out.

#### SyncBit and SyncAck

If SyncBit and SyncAck are set in one communication direction, then the channel is considered "synchronized", i.e. it is possible to send messages in this direction. The status bit of the remote station must be checked cyclically. If SyncAck has been reset, then SyncBit on that station must be adjusted. Before new data can be transferred, the channel must be resynchronized.

#### SequenceCounter and SequenceAck

The communication partners cyclically check whether the low nibble on the remote station changes. When one of the communication partners finishes writing a new sequence to the MTU, it increments its SequenceCounter. The current sequence is then transmitted to the receiver, which acknowledges its receipt with SequenceAck. In this way, a "handshake" is initiated.



#### Information:

**If communication is interrupted, segments from the unfinished message are discarded. All messages that were transferred completely are processed.**

## Function description

### 3.5.4.3 Synchronization

During synchronization, a communication channel is opened. It is important to make sure that a module is present and that the current value of SequenceCounter is stored on the station receiving the message. Flatstream can handle full-duplex communication. This means that both channels / communication directions can be handled separately. They must be synchronized independently so that simplex communication can theoretically be carried out as well.

#### Synchronization in the output direction (controller as the transmitter):

The corresponding synchronization bits (OutputSyncBit and OutputSyncAck) are reset. Because of this, Flatstream cannot be used at this point in time to transfer messages from the controller to the module.

#### Algorithm

1) The controller must write 000 to OutputSequenceCounter and reset OutputSyncBit. The controller must cyclically query the high nibble of register "InputSequence" (checks for 000 in OutputSequenceAck and 0 in OutputSyncAck). The module does not accept the current contents of InputMTU since the channel is not yet synchronized. The module matches OutputSequenceAck and OutputSyncAck to the values of OutputSequenceCounter and OutputSyncBit.
2) If the controller registers the expected values in OutputSequenceAck and OutputSyncAck, it is permitted to increment OutputSequenceCounter. The controller continues cyclically querying the high nibble of register "OutputSequence" (checks for 001 in OutputSequenceAck and 0 in InputSyncAck). The module does not accept the current contents of InputMTU since the channel is not yet synchronized. The module matches OutputSequenceAck and OutputSyncAck to the values of OutputSequenceCounter and OutputSyncBit.
3) If the controller registers the expected values in OutputSequenceAck and OutputSyncAck, it is permitted to increment OutputSequenceCounter. The controller continues cyclically querying the high nibble of register "OutputSequence" (checks for 001 in OutputSequenceAck and 1 in InputSyncAck).  <b>Note:</b> Theoretically, data can be transferred from this point forward. However, it is still recommended to wait until the output direction is completely synchronized before transferring data. The module sets OutputSyncAck. The output direction is synchronized, and the controller can transmit data to the module.

#### Synchronization in the input direction (controller as the receiver):

The corresponding synchronization bits (InputSyncBit and InputSyncAck) are reset. Because of this, Flatstream cannot be used at this point in time to transfer messages from the module to the controller.

#### Algorithm

The module writes 000 to InputSequenceCounter and resets InputSyncBit. The module monitors the high nibble of register "OutputSequence" and expects 000 in InputSequenceAck and 0 in InputSyncAck.
1) The controller is not permitted to accept the current contents of InputMTU since the channel is not yet synchronized. The controller must match InputSequenceAck and InputSyncAck to the values of InputSequenceCounter and InputSyncBit. If the module registers the expected values in InputSequenceAck and InputSyncAck, it increments InputSequenceCounter. The module monitors the high nibble of register "OutputSequence" and expects 001 in InputSequenceAck and 0 in InputSyncAck.
2) The controller is not permitted to accept the current contents of InputMTU since the channel is not yet synchronized. The controller must match InputSequenceAck and InputSyncAck to the values of InputSequenceCounter and InputSyncBit. If the module registers the expected values in InputSequenceAck and InputSyncAck, it sets InputSyncBit. The module monitors the high nibble of register "OutputSequence" and expects 1 in InputSyncAck.
3) The controller is permitted to set InputSyncAck.  <b>Note:</b> Theoretically, data could already be transferred in this cycle. If InputSyncBit is set and InputSequenceCounter has been increased by 1, the values in the enabled Rx bytes must be accepted and acknowledged (see also "Communication in the input direction"). The input direction is synchronized, and the module can transmit data to the controller.

### 3.5.4.4 Transmitting and receiving

If a channel is synchronized, then the remote station is ready to receive messages from the transmitter. Before the transmitter can send data, it must first create a transmit array in order to meet Flatstream requirements.

The transmitting station must also generate a control byte for each segment created. This control byte contains information about how the subsequent part of the data being transferred should be processed. The position of the next control byte in the data stream can vary. For this reason, it must be clearly defined at all times when a new control byte is being transmitted. The first control byte is always in the first byte of the first sequence. All subsequent positions are determined recursively.

Flatstream formula for calculating the position of the next control byte:

$$\text{Position (of the next control byte)} = \text{Current position} + 1 + \text{Segment length}$$

#### Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The rest of the configuration corresponds to the default settings.

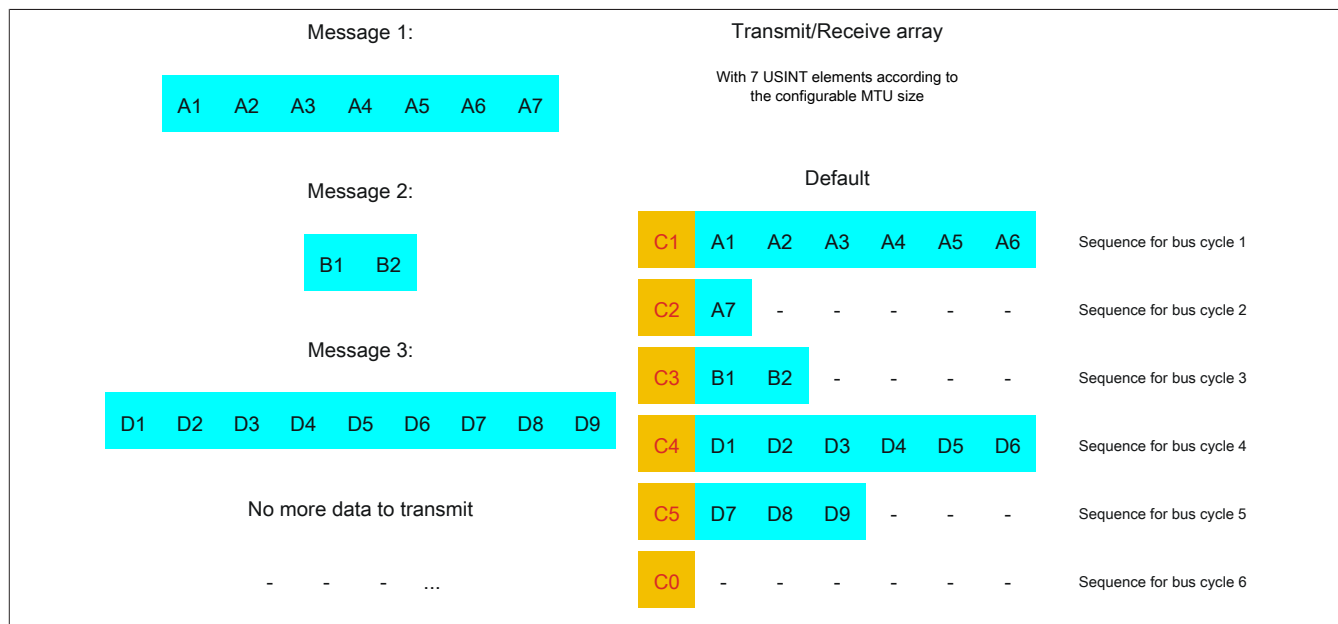


Figure 7: Transmit/Receive array (default)

## Function description

The messages must first be split into segments. In the default configuration, it is important to ensure that each sequence can hold an entire segment, including the associated control byte. The sequence is limited to the size of the enable MTU. In other words, a segment must be at least 1 byte smaller than the MTU.

MTU = 7 bytes → Max. segment length = 6 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 6 bytes of data
  - ⇒ Second segment = Control byte + 1 data byte
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 6 bytes of data
  - ⇒ Second segment = Control byte + 3 data bytes
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C0 (control byte 0)			C1 (control byte 1)			C2 (control byte 2)		
- SegmentLength (0)	=	0	- SegmentLength (6)	=	6	- SegmentLength (1)	=	1
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (0)	=	0	- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	128
Control byte	Σ	0	Control byte	Σ	6	Control byte	Σ	129

Table 3: Flatstream determination of the control bytes for the default configuration example (part 1)

C3 (control byte 3)			C4 (control byte 4)			C5 (control byte 5)		
- SegmentLength (2)	=	2	- SegmentLength (6)	=	6	- SegmentLength (3)	=	3
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (1)	=	128	- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	128
Control byte	Σ	130	Control byte	Σ	6	Control byte	Σ	131

Table 4: Flatstream determination of the control bytes for the default configuration example (part 2)

### 3.5.4.4.1 Transmitting data to a module (output)

When transmitting data, the transmit array must be generated in the application program. Sequences are then transferred one by one using Flatstream and received by the module.



#### Information:

Although all B&R modules with Flatstream communication always support the most compact transfers in the output direction, it is recommended to use the same design for the transfer arrays in both communication directions.

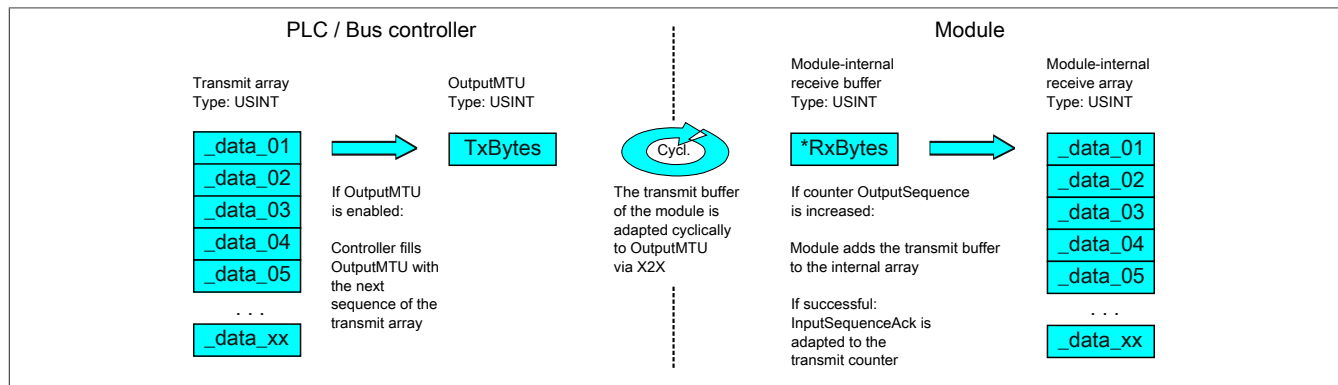


Figure 8: Flatstream communication (output)

#### Message smaller than OutputMTU

The length of the message is initially smaller than OutputMTU. In this case, one sequence would be sufficient to transfer the entire message and the necessary control byte.

#### Algorithm

Cyclic status query: - The module monitors OutputSequenceCounter.
0) Cyclic checks: - The controller must check OutputSyncAck. → If OutputSyncAck = 0: Reset OutputSyncBit and resynchronize the channel. - The controller must check whether OutputMTU is enabled. → If OutputSequenceCounter > InputSequenceAck: MTU is not enabled because the last sequence has not yet been acknowledged.
1) Preparation (create transmit array): - The controller must split up the message into valid segments and create the necessary control bytes. - The controller must add the segments and control bytes to the transmit array.
2) Transmit: - The controller transfers the current element of the transmit array to OutputMTU. → OutputMTU is transferred cyclically to the module's transmit buffer but not processed further. - The controller must increase OutputSequenceCounter.
Reaction: - The module accepts the bytes from the internal receive buffer and adds them to the internal receive array. - The module transmits acknowledgment and writes the value of OutputSequenceCounter to OutputSequenceAck.
3) Completion: - The controller must monitor OutputSequenceAck. → A sequence is only considered to have been transferred successfully if it has been acknowledged via OutputSequenceAck. In order to detect potential transfer errors in the last sequence as well, it is important to make sure that the length of the Completion phase is run through long enough.
<b>Note:</b> To monitor communication times exactly, the task cycles that have passed since the last increase of OutputSequenceCounter should be counted. In this way, the number of previous bus cycles necessary for the transfer can be measured. If the monitoring counter exceeds a predefined threshold, then the sequence can be considered lost. (The relationship of bus to task cycle can be influenced by the user so that the threshold value must be determined individually.) - Subsequent sequences are only permitted to be transmitted in the next bus cycle after the completion check has been carried out successfully.

## Function description

### Message larger than OutputMTU

The transmit array, which must be created in the program sequence, consists of several elements. The user must arrange the control and data bytes correctly and transfer the array elements one after the other. The transfer algorithm remains the same and is repeated starting at the point Cyclic checks.

### General flowchart

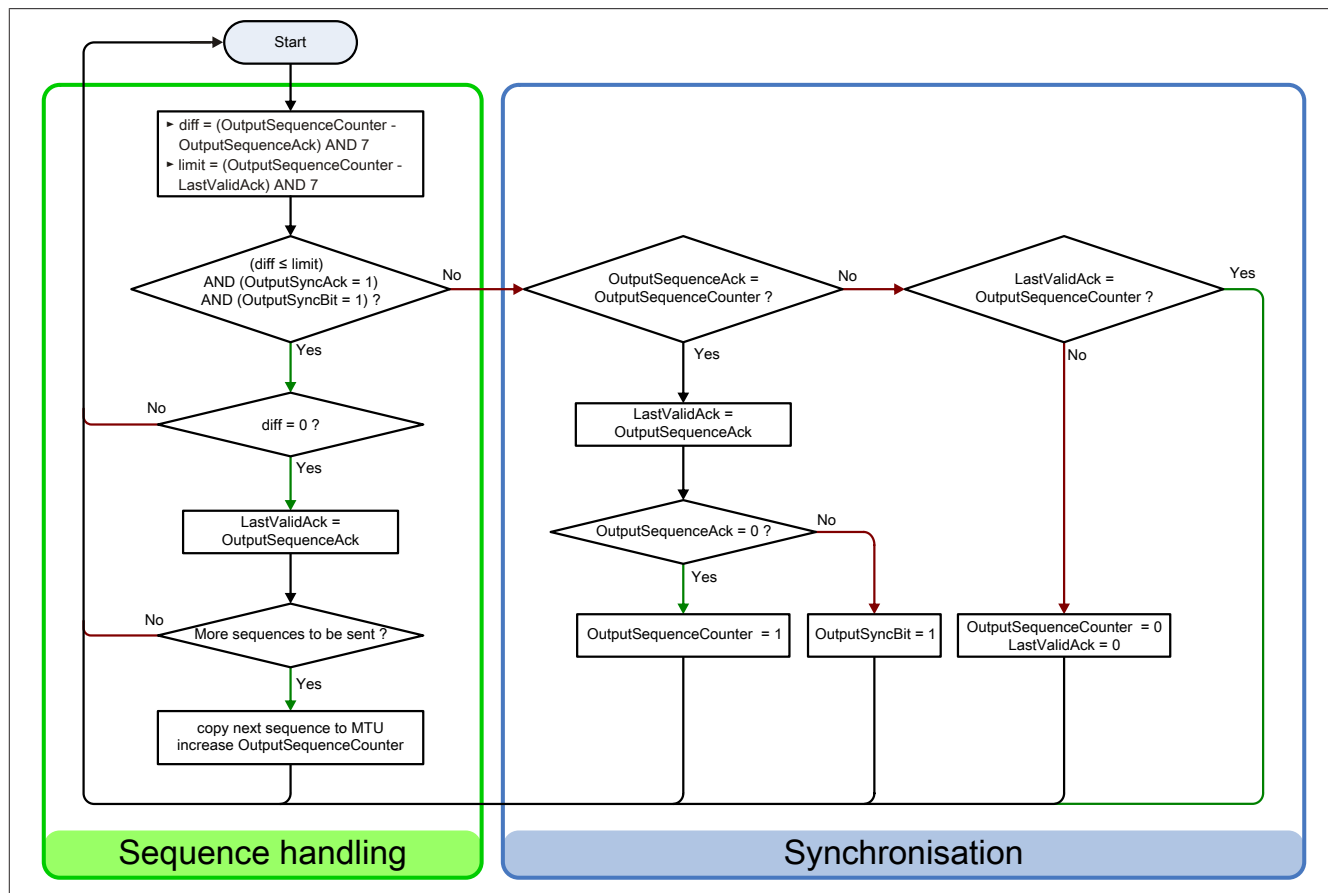


Figure 9: Flowchart for the output direction



### 3.5.4.4.2 Receiving data from a module (input)

When receiving data, the transmit array is generated by the module, transferred via Flatstream and must then be reproduced in the receive array. The structure of the incoming data stream can be set with the mode register. The algorithm for receiving the data remains unchanged in this regard.

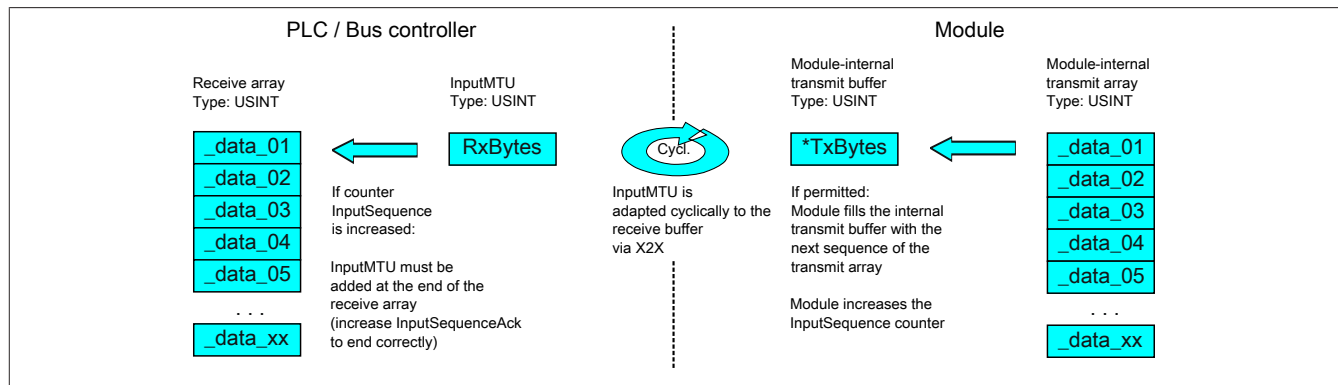
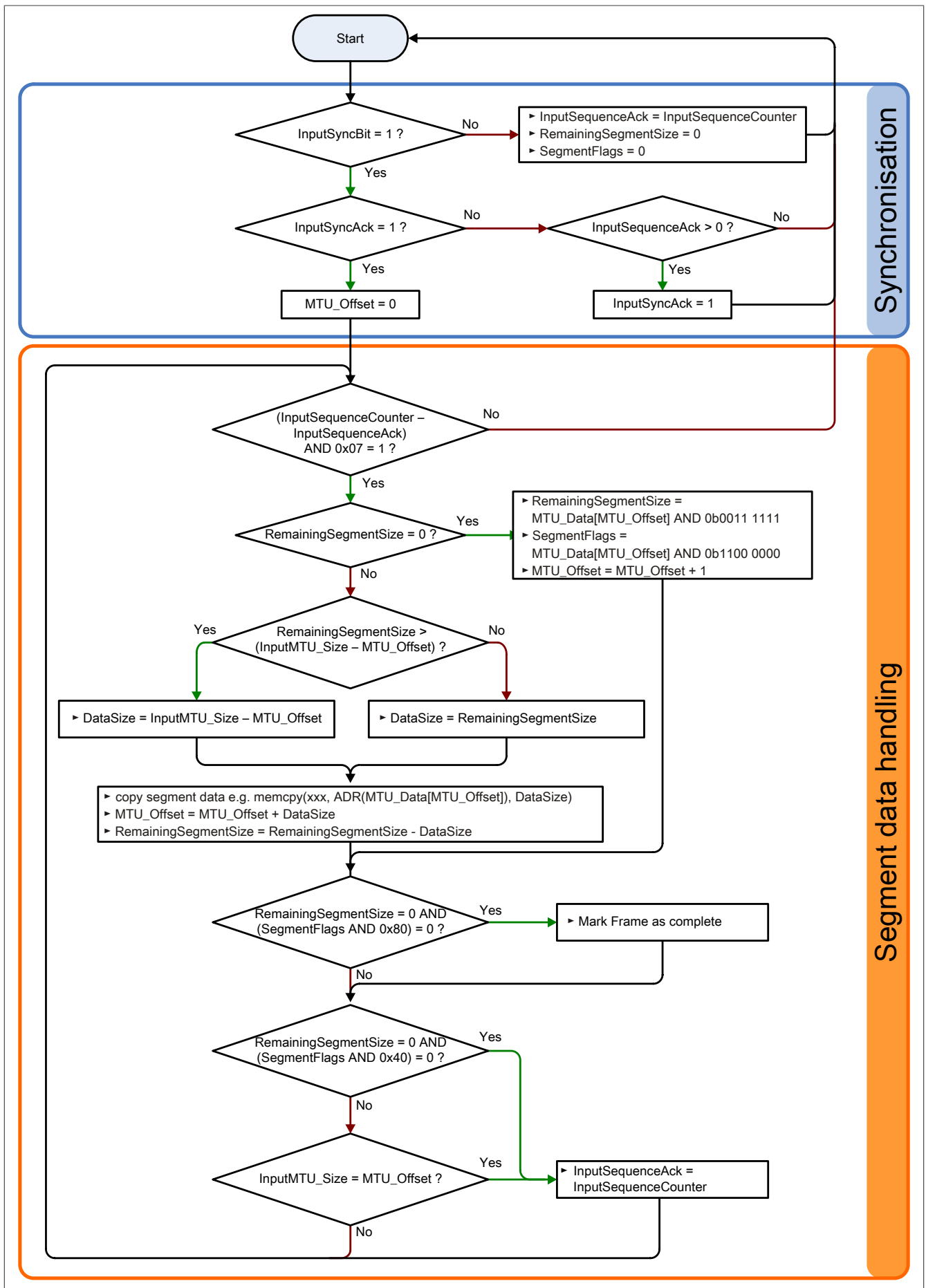


Figure 10: Flatstream communication (input)

### Algorithm

<b>0) Cyclic status query:</b> - The controller must monitor InputSequenceCounter.
Cyclic checks: - The module checks InputSyncAck. - The module checks InputSequenceAck.
Preparation: - The module forms the segments and control bytes and creates the transmit array.
Action: - The module transfers the current element of the internal transmit array to the internal transmit buffer. - The module increases InputSequenceCounter.
<b>1) Receiving (as soon as InputSequenceCounter is increased):</b> - The controller must apply data from InputMTU and append it to the end of the receive array. - The controller must match InputSequenceAck to InputSequenceCounter of the sequence currently being processed.
Completion: - The module monitors InputSequenceAck. → A sequence is only considered to have been transferred successfully if it has been acknowledged via InputSequenceAck. - Subsequent sequences are only transmitted in the next bus cycle after the completion check has been carried out successfully.



### 3.5.4.4.3 Details

**It is recommended to store transferred messages in separate receive arrays.**

After a set MessageEndBit is transmitted, the subsequent segment should be added to the receive array. The message is then complete and can be passed on internally for further processing. A new/separate array should be created for the next message.



**Information:**

When transferring with MultiSegmentMTUs, it is possible for several small messages to be part of one sequence. In the program, it is important to make sure that a sufficient number of receive arrays can be managed. The acknowledge register is only permitted to be adjusted after the entire sequence has been applied.

**If SequenceCounter is incremented by more than one counter, an error is present.**

In this case, the receiver stops. All additional incoming sequences are ignored until the transmission with the correct SequenceCounter is retried. This response prevents the transmitter from receiving any more acknowledgments for transmitted sequences. The transmitter can identify the last successfully transferred sequence from the remote station's SequenceAck and continue the transfer from this point.



**Information:**

This situation is very unlikely when operating without "Forward" functionality.

**Acknowledgments must be checked for validity.**

If the receiver has successfully accepted a sequence, it must be acknowledged. The receiver takes on the value of SequenceCounter sent along with the transmission and matches SequenceAck to it. The transmitter reads SequenceAck and registers the successful transmission. If the transmitter acknowledges a sequence that has not yet been dispatched, then the transfer must be interrupted and the channel resynchronized. The synchronization bits are reset and the current/incomplete message is discarded. It must be sent again after the channel has been resynchronized.

3.5.4.5 Flatstream mode

In the input direction, the transmit array is generated automatically. Flatstream mode offers several options to the user that allow an incoming data stream to have a more compact arrangement. These include:

- [Standard](#)
- [MultiSegmentMTU allowed](#)
- [Large segments allowed:](#)

Once enabled, the program code for evaluation must be adapted accordingly.



Information:

All B&R modules that offer Flatstream mode support options "Large segments" and "MultiSegmentMTU" in the output direction. Compact transfer must be explicitly allowed only in the input direction.

Standard

By default, both options relating to compact transfer in the input direction are disabled.

1. The module only forms segments that are at least one byte smaller than the enabled MTU. Each sequence begins with a control byte so that the data stream is clearly structured and relatively easy to evaluate.
2. Since a Flatstream message is permitted to be any length, the last segment of the message frequently does not fill up all of the MTU's space. By default, the remaining bytes during this type of transfer cycle are not used.

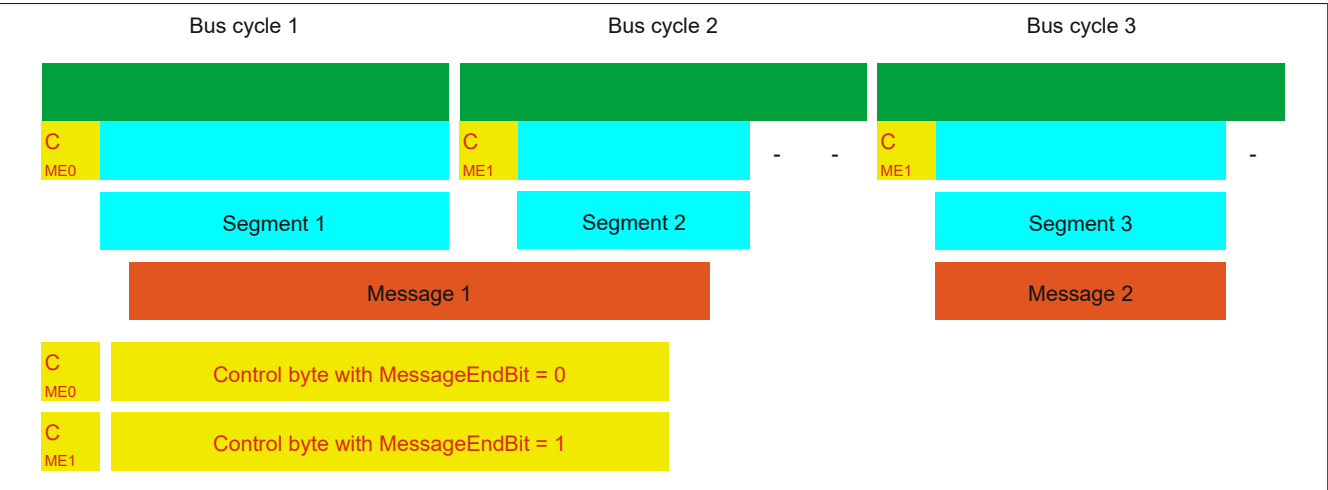


Figure 12: Message arrangement in the MTU (default)

### MultiSegmentMTU allowed

With this option, InputMTU is completely filled (if enough data is pending). The previously unfilled Rx bytes transfer the next control bytes and their segments. This allows the enabled Rx bytes to be used more efficiently.

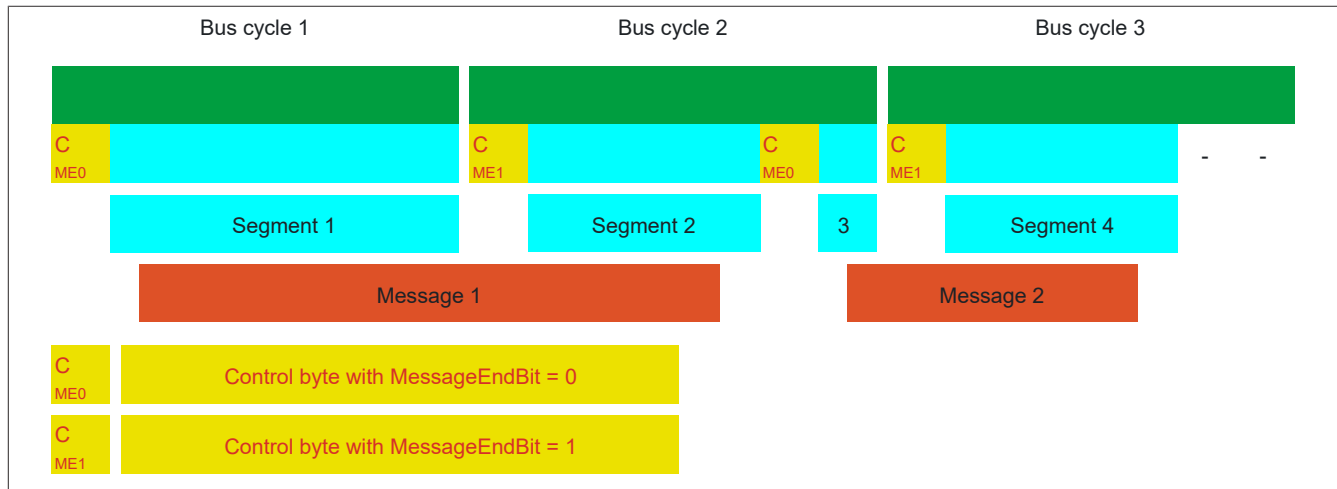


Figure 13: Arrangement of messages in the MTU (MultiSegmentMTU)

### Large segments allowed:

When transferring very long messages or when enabling only very few Rx bytes, then a great many segments must be created by default. The bus system is more stressed than necessary since an additional control byte must be created and transferred for each segment. With option "Large segments", the segment length is limited to 63 bytes independently of InputMTU. One segment is permitted to stretch across several sequences, i.e. it is possible for "pure" sequences to occur without a control byte.



#### Information:

**It is still possible to split up a message into several segments, however. If this option is used and messages with more than 63 bytes occur, for example, then messages can still be split up among several segments.**

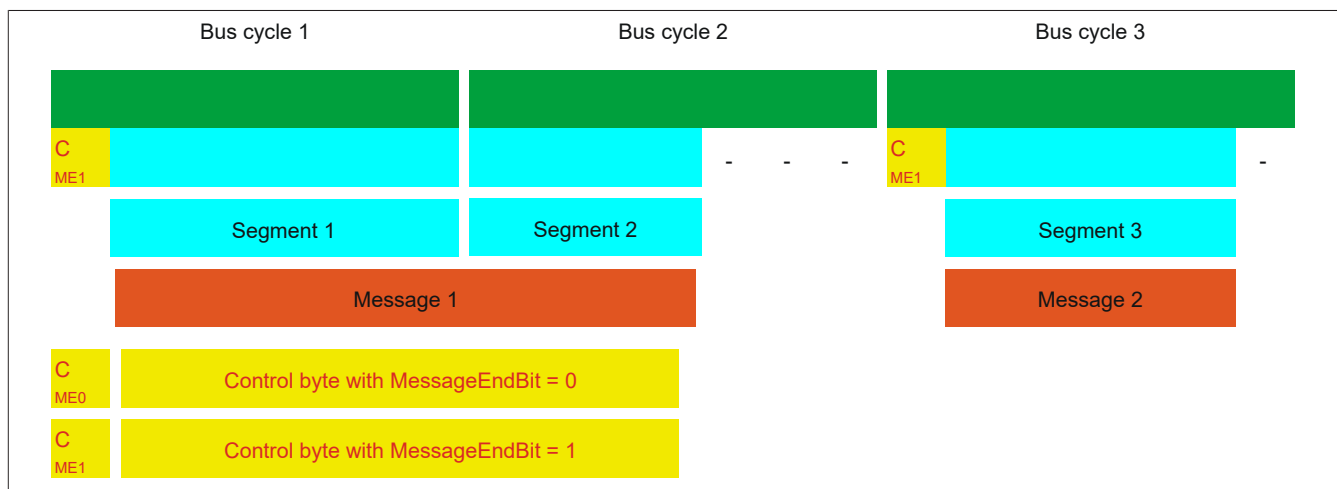


Figure 14: Arrangement of messages in the MTU (large segments)

Function description

Using both options

Using both options at the same time is also permitted.

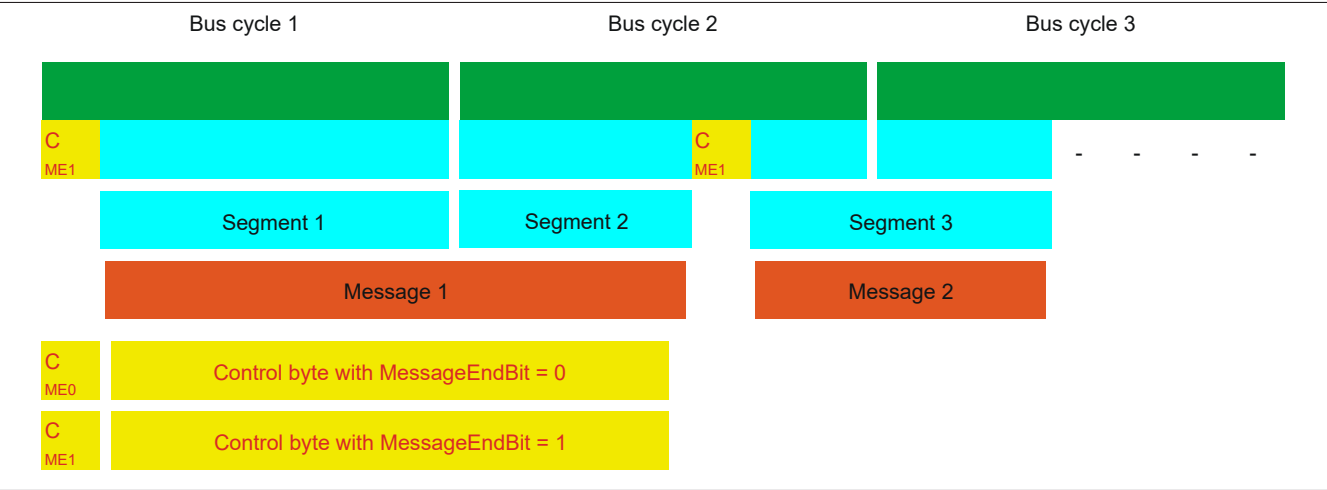


Figure 15: Arrangement of messages in the MTU (large segments and MultiSegmentMTU)

3.5.4.6 Adjusting the Flatstream

If the way messages are structured is changed, then the way data in the transmit/receive array is arranged is also different. The following changes apply to the example given earlier.

MultiSegmentMTU

If MultiSegmentMTUs are allowed, then "open positions" in an MTU can be used. These "open positions" occur if the last segment in a message does not fully use the entire MTU. MultiSegmentMTUs allow these bits to be used to transfer the subsequent control bytes and segments. In the program sequence, the "nextCB-Pos" bit in the control byte is set so that the receiver can correctly identify the next control byte.

Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The configuration allows the transfer of MultiSegmentMTUs.

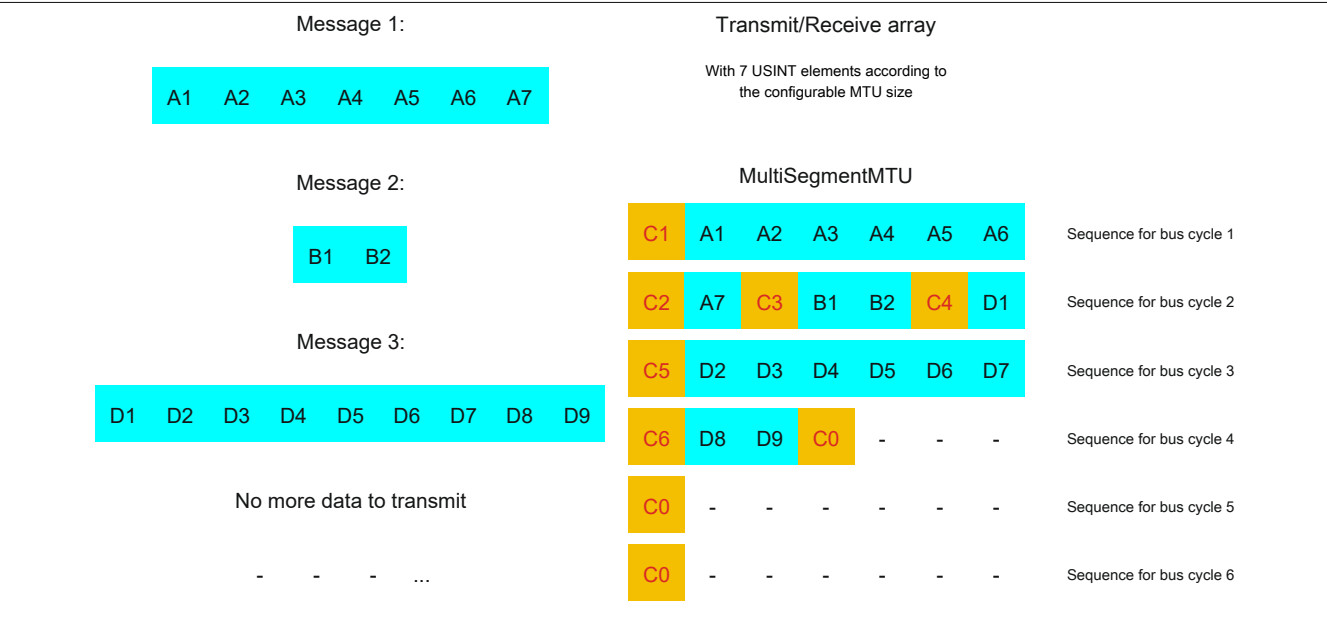


Figure 16: Transmit/Receive array (MultiSegmentMTU)

The messages must first be split into segments. As in the default configuration, it is important for each sequence to begin with a control byte. The free bits in the MTU at the end of a message are filled with data from the following message, however. With this option, the "nextCBPos" bit is always set if payload data is transferred after the control byte.

MTU = 7 bytes → Max. segment length = 6 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 6 bytes of data (MTU full)
  - ⇒ Second segment = Control byte + 1 byte of data (MTU still has 5 open bytes)
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data (MTU still has 2 open bytes)
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 1 byte of data (MTU full)
  - ⇒ Second segment = Control byte + 6 bytes of data (MTU full)
  - ⇒ Third segment = Control byte + 2 bytes of data (MTU still has 4 open bytes)
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C1 (control byte 1)			C2 (control byte 2)			C3 (control byte 3)		
- SegmentLength (6)	=	6	- SegmentLength (1)	=	1	- SegmentLength (2)	=	2
- nextCBPos (1)	=	64	- nextCBPos (1)	=	64	- nextCBPos (1)	=	64
- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128
Control byte	Σ	70	Control byte	Σ	193	Control byte	Σ	194

Table 5: Flatstream determination of the control bytes for the MultiSegmentMTU example (part 1)



### Warning!

**The second sequence is only permitted to be acknowledged via SequenceAck if it has been completely processed. In this example, there are 3 different segments within the second sequence, i.e. the program must include enough receive arrays to handle this situation.**

C4 (control byte 4)			C5 (control byte 5)			C6 (control byte 6)		
- SegmentLength (1)	=	1	- SegmentLength (6)	=	6	- SegmentLength (2)	=	2
- nextCBPos (6)	=	6	- nextCBPos (1)	=	64	- nextCBPos (1)	=	64
- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	0	- MessageEndBit (1)	=	128
Control byte	Σ	7	Control byte	Σ	70	Control byte	Σ	194

Table 6: Flatstream determination of the control bytes for the MultiSegmentMTU example (part 2)

## Function description

### Large segments

Segments are limited to a maximum of 63 bytes. This means they can be larger than the active MTU. These large segments are divided among several sequences when transferred. It is possible for sequences to be completely filled with payload data and not have a control byte.



#### Information:

**It is still possible to subdivide a message into several segments so that the size of a data packet does not also have to be limited to 63 bytes.**

### Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The configuration allows the transfer of large segments.

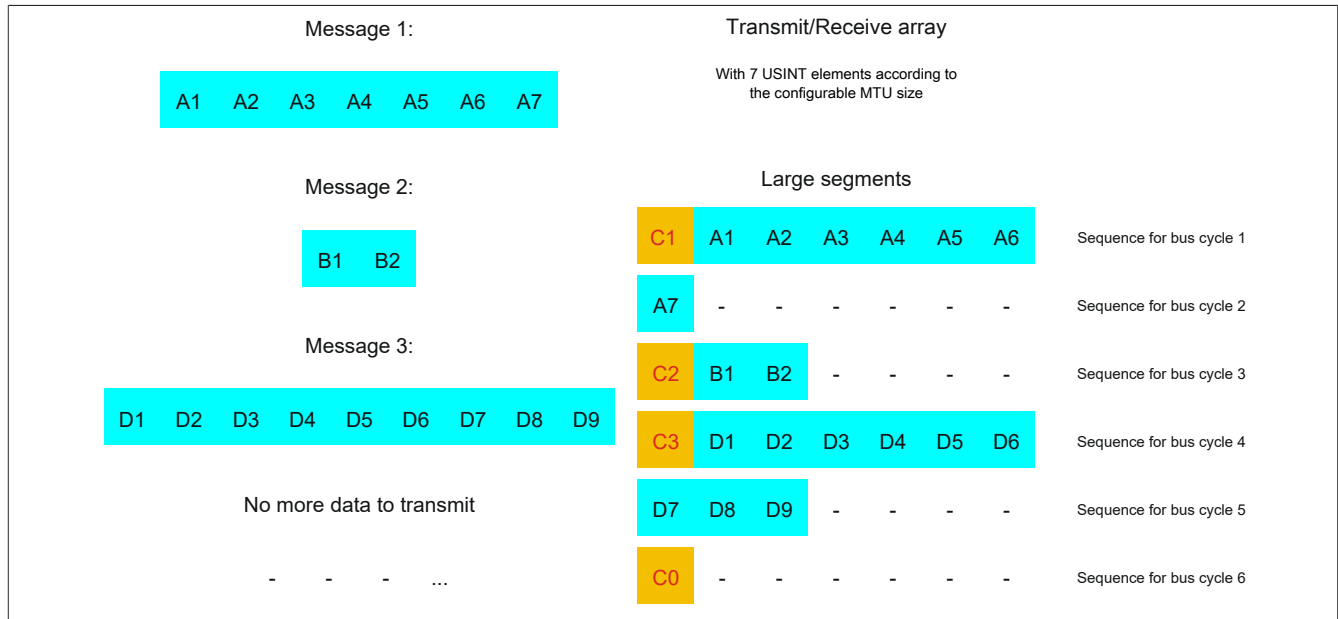


Figure 17: Transmit/receive array (large segments)

The messages must first be split into segments. The ability to form large segments means that messages are split up less frequently, which results in fewer control bytes generated.

Large segments allowed → Max. segment length = 63 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 7 bytes of data
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 9 bytes of data
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C1 (control byte 1)			C2 (control byte 2)			C3 (control byte 3)		
- SegmentLength (7)	=	7	- SegmentLength (2)	=	2	- SegmentLength (9)	=	9
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128
Control byte	Σ	135	Control byte	Σ	130	Control byte	Σ	137

Table 7: Flatstream determination of the control bytes for the large segment example



## Large segments and MultiSegmentMTU

### Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The configuration allows transfer of large segments as well as MultiSegmentMTUs.

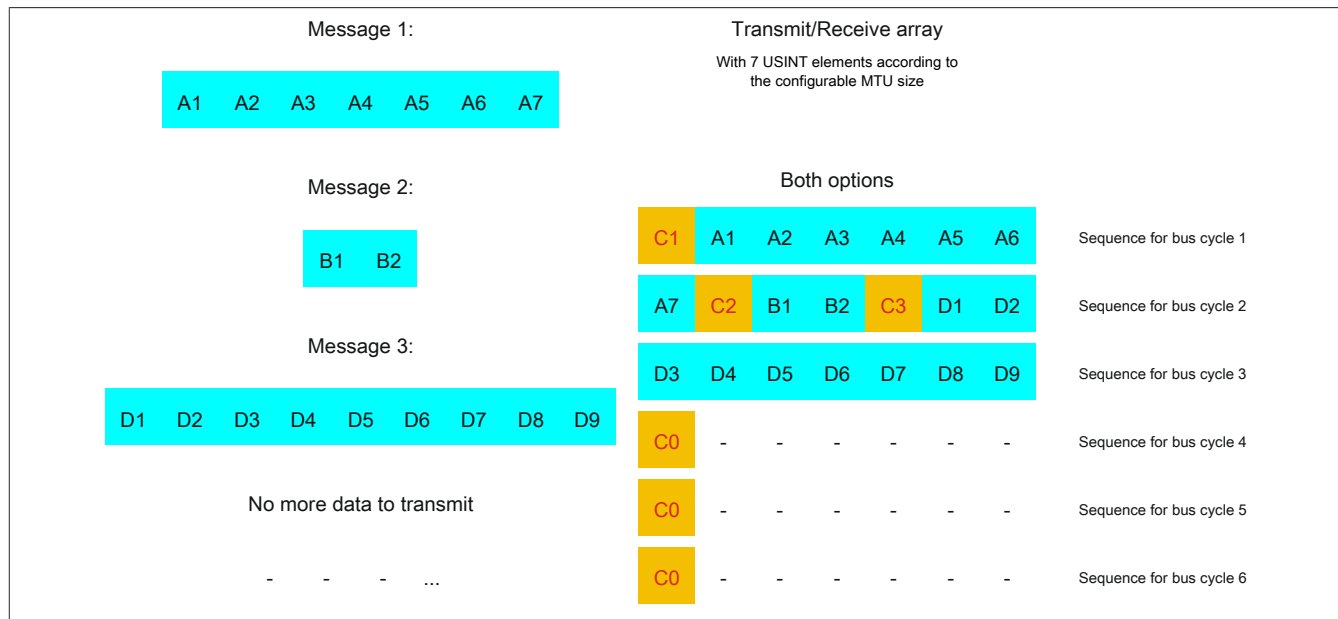


Figure 18: Transmit/Receive array (large segments and MultiSegmentMTU)

The messages must first be split into segments. If the last segment of a message does not completely fill the MTU, it is permitted to be used for other data in the data stream. Bit "nextCBPos" must always be set if the control byte belongs to a segment with payload data.

The ability to form large segments means that messages are split up less frequently, which results in fewer control bytes generated. Control bytes are generated in the same way as with option "Large segments".

Large segments allowed → Max. segment length = 63 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 7 bytes of data
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 9 bytes of data
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C1 (control byte 1)			C2 (control byte 2)			C3 (control byte 3)		
- SegmentLength (7)	=	7	- SegmentLength (2)	=	2	- SegmentLength (9)	=	9
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128
Control byte	Σ	135	Control byte	Σ	130	Control byte	Σ	137

Table 8: Flatstream determination of the control bytes for the large segment and MultiSegmentMTU example

### 3.5.5 Example of function "Forward" with X2X Link

Function "Forward" is a method that can be used to substantially increase the Flatstream data rate. The basic principle is also used in other technical areas such as "pipelining" for microprocessors.

#### 3.5.5.1 Function principle

X2X Link communication cycles through 5 different steps to transfer a Flatstream sequence. At least 5 bus cycles are therefore required to successfully transfer the sequence.

	Step I	Step II	Step III	Step IV	Step V
<b>Actions</b>	Transfer sequence from transmit array, increase Sequence-Counter	Cyclic synchronization of MTU and module buffer	Append sequence to receive array, adjust SequenceAck	Cyclic synchronization MTU and module buffer	Check SequenceAck
<b>Resource</b>	Transmitter (task to transmit)	Bus system (direction 1)	Recipients (task to receive)	Bus system (direction 2)	Transmitter (task for Ack checking)

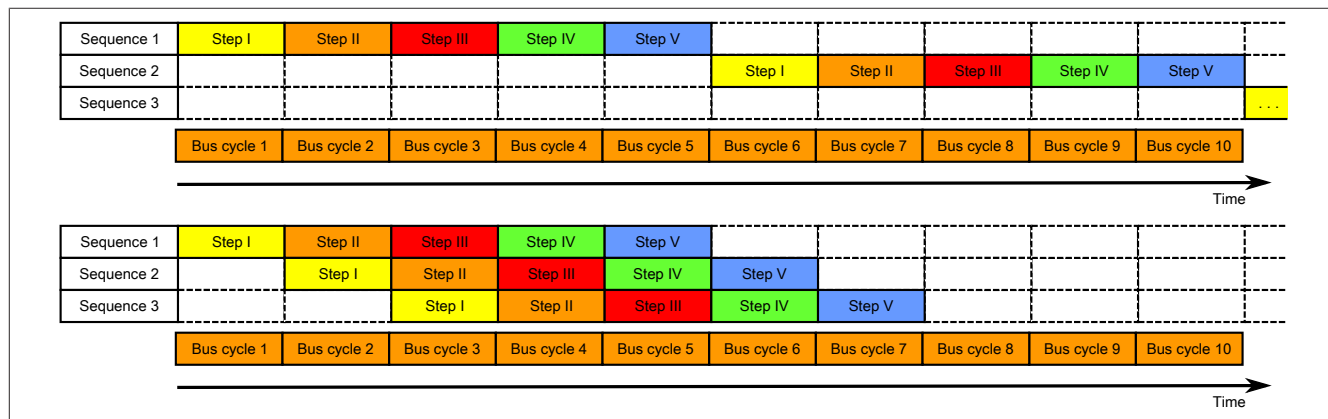


Figure 19: Comparison of transfer without/with Forward

Each of the 5 steps (tasks) requires different resources. If Forward functionality is not used, the sequences are executed one after the other. Each resource is then only active if it is needed for the current sub-action. With Forward, a resource that has executed its task can already be used for the next message. The condition for enabling the MTU is changed to allow for this. Sequences are then passed to the MTU according to the timing. The transmitting station no longer waits for an acknowledgment from SequenceAck, which means that the available bandwidth can be used much more efficiently.

In the most ideal situation, all resources are working during each bus cycle. The receiver must still acknowledge every sequence received. Only when SequenceAck has been changed and checked by the transmitter is the sequence considered as having been transferred successfully.

### 3.5.5.2 Configuration

The Forward function must only be enabled for the input direction. Flatstream modules have been optimized in such a way that they support this function. In the output direction, the Forward function can be used as soon as the size of OutputMTU is specified.



#### Information:

The registers are described in ["Flatstream registers" on page 60](#).

Registers are described in section ["Flatstream communication"](#) in the respective data sheets.

#### 3.5.5.2.1 Delay time

The delay time is specified in microseconds. This is the amount of time the module must wait after sending a sequence until it is permitted to write new data to the MTU in the following bus cycle. The program routine for receiving sequences from a module can therefore be run in a task class whose cycle time is slower than the bus cycle.

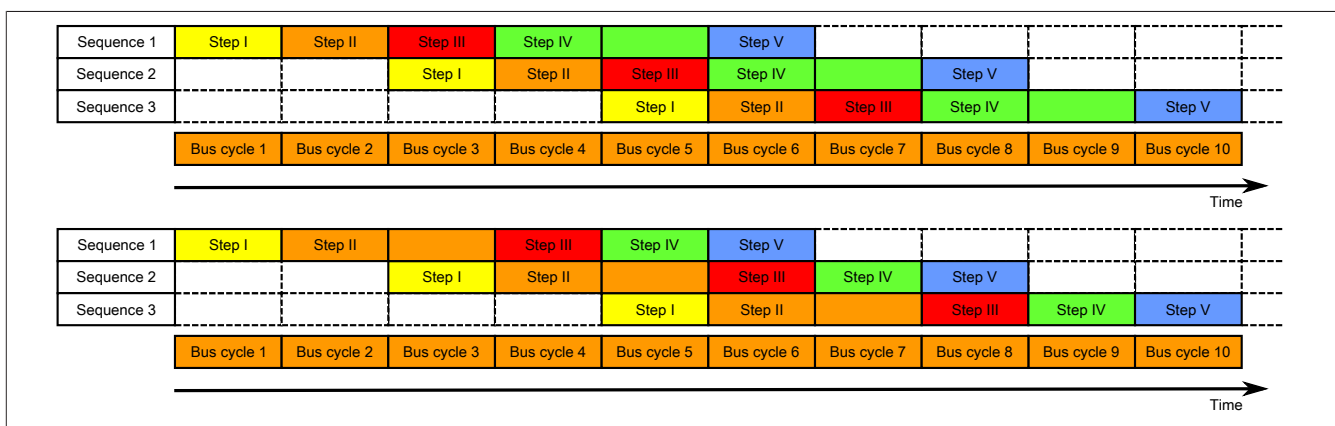


Figure 20: Effect of ForwardDelay when using Flatstream communication with the Forward function

In the program, it is important to make sure that the controller is processing all of the incoming InputSequences and InputMTUs. The ForwardDelay value causes delayed acknowledgment in the output direction and delayed reception in the input direction. In this way, the controller has more time to process the incoming InputSequence or InputMTU.

## Function description

### 3.5.5.3 Transmitting and receiving with Forward

The basic algorithm for transmitting and receiving data remains the same. With the Forward function, up to 7 unacknowledged sequences can be transmitted. Sequences can be transmitted without having to wait for the previous message to be acknowledged. Since the delay between writing and response is eliminated, a considerable amount of additional data can be transferred in the same time window.

#### Algorithm for transmitting

Cyclic status query: - The module monitors OutputSequenceCounter.
0) Cyclic checks: - The controller must check OutputSyncAck. → If OutputSyncAck = 0: Reset OutputSyncBit and resynchronize the channel. - The controller must check whether OutputMTU is enabled. → If OutputSequenceCounter > OutputSequenceAck + 7, then it is not enabled because the last sequence has not yet been acknowledged.
1) Preparation (create transmit array): - The controller must split up the message into valid segments and create the necessary control bytes. - The controller must add the segments and control bytes to the transmit array.
2) Transmit: - The controller must transfer the current part of the transmit array to OutputMTU. - The controller must increase OutputSequenceCounter for the sequence to be accepted by the module. - The controller is then permitted to transmit in the next bus cycle if the MTU has been enabled.
The module responds since OutputSequenceCounter > OutputSequenceAck: - The module accepts data from the internal receive buffer and appends it to the end of the internal receive array. - The module is acknowledged and the currently received value of OutputSequenceCounter is transferred to OutputSequenceAck. - The module queries the status cyclically again.
3) Completion (acknowledgment): - The controller must check OutputSequenceAck cyclically. → A sequence is only considered to have been transferred successfully if it has been acknowledged via OutputSequenceAck. In order to detect potential transfer errors in the last sequence as well, it is important to make sure that the algorithm is run through long enough.  <b>Note:</b> To monitor communication times exactly, the task cycles that have passed since the last increase of OutputSequenceCounter should be counted. In this way, the number of previous bus cycles necessary for the transfer can be measured. If the monitoring counter exceeds a predefined threshold, then the sequence can be considered lost (the relationship of bus to task cycle can be influenced by the user so that the threshold value must be determined individually).

#### Algorithm for receiving

0) Cyclic status query: - The controller must monitor InputSequenceCounter.
Cyclic checks: - The module checks InputSyncAck. - The module checks if InputMTU for enabling. → Enabling criteria: InputSequenceCounter > InputSequenceAck + Forward
Preparation: - The module forms the control bytes / segments and creates the transmit array.
Action: - The module transfers the current part of the transmit array to the receive buffer. - The module increases InputSequenceCounter. - The module waits for a new bus cycle after time from ForwardDelay has expired. - The module repeats the action if InputMTU is enabled.
1) Receiving (InputSequenceCounter > InputSequenceAck): - The controller must apply data from InputMTU and append it to the end of the receive array. - The controller must match InputSequenceAck to InputSequenceCounter of the sequence currently being processed.
Completion: - The module monitors InputSequenceAck. → A sequence is only considered to have been transferred successfully if it has been acknowledged via InputSequenceAck.

## Details/Background

### 1. Illegal SequenceCounter size (counter offset)

Error situation: MTU not enabled

If the difference between SequenceCounter and SequenceAck during transmission is larger than permitted, a transfer error occurs. In this case, all unacknowledged sequences must be repeated with the old SequenceCounter value.

### 2. Checking an acknowledgment

After an acknowledgment has been received, a check must verify whether the acknowledged sequence has been transmitted and had not yet been unacknowledged. If a sequence is acknowledged multiple times, a severe error occurs. The channel must be closed and resynchronized (same behavior as when not using Forward).



#### Information:

**In exceptional cases, the module can increment OutputSequenceAck by more than 1 when using Forward.**

**An error does not occur in this case. The controller is permitted to consider all sequences up to the one being acknowledged as having been transferred successfully.**

### 3. Transmit and receive arrays

The Forward function has no effect on the structure of the transmit and receive arrays. They are created and must be evaluated in the same way.

## 3.5.5.4 Errors when using Forward

In industrial environments, it is often the case that many different devices from various manufacturers are being used side by side. The electrical and/or electromagnetic properties of these technical devices can sometimes cause them to interfere with one another. These kinds of situations can be reproduced and protected against in laboratory conditions only to a certain point.

Precautions have been taken for transfer via X2X Link in case such interference should occur. For example, if an invalid checksum occurs, the I/O system will ignore the data from this bus cycle and the receiver receives the last valid data once more. With conventional (cyclic) data points, this error can often be ignored. In the following cycle, the same data point is again retrieved, adjusted and transferred.

Using Forward functionality with Flatstream communication makes this situation more complex. The receiver receives the old data again in this situation as well, i.e. the previous values for SequenceAck/SequenceCounter and the old MTU.

### Loss of acknowledgment (SequenceAck)

If a SequenceAck value is lost, then the MTU was already transferred properly. For this reason, the receiver is permitted to continue processing with the next sequence. The SequenceAck is aligned with the associated SequenceCounter and sent back to the transmitter. Checking the incoming acknowledgments shows that all sequences up to the last one acknowledged have been transferred successfully (see sequences 1 and 2 in the image).

Function description

Loss of transmission (SequenceCounter, MTU):

If a bus cycle drops out and causes the value of SequenceCounter and/or the filled MTU to be lost, then no data reaches the receiver. At this point, the transmission routine is not yet affected by the error. The time-controlled MTU is released again and can be rewritten to. The receiver receives SequenceCounter values that have been incremented several times. For the receive array to be put together correctly, the receiver is only permitted to process transmissions whose SequenceCounter has been increased by one. The incoming sequences must be ignored, i.e. the receiver stops and no longer transmits back any acknowledgments. If the maximum number of unacknowledged sequences has been sent and no acknowledgments are returned, the transmitter must repeat the affected SequenceCounter and associated MTUs (see sequence 3 and 4 in the image).

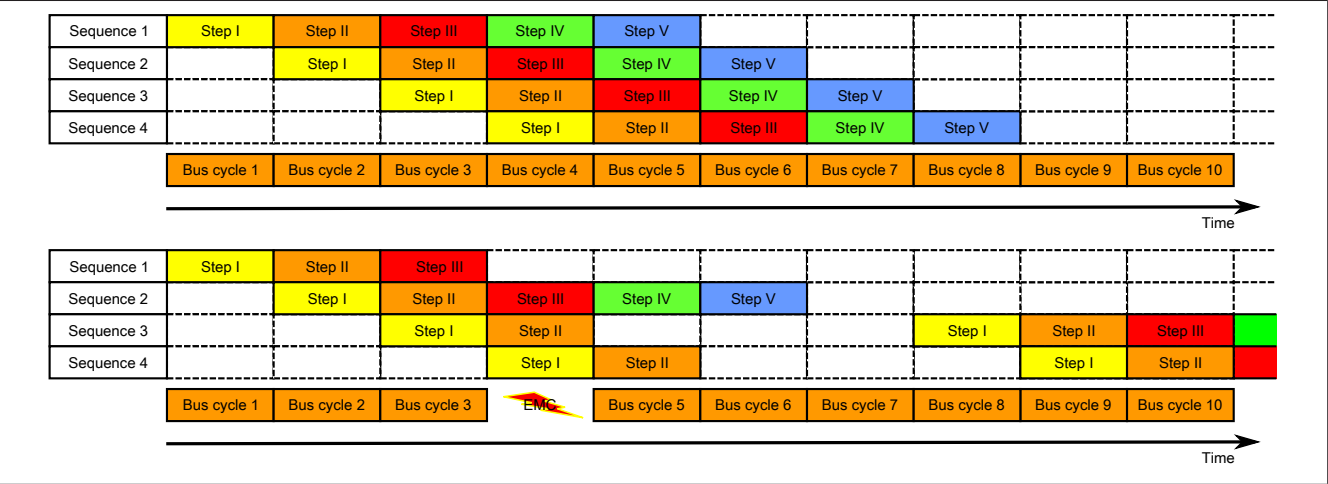


Figure 21: Effect of a lost bus cycle

Loss of acknowledgment

In sequence 1, the acknowledgment is lost due to disturbance. Sequences 1 and 2 are therefore acknowledged in Step V of sequence 2.

Loss of transmission

In sequence 3, the entire transmission is lost due to disturbance. The receiver stops and no longer sends back any acknowledgments. The transmitting station continues transmitting until it has issued the maximum permissible number of unacknowledged transmissions. 5 bus cycles later at the earliest (depending on the configuration), it begins resending the unsuccessfully sent transmissions.

## 4 Commissioning

---

### 4.1 Using the module on the bus controller

Function model 254 "Bus controller" is used by default only by non-configurable bus controllers. All other bus controllers can use other registers and functions depending on the fieldbus used.

For detailed information, see section "Additional information - Using I/O modules on the bus controller" in the X20 user's manual (version 3.50 or later).

#### 4.1.1 CAN I/O bus controller

The module occupies 2 analog logical slots on CAN I/O.

## 5 Register description

### 5.1 General data points

In addition to the registers described in the register description, the module has additional general data points. These are not module-specific but contain general information such as serial number and hardware variant.

General data points are described in section "Additional information - General data points" in the X20 System user's manual.

### 5.2 Function model 0 - Standard

Register	Name	Data type	Read		Write		
			Cyclic	Non-cyclic	Cyclic	Non-cyclic	
Analog signal - Configuration							
386	AnalogMode01	UINT				•	
394	AnalogMode02						
390	DACSlewrate01	UINT				•	
398	DACSlewrate02						
Analog signal - Communication							
0	AnalogOutput01	(U)INT			•		
2	AnalogOutput02						
30	AnalogStatus01	USINT	•				
31	AnalogStatus02						
	OpenLineAnalogOutput01 or OpenLineAnalogOutput02	Bit 2					
	ConversionErrorAnalogOutput01 or ConversionErrorAnalogOutput02	Bit 3					
	IoSuppErrorAnalogOutput01 or IoSuppErrorAnalogOutput02	Bit 7					
HART - Configuration							
1537	HartNodeCnt_1	USINT				•	
1665	HartNodeCnt_2						
1539	HartMode_1	USINT				•	
1667	HartMode_2						
1541	HartBurstNode_1	USINT				•	
1669	HartBurtNode_2						
HART - Extended configuration							
1558	HartNodeDisable_1	UINT				•	
1668	HartNodeDisable_2						
1546	HartProtTimeOut_1	UINT				•	
1674	HartProtTimeOut_2						
1550	HartProtRetry_1	UINT				•	
1678	HartProtRetry_2						
1554	HartPreamble_1	UINT				•	
1682	HartPreamble_2						
HART - Communication (P2P)							
612 + Index*24	PvInput01_N (Index N = 01 to 04)	REAL	•	• <sup>1)</sup>			
1124 + Index*24	PvInput02_N (Index N = 01 to 04)						
617 + Index*24	PvUnit01_N (Index N = 01 to 04)	USINT	•	• <sup>1)</sup>			
1129 + Index*24	PvUnit02_N (Index N = 01 to 04)						
628	PvSampleTime01	DINT	•	• <sup>1)</sup>			
1140	PvSampleTime02						
626	PvSampleTime01	INT	•				
1138	PvSampleTime02						
566	PvNodeComStatus01	UINT		•			
1078	PvNodeComStatus02						
HART - Communication (multidrop)							
612 + Index*24	PvInput01_N (Index N = 01 to 15)	REAL	•	• <sup>1)</sup>			
1124 + Index*24	PvInput02_N (Index N = 01 to 15)						
617 + Index*24	PvUnit01_N (Index N = 01 to 15)	USINT	•	• <sup>1)</sup>			
1129 + Index*24	PvUnit02_N (Index N = 01 to 15)						
604 + Index*24	PvSampleTime01_N (Index N = 01 to 15)	DINT	•	• <sup>1)</sup>			
1116 + Index*24	PvSampleTime02_N (Index N = 01 to 15)						
602 + Index*24	PvSampleTime01_N (Index N = 01 to 15)	INT	•				
1114 + Index*24	PvSampleTime02_N (Index N = 01 to 15)						
562 + Index*4	PvNodeComStatus01_N (Index N = 01 to 15)	UINT		•			
1074 + Index*4	PvNodeComStatus02_N (Index N = 01 to 15)						



Register	Name	Data type	Read		Write	
			Cyclic	Non-cyclic	Cyclic	Non-cyclic
HART - Extended communication						
522 1034	PvCountHartRequest01 PvCountHartRequest02	UINT	•			
530 1042	PvCountHartTimeout01 PvCountHartTimeout02	UINT	•			
538 1050	PvCountHartRxError01 PvCountHartRxError02	UINT	•			
546 1058	PvCountHartFrameError01 PvCountHartFrameError02	UINT	•			
554 1066	PvNodeFound01 PvNodeFound02	UINT	•			
558 1070	PvNodeError01 PvNodeError02	UINT	•			
Flatstream - Configuration						
1793	OutputMTU	USINT				•
1795	InputMTU	USINT				•
1797	FlatstreamMode	USINT				•
1799	Forward	USINT				•
1802	ForwardDelay	UINT				•
Flatstream - Communication						
1857	InputSequence	USINT	•			
1857 + Index*2	RxByteN (index N = 1 to 15)	USINT	•			
1889	OutputSequence	USINT			•	
1889 + Index*2	TxByteN (index N = 1 to 15)	USINT			•	

- 1) These HART registers are defined multiple times. Hence, they can be activated acyclically, if they are not registered during the cyclical phase of the X2X transmission.

## 5.3 Function model 1 - OSP

Register	Name	Data type	Read		Write	
			Cyclic	Non-cyclic	Cyclic	Non-cyclic
Analog signal - Configuration						
386 394	AnalogMode01 AnalogMode02	UINT				•
390 398	DACSlewrate01 DACSlewrate02	UINT				•
Analog signal - Communication						
0 2	AnalogOutput01 AnalogOutput02	(U)INT			•	
30 31	AnalogStatus01 AnalogStatus02	USINT	•			
	OpenLineAnalogOutput01 or OpenLineAnalogOutput02	Bit 2				
	ConversionErrorAnalogOutput01 or ConversionErrorAnalogOutput02	Bit 3				
	IoSuppErrorAnalogOutput01 or IoSuppErrorAnalogOutput02	Bit 7				
HART - Configuration						
1537 1665	HartNodeCnt_1 HartNodeCnt_2	USINT				•
1539 1667	HartMode_1 HartMode_2	USINT				•
1541 1669	HartBurstNode_1 HartBurtNode_2	USINT				•
HART - Extended configuration						
1558 1668	HartNodeDisable_1 HartNodeDisable_2	UINT				•
1546 1674	HartProtTimeOut_1 HartProtTimeOut_2	UINT				•
1550 1678	HartProtRetry_1 HartProtRetry_2	UINT				•
1554 1682	HartPreamble_1 HartPreamble_2	UINT				•
HART - Communication (P2P)						
612 + Index*24 1124 + Index*24	PvInput01_N (Index N = 01 to 04) PvInput02_N (Index N = 01 to 04)	REAL	•	• <sup>1)</sup>		
617 + Index*24 1129 + Index*24	PvUnit01_N (Index N = 01 to 04) PvUnit02_N (Index N = 01 to 04)	USINT	•	• <sup>1)</sup>		
628 1140	PvSampleTime01 PvSampleTime02	DINT	•	• <sup>1)</sup>		
626 1138	PvSampleTime01 PvSampleTime02	INT	•			
566 1078	PvNodeComStatus01 PvNodeComStatus02	UINT		•		

## Register description

Register	Name	Data type	Read		Write	
			Cyclic	Non-cyclic	Cyclic	Non-cyclic
HART - Communication (multidrop)						
612 + Index*24 1124 + Index*24	PvInput01_N (Index N = 01 to 15) PvInput02_N (Index N = 01 to 15)	REAL	●	● <sup>1)</sup>		
617 + Index*24 1129 + Index*24	PvUnit01_N (Index N = 01 to 15) PvUnit02_N (Index N = 01 to 15)	USINT	●	● <sup>1)</sup>		
604 + Index*24 1116 + Index*24	PvSampleTime01_N (Index N = 01 to 15) PvSampleTime02_N (Index N = 01 to 15)	DINT	●	● <sup>1)</sup>		
602 + Index*24 1114 + Index*24	PvSampleTime01_N (Index N = 01 to 15) PvSampleTime02_N (Index N = 01 to 15)	INT	●			
562 + Index*4 1074 + Index*4	PvNodeComStatus01_N (Index N = 01 to 15) PvNodeComStatus02_N (Index N = 01 to 15)	UINT		●		
HART - Extended communication						
522 1034	PvCountHartRequest01 PvCountHartRequest02	UINT	●			
530 1042	PvCountHartTimeout01 PvCountHartTimeout02	UINT	●			
538 1050	PvCountHartRxError01 PvCountHartRxError02	UINT	●			
546 1058	PvCountHartFrameError01 PvCountHartFrameError02	UINT	●			
554 1066	PvNodeFound01 PvNodeFound02	UINT	●			
558 1070	PvNodeError01 PvNodeError02	UINT	●			
Flatstream - Configuration						
1793	OutputMTU	USINT				●
1795	InputMTU	USINT				●
1797	FlatstreamMode	USINT				●
1799	Forward	USINT				●
1802	ForwardDelay	UINT				●
Flatstream - Communication						
1857	InputSequence	USINT	●			
1857 + Index*2	RxByteN (index N = 1 to 15)	USINT	●			
1889	OutputSequence	USINT			●	
1889 + Index*2	TxByteN (index N = 1 to 15)	USINT			●	
The OSP function model						
32	OSPComByte	USINT			●	
	OSPValid	Bit 0				
401 403	CfgOSPMode01 CfgOSPMode02	USINT				●
34 36	CfgOSPValue01 CfgOSPValue02	INT				●

- 1) These HART registers are defined multiple times. Hence, they can be activated acyclically, if they are not registered during the cyclical phase of the X2X transmission.

## 5.4 Function model 254 - Bus controller

Register	Offset <sup>1)</sup>	Name	Data type	Read		Write	
				Cyclic	Non-cyclic	Cyclic	Non-cyclic
Analog signal - Configuration							
386	-	AnalogMode01	UINT				•
394	-	AnalogMode02					
390	-	DACSlewrate01	UINT				•
398	-	DACSlewrate02					
Analog signal - Communication							
0	0	AnalogOutput01	(U)INT			•	
2	8	AnalogOutput02					
30	-	AnalogStatus01	USINT		•		
31	-	AnalogStatus02					
		OpenLineAnalogOutput01 or OpenLineAnalogOutput02	Bit 2				
		ConversionErrorAnalogOutput01 or ConversionErrorAnalogOutput02	Bit 3				
		IoSuppErrorAnalogOutput01 or IoSuppErrorAnalogOutput02	Bit 7				
HART - Configuration							
1537	-	HartNodeCnt_1	USINT				•
1665	-	HartNodeCnt_2					
1539	-	HartMode_1	USINT				•
1667	-	HartMode_2					
1541	-	HartBurstNode_1	USINT				•
1669	-	HartBurtNode_2					
HART - Extended configuration							
1558	-	HartNodeDisable_1	UINT				•
1668	-	HartNodeDisable_2					
1546	-	HartProtTimeOut_1	UINT				•
1674	-	HartProtTimeOut_2					
1550	-	HartProtRetry_1	UINT				•
1678	-	HartProtRetry_2					
1554	-	HartPreamble_1	UINT				•
1682	-	HartPreamble_2					
HART - Communication (P2P)							
636	4	PvInput01_01	REAL	•			
1148	12	PvInput02_01					
612 + Index*24	-	PvInput01_N (Index N = 02 to 04)	REAL		•		
1124 + Index*24	-	PvInput02_N (Index N = 02 to 04)					
641	2	PvUnit01_01	USINT	•			
1153	10	PvUnit02_01					
617 + Index*24	-	PvUnit01_N (Index N = 02 to 04)	USINT		•		
1129 + Index*24	-	PvUnit02_N (Index N = 02 to 04)					
566	-	PvNodeComStatus01	UINT		•		
1078	-	PvNodeComStatus02					
HART - Communication (multidrop)							
636	4	PvInput01_01	REAL	•			
1148	12	PvInput02_01					
612 + Index*24	-	PvInput01_N (Index N = 02 to 15)	REAL		•		
1124 + Index*24	-	PvInput02_N (Index N = 02 to 15)					
641	2	PvUnit01_01	USINT	•			
1153	10	PvUnit02_01					
617 + Index*24	-	PvUnit01_N (Index N = 02 to 15)	USINT		•		
1129 + Index*24	-	PvUnit02_N (Index N = 02 to 15)					
562 + Index*4	-	PvNodeComStatus01_N (Index N = 01 to 15)	UINT		•		
1074 + Index*4	-	PvNodeComStatus02_N (Index N = 01 to 15)					
HART - Extended communication							
522	-	PvCountHartRequest01	UINT		•		
1034	-	PvCountHartRequest02					
530	-	PvCountHartTimeout01	UINT		•		
1042	-	PvCountHartTimeout02					
538	-	PvCountHartRxError01	UINT		•		
1050	-	PvCountHartRxError02					
546	-	PvCountHartFrameError01	UINT		•		
1058	-	PvCountHartFrameError02					
554	-	PvNodeFound01	UINT		•		
1066	-	PvNodeFound02					
558	-	PvNodeError01	UINT		•		
1070	-	PvNodeError02					

1) The offset specifies the position of the register within the CAN object.

## 5.5 Analog signal - Configuration

The module is equipped with 2 independent galvanically isolated channels with integrated HART modems. Both channels can be used to output an analog signal as well as for HART communication.

### 5.5.1 Operating parameters

Name:

AnalogMode01 to AnalogMode02

These registers are used to predefine the operating parameters that the module will be using for the respective channel. Each channel must be activated and configured separately.

Data type	Values	Bus controller default setting
UINT	See the bit structure.	33

Bit structure:

Bit	Name	Value	Information
0	Channel	0	Disabled
		1	Enabled (bus controller default setting)
1	Check - D/A converter configuration/status	0	Enabled (bus controller default setting)
		1	Disabled
2 - 3	Reserved	-	
4	Scaling 0 to 20 mA (Resolution 0 to 32767)	0	Disabled
		1	Enabled
5	Scaling 4 to 20 mA (Resolution 0 to 32767)	0	Disabled
		1	Enabled (bus controller default setting)
6	Scaling 0 to 24 mA (Resolution 0 to 24000)	0	Disabled
		1	Enabled
7	Scaling 0 to 20 mA (Resolution 0 to 65535)	0	Disabled
		1	Enabled
8 - 15	Reserved	-	



#### Information:

The operating parameter registers offer the option of bypassing the cyclic check of the D/A converter configuration. To ensure reliable communication, this option should only be used if there is no HART communication on the channel.

## 5.5.2 Slew rate of the analog signal

Name:

DACSlewrate01 to DACSlewrate02

These registers limit the rate at which the analog signal is modified. This makes it possible to define a sort of upper limit frequency.

Data type	Values	Bus controller default setting
UINT	See the bit structure.	514

Bit structure:

Bit	Name	Value	Information
0 - 2	Permitted change per rate	000	1-bit
		001	2-bit
		010	4-bit (bus controller default setting)
		011	8-bit
		100	16-bit
		101	32-bit
		110	64-bit
		111	128-bit
3 - 7	Reserved	-	
8 - 11	Output rate	0000	257730 Hz
		0001	198410 Hz
		0010	152440 Hz (bus controller default setting)
		0011	131580 Hz
		0100	115740 Hz
		0101	69440 Hz
		0110	37590 Hz
		0111	25770 Hz
		1000	20160 Hz
		1001	16030 Hz
		1010	10290 Hz
		1011	8280 Hz
		1100	6900 Hz
		1101	5530 Hz
		1110	4240 Hz
		1111	3300 Hz
12 - 14	Reserved	-	
15	Slewrate enable (ramp functionality)	0	Disabled (undefined jump behavior)
		1	Enabled (defined transitions)

## 5.6 Analog signal - Communication

In order to output the desired current signal (default: 4 to 20 mA), the module must be provided with the normalized output value (default: 0 to 32767).

### 5.6.1 Output value

Name:

AnalogOutput01 to AnalogOutput02

The normalized output values are specified via these registers. Depending on the choice of scaling (see register "[AnalogMode](#)" on page 52), the range of values and data type can be adapted to the requirements of the application. After a permissible value is transferred, the module outputs the corresponding current.



#### Information:

The value "0" disables the channel status LED.

Data type	Value
INT	0 to 32767
Optional: UINT	0 to 65535

### 5.6.2 Channel status

Name:

AnalogStatus01 to AnalogStatus02

The status register gives the user feedback about whether the respective channel is functioning properly.

Data type	Value
USINT	See bit structure

Bit structure:

Bit	Name	Value	Information
0 - 1	Reserved	-	
2	OpenLineAnalogOutput01, 02	0	Line OK
		1	Open line
3	ConversionErrorAnalogOutput01, 02	0	Conversion temperature OK
		1	Conversion temperature too high
4 - 6	Reserved	-	
7	IoSuppErrorAnalogOutput01, 02	0	Module supply OK
		1	Module supply error

## 5.7 HART

### 5.7.1 HART configuration

#### 5.7.1.1 Number of HART slaves

Name:

HartNodeCnt\_1 to HartNodeCnt\_2

These registers tell the module how many HART slaves are connected to a channel.



#### Information:

If a slave is not connected to one of the HART channels, the value "0" should be defined in this register. This shortens the I/O update time and avoids superfluous error messages.

Data type	Value	Information
USINT	0	HART communication disabled for this channel
	1	Point-to-point Standard HART communication (bus controller default setting)
	2 to 15	Multidrop Number of HART slave nodes

#### 5.7.1.2 Communication behavior

Name:

HartMode\_1 to HartMode\_2

The user can use these registers to configure the communication behavior of each of the HART channels. Generally, the HART nodes are polled individually. This register can still be used to start or stop burst mode when needed.

In burst mode, a node transmits its information cyclically instead of continuously. As a result, the HART standard allows the simultaneous usage of both burst mode and polling.



#### Information:

Register "[HartBurstNode](#)" on page 55 must be configured correctly for burst queries.

Data type	Values	Bus controller default setting
UINT	See bit structure.	0

Bit structure:

Bit	Name	Value	Information
0	Slave polling mode	0	Polling mode enabled (bus controller default setting)
		1	Polling mode disabled
1	Start slave burst mode	0	No response to burst (bus controller default setting)
		1	Enables burst mode in the " <a href="#">HartBurstNode</a> " on page 55 node
2	Stop slave burst mode	0	No response to burst (bus controller default setting)
		1	Disables burst mode, if enabled
3 - 7	Reserved	-	

#### 5.7.1.3 Node number for burst mode

Name:

HartBurstNode\_1 to HartBurstNode\_2

The node numbers (short address) whose information should be queried in burst mode are entered channel by channel in the "HartBurstNode" registers. Burst mode is enabled via register "[HartMode](#)" on page 55.

Data type	Value	Information
USINT	0 to 15	Point-to-point. Bus controller default setting: 0

## 5.7.2 HART communication

### 5.7.2.1 Current value

Name:

PvInput01\_01 to PvInput01\_15

PvInput02\_01 to PvInput02\_15

These registers return the current value of the process variable that has been read.



#### Information:

These registers are of data type REAL, which means that the available bytes on the X2X Link are filled more quickly when operated cyclically. If information from several slave nodes is needed, it must be retrieved acyclically or using Flatstream .

Data type	Value	Information
REAL	IEEE745 SPF	32-bit data type with valid value
	0x7FA00000	Not a number (NaN) with invalid value

### 5.7.2.2 Unit of the measured value

Name:

PvUnit01\_01 to PvUnit01\_15

PvUnit02\_01 to PvUnit02\_15

These registers provide a HART-specific code to describe the unit of the measured value. The encoding is precisely defined in the HART specification.

Data type	Value
USINT	See description of the HART slave See HART specification

### 5.7.2.3 Timestamp

Name:

PvSampleTime01 to PvSampleTime02

PvSampleTime01\_01 to PvSampleTime01\_15

PvSampleTime02\_01 to PvSampleTime02\_15

These registers return the timestamp for when the module reads the current channel mapping. The values are provided as signed 2-byte or 4-byte values.

For additional information about NetTime and timestamps, see "[NetTime Technology](#)" on page 19.

Data type	Values	Information
INT	-32,768 to 32767	NetTime timestamp of the current input value in microseconds
DINT	-2147483648 to 2147483647	NetTime timestamp of the current input value in microseconds

This is the time at which the HART master receives the slave response. In this way, it is possible to check whether new HART information has been read in since the last X2X cycle.



#### Information:

The cycle times of a HART network are relatively long so that it is not possible to reliably determine when the measured value is retrieved with just this information.



### 5.7.2.4 Status of the process variables

Name:

PvNodeComStatus01 to PvNodeComStatus02

PvNodeComStatus01\_01 to PvNodeComStatus01\_15

PvNodeComStatus02\_01 to PvNodeComStatus02\_15

These registers provide information about whether a read value is valid. Per the HART specification, this type of status register consists of 2 parts. The "response code" is stored in the high byte; the "field device status" is stored in the low byte. This makes it possible to check the current state of a read process variable.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	Quality - Node information 2 to n	0	Digital measured value okay
		1	Measured value outside the permissible operating range
1	Quality - Node information 1	0	Digital measured value okay
		1	Measured value outside the permissible operating range
2	Limit violation	0	Parameter okay
		1	Invalid measured value(s) or encoder supply value
3	Static analog signal	0	Normal value change/fluctuation
		1	Constant analog value of Node 1 slave
4	Additional status information (only supported by a few slaves)	0	Not available
		1	Available (only using Flatstream command #48)
5	Restart	0	Normal operation
		1	Field device restarts
6	Device ID	0	Unchanged
		1	Changed
7	Device error	0	Measured value okay
		1	Doubtful measured value information
8 - 14	Response code, if relevant	x	See <a href="#">HART-specific response code</a> .
15	Error - Communication	0	Error-free communication (response code irrelevant)
		1	Faulty communication (response code relevant)

### 5.7.2.5 Counter for transmit request

Name:

PvCountHartRequest01 to PvCountHartRequest02

These registers are increased once the module is ready to transmit a message to the corresponding channel.

Data type	Values
UDINT	0 to 4,294,967,295

### 5.7.2.6 Counter for timeout

Name:

PvCountHartTimeout01 to PvCountHartTimeout02

These registers are increased if the slave exceeds the maximum permitted time before responding to the module's request.

Data type	Values
UDINT	0 to 4,294,967,295

### 5.7.2.7 Counter for communication errors on layer 1 of the OSI model

Name:

PvCountHartRxError01 to PvCountHartRxError02

These registers are increased if communication errors occur on Layer 1 of the OSI model (e.g. transmission error as per parity bit).

Data type	Values
UDINT	0 to 4,294,967,295

## Register description

### 5.7.2.8 Counter for communication errors on layer 2 of the OSI model

Name:

PvCountHartFrameError01 to PvCountHartFrameError02

These registers are increased if communication errors occur on Layer 2 of the OSI model (e.g. faulty telegram structure).

Data type	Values
UDINT	0 to 4,294,967,295

### 5.7.2.9 Detected nodes

Name:

PvNodeFound01 to PvNodeFound02

These registers provide information about which nodes were detected on which channel (slave identified successfully).

Data type	Values
UINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	Node 0 (default mode) Node 1 (multidrop mode)	0	Not detected as valid
		1	Detected as valid
1	Node 2 (multidrop mode)	0	Not detected as valid
		1	Detected as valid
...		...	
13	Node 14 (multidrop mode)	0	Not detected as valid
		1	Detected as valid
14	Node 15 (multidrop mode)	0	Not detected as valid
		1	Detected as valid
15	Reserved	-	

### 5.7.2.10 HART communication error bits

Name:

PvNodeError01 to PvNodeError02

These registers contain the HART communication error bits. These bits are set if the connection to a node has been successfully established and then this node no longer responds correctly (e.g. HART slave exceeds configured timeout or configured number of attempts).

Data type	Values
UINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	Node 0 (default mode) Node 1 (multidrop mode)	0	Detected as having no errors
		1	Detected as having errors
1	Node 2 (multidrop mode)	0	Detected as having no errors
		1	Detected as having errors
...		...	
13	Node 14 (multidrop mode)	0	Detected as having no errors
		1	Detected as having errors
14	Node 15 (multidrop mode)	0	Detected as having no errors
		1	Detected as having errors
15	Reserved	-	

### 5.7.3 Extended configuration

#### 5.7.3.1 Disconnecting HART nodes

Name:

HartNodeDisable\_1 to HartNodeDisable\_2

These registers are intended for things like maintenance. They make it possible to cut off configured HART nodes to suppress error messages for a certain period of time. During normal operation, the configured nodes must be switched active to guarantee that the procedure runs smoothly.

Data type	Values	Bus controller default setting
UINT	See bit structure.	0x3FFF

Bit structure:

Bit	Name	Value	Information
0	Node 0 (default mode)	0	Enabled (bus controller default setting)
	Node 1 (multidrop mode)	1	Disabled
1	Node 2 (multidrop mode)	0	Enabled
		1	Disabled (bus controller default setting)
...		...	
13	Node 14 (multidrop mode)	0	Enabled
		1	Disabled (bus controller default setting)
14	Node 15 (multidrop mode)	0	Enabled
		1	Disabled (bus controller default setting)
15	Reserved	-	

#### 5.7.3.2 Time period for response

Name:

HartProtTimeOut\_1 to HartProtTimeOut\_2

The time period within which a slave must react in order to give a valid response is defined in these registers.

Data type	Values [ms]	Information
UINT	0 to 65535	Bus controller default setting: 256 [ms]

#### 5.7.3.3 Request repetitions

Name:

HartProtRetry\_1 to HartProtRetry\_2

These registers determine how many times the master retries a request if it receives an invalid response or no response at all.

Data type	Value	Information
UINT	0 to 65535	Bus controller default setting: 3 attempts

#### 5.7.3.4 Length of the preamble

Name:

HartPreamble\_1 to HartPreamble\_2

The length of the preamble can be set in these registers. The preamble is used to synchronize the receiver to the transmitter. The longer the declared preamble, the less chance that a communication error will occur. Nevertheless, a useful signal is not transmitted during synchronization so the preamble should be kept as short as possible.

Data type	Value	Information
UINT	5 to 20	Bus controller default setting: 20

## 5.8 Flatstream registers

At the absolute minimum, registers "InputMTU" and "OutputMTU" must be set. All other registers are filled in with default values at the beginning and can be used immediately. These registers are used for additional options, e.g. to transfer data in a more compact way or to increase the efficiency of the general procedure.



### Information:

For detailed information about Flatstream, see ["Flatstream communication" on page 22](#).

### 5.8.1 Number of enabled Tx and Rx bytes

Name:

OutputMTU

InputMTU

These registers define the number of enabled Tx or Rx bytes and thus also the maximum size of a sequence. The user must consider that the more bytes made available also means a higher load on the bus system.

Data type	Values
USINT	See the register overview.

### 5.8.2 Transporting payload data and control bytes

Name:

TxByte1 to TxByteN

RxByte1 to RxByteN

(The value the number N is different depending on the bus controller model used.)

The Tx and Rx bytes are cyclic registers used to transport the payload data and the necessary control bytes. The number of active Tx and Rx bytes is taken from the configuration of registers ["OutputMTU"](#) and ["InputMTU"](#), respectively.

- "T" - "Transmit" → Controller transmits data to the module.
- "R" - "Receive" → Controller receives data from the module.

Data type	Values
USINT	0 to 255

### 5.8.3 Communication status of the controller

Name:

OutputSequence

This register contains information about the communication status of the controller. It is written by the controller and read by the module.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0 - 2	OutputSequenceCounter	0 - 7	Counter for the sequences issued in the output direction
3	OutputSyncBit	0	Output direction (disable)
		1	Output direction (enable)
4 - 6	InputSequenceAck	0 - 7	Mirrors InputSequenceCounter
7	InputSyncAck	0	Input direction not ready (disabled)
		1	Input direction ready (enabled)

#### OutputSequenceCounter

The OutputSequenceCounter is a continuous counter of sequences that have been issued by the controller. The controller uses OutputSequenceCounter to direct the module to accept a sequence (the output direction must be synchronized when this happens).

#### OutputSyncBit

The controller uses OutputSyncBit to attempt to synchronize the output channel.

#### InputSequenceAck

InputSequenceAck is used for acknowledgment. The value of InputSequenceCounter is mirrored if the controller has received a sequence successfully.

#### InputSyncAck

The InputSyncAck bit acknowledges the synchronization of the input channel for the module. This indicates that the controller is ready to receive data.

## Register description

### 5.8.4 Communication status of the module

Name:

InputSequence

This register contains information about the communication status of the module. It is written by the module and should only be read by the controller.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0 - 2	InputSequenceCounter	0 - 7	Counter for sequences issued in the input direction
3	InputSyncBit	0	Not ready (disabled)
		1	Ready (enabled)
4 - 6	OutputSequenceAck	0 - 7	Mirrors OutputSequenceCounter
7	OutputSyncAck	0	Not ready (disabled)
		1	Ready (enabled)

#### InputSequenceCounter

The InputSequenceCounter is a continuous counter of sequences that have been issued by the module. The module uses InputSequenceCounter to direct the controller to accept a sequence (the input direction must be synchronized when this happens).

#### InputSyncBit

The module uses InputSyncBit to attempt to synchronize the input channel.

#### OutputSequenceAck

OutputSequenceAck is used for acknowledgment. The value of OutputSequenceCounter is mirrored if the module has received a sequence successfully.

#### OutputSyncAck

The OutputSyncAck bit acknowledges the synchronization of the output channel for the controller. This indicates that the module is ready to receive data.

### 5.8.5 Flatstream mode

Name:

FlatstreamMode

A more compact arrangement can be achieved with the incoming data stream using this register.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	MultiSegmentMTU	0	Not allowed (default)
		1	Permitted
1	Large segments	0	Not allowed (default)
		1	Permitted
2 - 7	Reserved		

## 5.8.6 Number of unacknowledged sequences

Name:  
Forward

With register "Forward", the user specifies how many unacknowledged sequences the module is permitted to transmit.

Recommendation:

X2X Link:           Max. 5  
POWERLINK:       Max. 7

Data type	Values
USINT	1 to 7 Default: 1

## 5.8.7 Delay time

Name:  
ForwardDelay

This register is used to specify the delay time in microseconds.

Data type	Values
UINT	0 to 65535 [ $\mu$ s] Default: 0

## 5.9 Function model "OSP"

In function model "OSP" (Operator Set Predefined), the user defines an analog value or a digital pattern. This OSP value is output as soon as the communication between the module and master is aborted.

### 5.9.1 Activating the OSP output in the module

Name:  
OSPValid

This data point offers the possibility to start module output and request OSP operation during running operation.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	OSPValid	0	Request OSP operation (after initial start or module in Standby)
		1	Request normal operation
1 - 7	Reserved	0	

There is one OSPValid bit on the module, which is managed by the user task. It must be set when the enabled channels are started. As long as the OSPValid bit remains set in the module, the module behaves the same as the "Standard" function model.

### 5.9.2 Setting the OSP mode

Name:  
CfgOSPMode01 to CfgOSPMode02

This register essentially controls a channel's behavior when OSP is being used.

Data type	Value	Description
USINT	0	Replace with static value
	1	Retain last valid value

### 5.9.3 OSP replacement value

Name:

CfgOSPValue01 to CfgOSPValue02

This register contains the analog output value, which is output in "Replace with static value" mode during OSP operation.

Data type	Value
Corresponds to AnalogOutput0x	Corresponds to AnalogOutput0x



#### Warning!

The OSP replacement value is only applied by the module if bit "OSPValid" has been set in the module.

### 5.10 Minimum cycle time

The minimum cycle time specifies how far the bus cycle can be reduced without communication errors occurring. It is important to note that very fast cycles reduce the idle time available for handling monitoring, diagnostics and acyclic commands.

Minimum cycle time
200 µs

### 5.11 Minimum I/O update time

The minimum I/O update time specifies how far the bus cycle can be reduced so that an I/O update is performed in each cycle.

Minimum I/O update time	
Analog outputs	1 ms
Minimum I/O update time Hart Communication	
Point-to-point	500 ms
Multidrop	500 ms * number of stations