X20(c)CMR011

Data sheet
1.05 (August 2025)

Translation of the original documentation

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

Disclaimer

1618586120832-1.05

1 General information

1.1 Other applicable documents

For additional and supplementary information, see the following documents.

Other applicable documents

Document name	Title
MAX20	X20 System user's manual

1.2 Order data

Order number	Short description	Figure
	Other functions	
X20CMR011	X20 cabinet monitoring module, integrated temperature sensor, moisture sensor and accelerometer, production data acquisition, 2 inputs for external PT1000, 2 digital inputs 24 V, 1 digital output 24 V, 0.5 A, 512 kB flash memory for user data	
X20cCMR011	X20 cabinet monitoring module, coated, integrated temperature sensor, moisture sensor and accelerometer, production data acquisition, 2 inputs for external Pt1000, 2 digital inputs 24 V, 1 digital output 24 V, 0.5 A, 512 kB flash memory for user data	
	Required accessories	
	Bus modules	
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	
	Terminal blocks	
X20TB12	X20 terminal block, 12-pin, 24 VDC keyed	

Table 1: X20CMR011, X20cCMR011 - Order data

1.3 Module description

The module is designed for measuring ambient conditions in the control cabinet as well as recording operating hours and power-on cycles. In addition, the module offers the option of storing user data directly on the module, is equipped with inputs/outputs and supports blackout mode.

Functions:

- Measuring and evaluating ambient conditions
- Recording operating data
- Internal module memory for user data
- Configuration of inputs
- "Blackout mode"
- "Flatstream communication"

Measuring and evaluating ambient conditions

The ambient conditions are continuously evaluated by the module. The duration in which individual parameters are within certain ranges is stored internally. This makes it possible, for example, to determine how long the system remained in a certain temperature range. The histograms recorded by the module can be read out by the user.

Internal module memory for user data

Data from the application can be saved directly on the module and also read back from the module via 512 kB nonvolatile user memory (flash memory). The data is therefore retained after the module or controller is restarted and remains with the module in the event that the module is connected to another machine or system, for example. Data retention is maintenance-free – without batteries.

Information:

It is important to note that the internal module memory is not available in function model "Bus controller"!

Inputs/Outputs

The module is equipped with 2 inputs for PT1000 temperature sensors, 2 digital inputs (24 VDC) and 1 digital output (24 VDC).

Blackout mode

The integrated blackout mode ensures that module functionality is maintained even in the event of network failure.

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	X20CMR011	X20cCMR011
Short description		
I/O module	Measurement of ambient conditions: Temperature in the module, relative humidity, vibration, shock, position, operating hours, power-on cycles 2 digital inputs, 1 digital output, 2 PT1000 temperature inputs	Measurement of ambient conditions: Temperature in the module, relative humidity, vibration, shock, position, operating hours, power-on cycles 2 digital inputs, 1 digital output, 2 Pt1000 temperature inputs
General information		
B&R ID code	0x28B4	0x2A02
Status indicators	I/O function by channel, overload, memory access, operating state, module status	I/O function per channel, overload, memory access, operating state, module status
Diagnostics		
Module run/error	Yes, using LED status indicator and software	
Output	Yes, using LED status indicator and software (output error status)	
Blackout mode		
Scope	Module	
Function	Module functionality	
Standalone mode	No	
Power consumption		
Bus	0.4 W	
Internal I/O	0.1 W	
Additional power dissipation caused by actuators (resistive) [W]	+0.075	
Application memory		
Type	512 kB flash memory	
Sectors	8 sectors, 64 kB each	
Data retention	20 years at 55°C	
Guaranteed erase/write cycles	100,000 per sector	
Error-correcting code (ECC)	No	
Write protection	No	
Certifications		
CE	Yes	
UKCA	Yes	
DNV	Temperature: **B** (0 to 55°C) Humidity: **B** (up to 100%) Vibration: **B** (4 g) EMC: **B** (bridge and open deck)	
CCS	Yes	-
LR	ENV1	
KR	Yes	
ABS	Yes	
Digital inputs		
Quantity	2	
Nominal voltage	24 VDC	
Input voltage	24 VDC -15% / +20%	
Input current at 24 VDC	Typ. 3.75 mA	
Input circuit	Sink	
Input filter		
Hardware	≤100 µs	
Software	Default 1 ms, configurable between 0 and 25 ms in 0.2 ms increments	
Connection type	1-wire connections	
Input resistance	Typ. 6.4 kΩ	
Switching threshold		
Low	<5 VDC	
High	>15 VDC	
Insulation voltage between channel and bus	500 V_{eff}	
Resistance measurement temperature inputs		
Quantity	2	
Input	Resistance measurement for 2-wire connections	
Digital converter resolution	12-bit	
Filter time	Configurable	
Conversion time	1 ms for all inputs	
Output format	INT	

Table 2: X20CMR011, X20cCMR011 - Technical data

Technical description

Order number	X20CMR011	X20cCMR011
Sensor		
Pt1000	-40 to 125°C	
Input filter	Averaging with clipping function	
Linearization method	Internal	
Permissible input signal	Short-term ±15 V	
Max. gain drift	0.0003 %/°C	
Max. offset drift	0.06 %/°C	
Nonlinearity	<0.010%	
Temperature sensor resolution		
Pt1000	1 LSB = 0.1°C	
Temperature measurement monitoring		
Range undershoot	0x8001	
Range overshoot	0x7FFF	
Open circuit	0x7FFF	
General fault	0x8000	
Open inputs	0x7FFF	
Temperature and humidity sensor		
Sensor position	Module-internal	
Sampling rate	1 s	
Temperature measurement		
Measurement range	-25 to 125°C	
Resolution	0.1°C/LSB	
Max. error	±0.3°C	
Humidity measurement		
Measurement range	5 to 95%	
Resolution	1%/LSB	
Max. error	±2% at 10 to 80% relative humidity ±3% at <10 and >80% relative humidity	
Acceleration and angle sensor		
Sensor position	Module-internal	
Sampling rate	Typ. 10 ms	
Acceleration measurement		
Measurement range	±16 g	
Resolution	0.488 mg/LSB	
Linearity over temperature	±1% at -40 to 85°C	
Offset calibration error	±40 mg	
Angular velocity measurement		
Measurement range	±2000 dps [1)]	
Resolution	±70 mdps per LSB	
Linearity over temperature	±1.5% at -40 to 85°C	
Offset calibration error	±10 dps	
Digital outputs		
Quantity	1	
Variant	Current-sourcing FET	
Nominal voltage	24 VDC	
Switching voltage	24 VDC -15% / +20%	
Nominal output current	0.5 A	
Connection type	1- or 2-wire connections	
Output circuit	Source	
Diagnostic status	Output monitoring with 10 ms delay	
Leakage current when the output is switched off	120 µA	
$R_{DS(on)}$	300 mΩ	
Switching delay		
0 → 1	<300 µs	
1 → 0	<300 µs	
Switching frequency		
Resistive load	Max. 100 Hz	
Inductive load	See section "Switching inductive loads".	
Insulation voltage between channel and bus	500 V_{eff}	
Protective measures		
Short-circuit proof	Yes	
Overload-proof	Temporary overload	
Electrical properties		
Electrical isolation	Analog channel not isolated from bus Digital channel isolated from bus Digital channel not isolated from digital channel	
Operating conditions		
Mounting orientation		
Horizontal	Yes	
Vertical	Yes	

Table 2: X20CMR011, X20cCMR011 - Technical data

Order number	X20CMR011	X20cCMR011
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	-	
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	$12.5^{+0.2}$ mm	

Table 2: X20CMR011, X20cCMR011 - Technical data

1) dps: Degrees per second (°/s)

2.2 LED status indicators

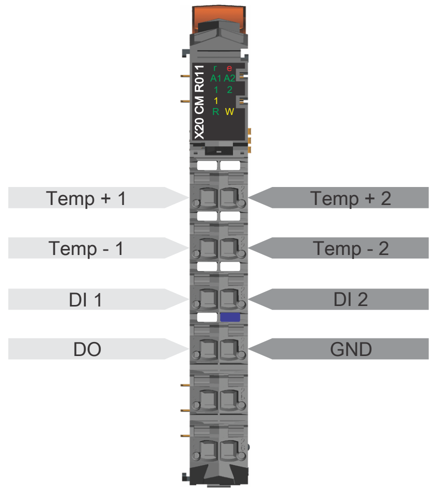
For a description of the various operating modes, see section "Additional information - Diagnostic LEDs" in the X20 system user's manual.

Figure	LED	Color	Status	Description
	r	Green	Off	No power to module
			Single flash	Mode RESET
			Double flash	Blackout mode active
			Blinking	Mode PREOPERATIONAL
			On	Mode RUN
	e	Red	Off	Module not supplied with power or everything OK
	e + r	Solid red / Single green flash		Invalid firmware
	A1 - A2	Green	Off	The channel is switched off.
			Blinking	Open circuit, overflow or underflow of the input signal
			On	Analog/Digital converter running, value OK
	1 - 2	Green		Input state of the corresponding digital input
	1	Yellow		Output state of the corresponding digital output
	R	Green	Off	No data is being read from internal memory.
			On	The module is reading data from internal memory.
	W	Yellow	Off	No data is being written to internal memory.
			On	The module is writing data to internal memory.

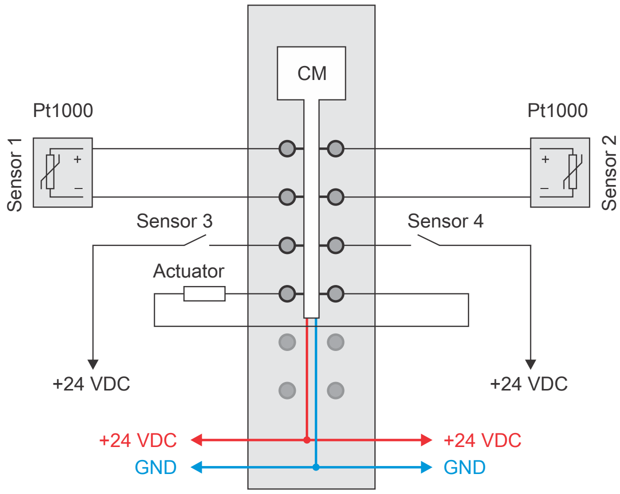
2.3 Pinout

Information:

The maximum cable length for the external temperature sensor is 20 m.

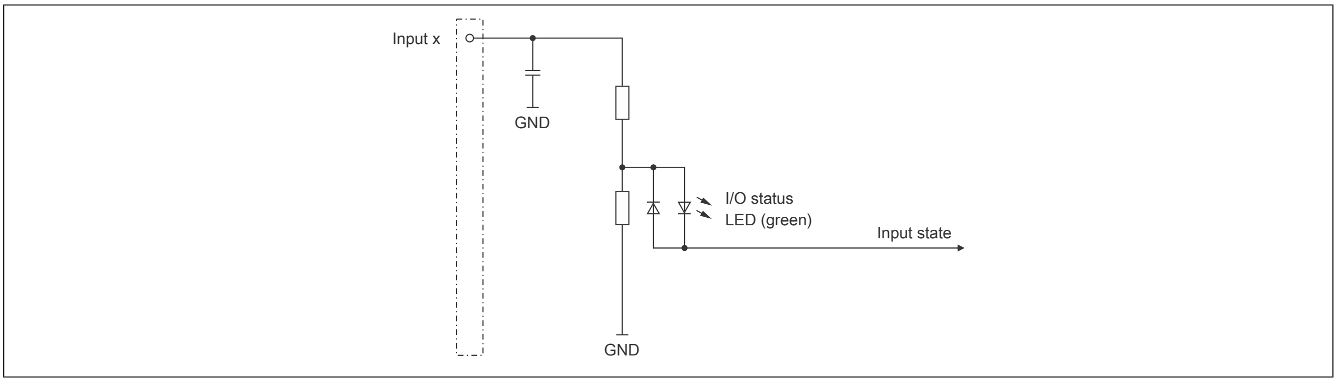


2.4 Connection example

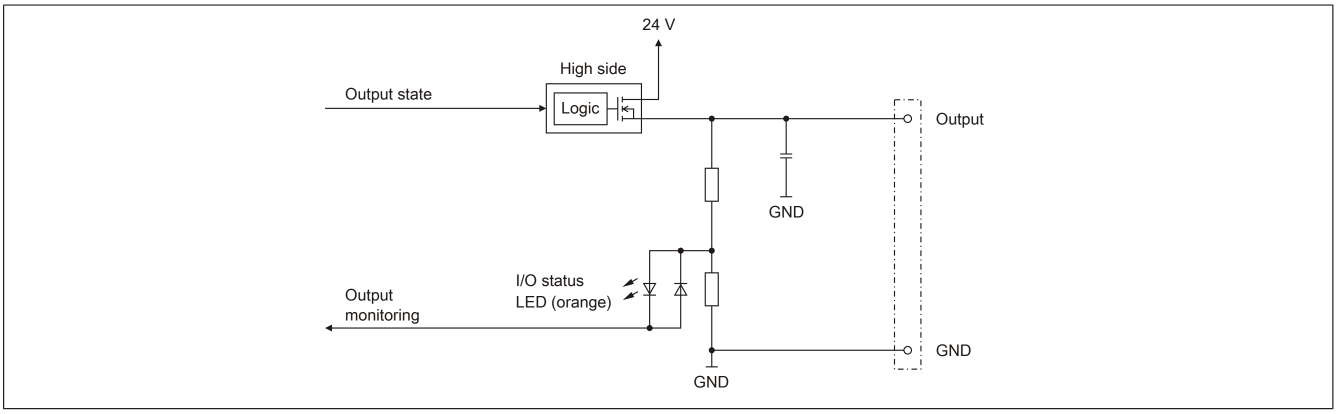


2.5 Input/Output circuit diagram

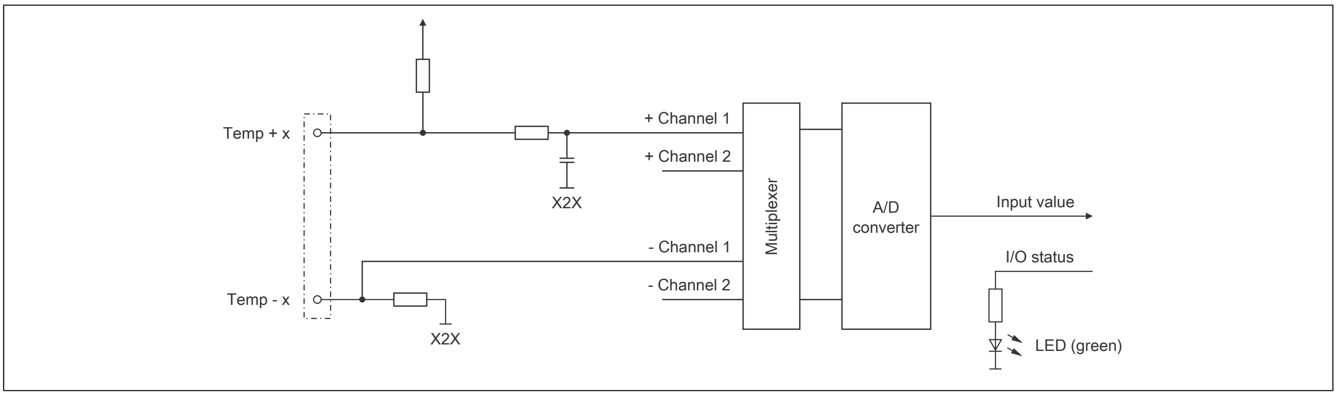
2.5.1 Digital inputs



2.5.2 Digital output

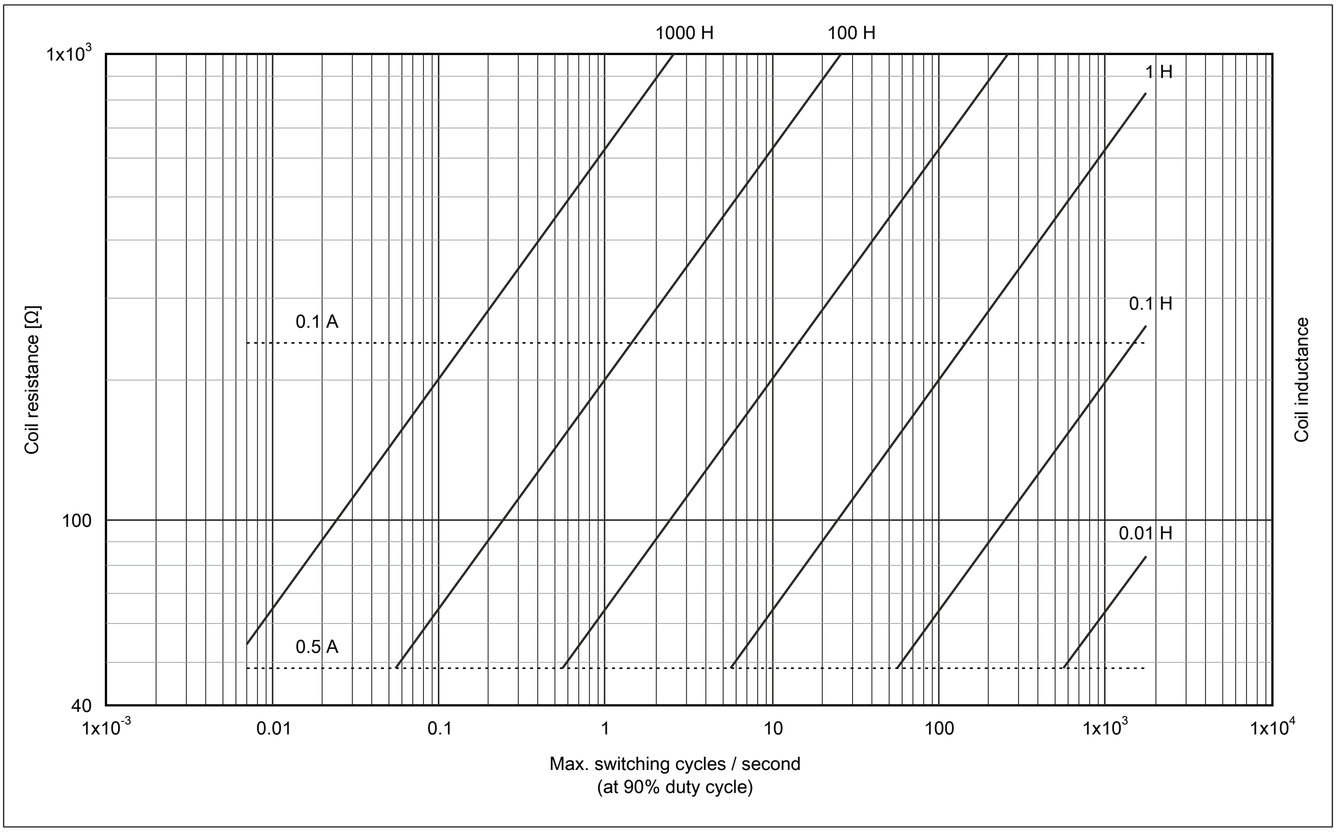


2.5.3 PT1000 temperature inputs



2.6 Switching inductive loads

Ambient temperature: 60°C



Information:

If the maximum number of operating cycles per second is exceeded, an external inverse diode must be used.

Operating conditions outside of the area in the diagram are not permitted!

3 Function description

3.1 Measuring and evaluating ambient conditions

The module is equipped with internal sensors to collect the following conditions:

Condition	Unit	Sampling rate
Relative humidity	[%]	1 s
Ambient temperature	[°C]	1 s
Acceleration	[g]	Typ. 10 ms
Rotation	[dps]	Typ. 10 ms

3.1.1 Relative humidity and ambient temperature

Since the sensor for relative humidity and ambient temperature is located directly in the module, the measured values depend on the intrinsic heating of the module and the heat radiated by neighboring modules.

The effect of this warming on the measured values can be circumvented by connecting an external temperature sensor on one of the temperature sensors on the module. The value measured with the external temperature sensor is used as a reference. With this value, the relative humidity at the position of the external temperature sensor is calculated using the Magnus formula.

$$\text{saturated water vapor pressure [Pa]} = 611.2 \cdot e^{\frac{17.62 \cdot \text{temperature}}{243.12 + \text{temperature}}}$$

$$\text{absolute humidity [g/m}^3\text{]} = \frac{\text{saturated water vapor pressure}}{461.52 \cdot (273.15 + \text{temperature})} \cdot 1000$$

$$\text{humidity [g/m}^3\text{]} = \text{absolute humidity} \cdot \text{relative humidity}$$

$$\text{relative humidity [\%]} = \frac{\text{humidity}}{\text{absolute humidity}} \cdot 100$$

Example

The following example calculates the relative humidity at the position of the external temperature sensor using the Magnus formula.

- Relative humidity in module: 20%
- Ambient temperature in module: 40°C
- External temperature sensor: 35°C

<u>Module</u>

$$\text{saturated water vapor pressure}_{\text{module}} = 611.2 \cdot e^{\frac{17.62 \cdot 40}{243.12 + 40}} = 7367.5 \text{ Pa}$$

$$\text{absolute humidity}_{\text{module}} = \frac{7367.5}{461.52 \cdot (273.15 + 40)} \cdot 1000 = 50.98 \text{ g/m}^3$$

$$\text{humidity}_{\text{module}} = 50.98 \cdot 0.2 = 10.2 \text{ g/m}^3$$

<u>External temperature sensor</u>

$$\text{saturated water vapor pressure}_{\text{ExtSensor}} = 611.2 \cdot e^{\frac{17.62 \cdot 35}{243.12 + 35}} = 5612.8 \text{ Pa}$$

$$\text{absolute humidity}_{\text{ExtSensor}} = \frac{5612.8}{461.52 \cdot (273.15 + 35)} \cdot 1000 = 39.47 \text{ g/m}^3$$

$$\text{relative humidity}_{\text{ExtSensor}} = \frac{10.2}{39.47} \cdot 100 = 25.84\%$$

In this example, a deviation of the relative humidity of approx. 6% results between the measured value in the module and the calculated value at the position of the external temperature sensor.

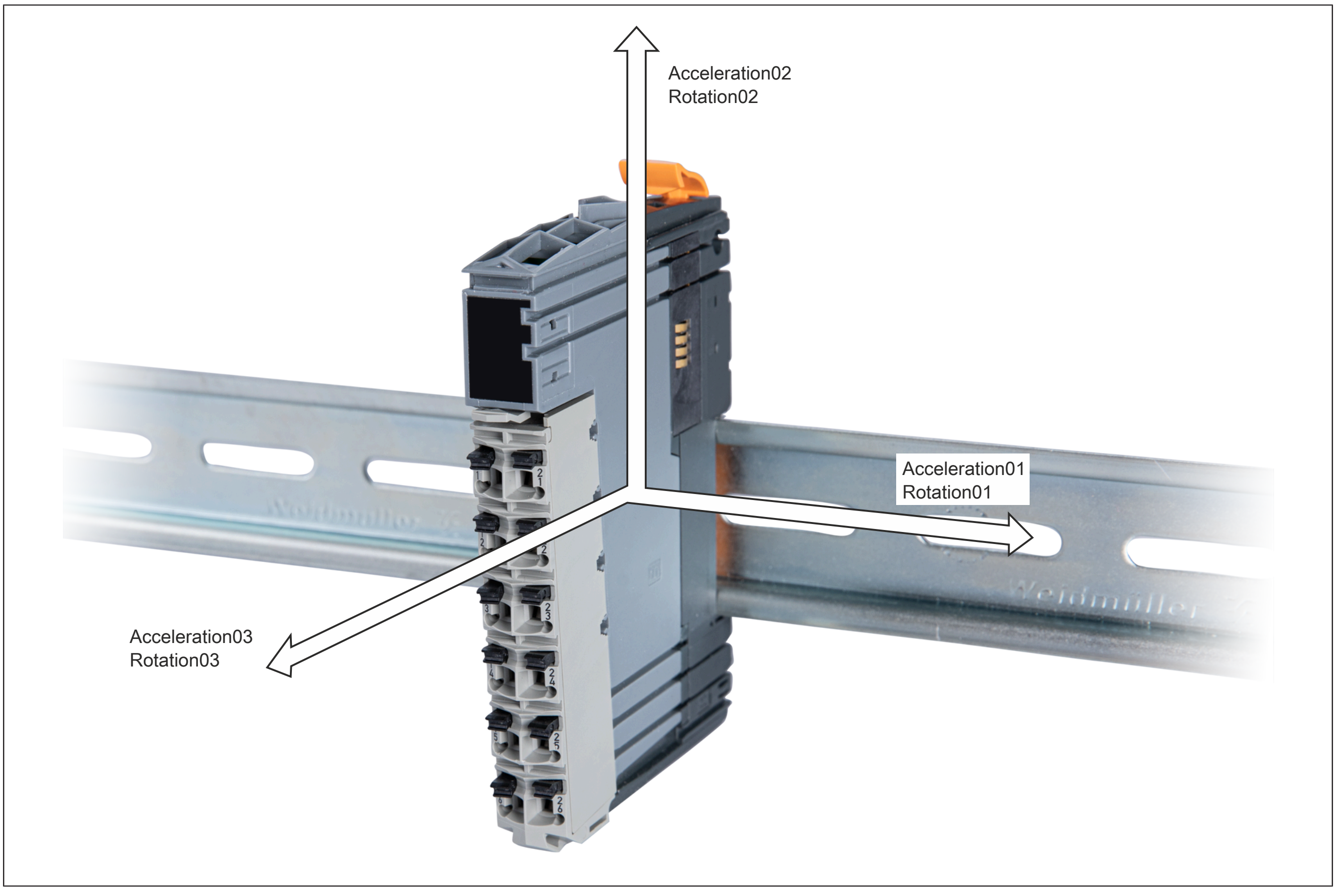
3.1.2 Acceleration and rotation

Acceleration and rotation measured by the internal sensor are returned as raw values. The conversion must be performed in the application.

Measured size	Conversion
Acceleration	16 g = 32767 -16 g = -32768
Rotation	2000 dps = 32767 -2000 dps = -32768

Assigning the axes

The sensor determines the acceleration and rotation values of all 3 axes. The following image shows the assignment of axes in relation to the module installed on the top-hat rail.



3.1.3 Additional information

The ambient conditions are recorded and evaluated in the module. The following values can be read:

- Smallest value occurred
- Largest value occurred

Information:

The values are saved to module-internal FRAM.

If needed, the values can be reset.

Registers are described in section **"Additional information" on page 54****.**

3.1.4 Histogram for relative humidity

A histogram for relative humidity is recorded in the module. The measuring range for the relative humidity is divided into 10 areas:

Area	Relative humidity	Register
1	0 to <10%	RelHumHist01Entry RelHumHist01Time
2	10 to <20%	RelHumHist02Entry RelHumHist02Time
3	20 to <30%	RelHumHist03Entry RelHumHist03Time
4	30 to <40%	RelHumHist04Entry RelHumHist04Time
5	40 to <50%	RelHumHist05Entry RelHumHist05Time
6	50 to <60%	RelHumHist06Entry RelHumHist06Time
7	60 to <70%	RelHumHist07Entry RelHumHist07Time
8	70 to <80%	RelHumHist08Entry RelHumHist08Time
9	80 to <90%	RelHumHist09Entry RelHumHist09Time
10	90 to 100%	RelHumHist10Entry RelHumHist10Time

As soon as the relative humidity reaches one of the predefined areas, a delay time of 3 s begins. After the delay time has expired, the entry counter is increased by 1 and the dwell time begins. The delay time prevents the counter from constantly being incremented in the crossover area.

Information:

The values are saved to module-internal FRAM.

If needed, the registers can be reset.

Registers are described in section **"Relative humidity" on page 56****.**

3.1.5 Histogram for ambient temperature

A histogram for ambient temperature is recorded in the module. The measuring range for the ambient temperature is divided into 12 areas:

Area	Ambient temperature	Register
1	<-20°C	TempHist01Entry TempHist01Time
2	-20 to <-10°C	TempHist02Entry TempHist02Time
3	-10 to <0°C	TempHist03Entry TempHist03Time
4	0 to <10°C	TempHist04Entry TempHist04Time
5	10 to <20°C	TempHist05Entry TempHist05Time
6	20 to <30°C	TempHist06Entry TempHist06Time
7	30 to <40°C	TempHist07Entry TempHist07Time
8	40 to <50°C	TempHist08Entry TempHist08Time
9	50 to <60°C	TempHist09Entry TempHist09Time
10	60 to <70°C	TempHist10Entry TempHist10Time
11	70 to <80°C	TempHist11Entry TempHist11Time
12	≥80°C	TempHist12Entry TempHist12Time

As soon as the ambient temperature reaches one of the predefined areas, a delay time of 3 s begins. After the delay time has expired, the entry counter is increased by 1 and the dwell time begins. The delay time prevents the counter from constantly being incremented in the crossover area.

Information:

The values are saved to module-internal FRAM.

If needed, the registers can be reset.

Registers are described in section "Ambient temperature" on page 56.

3.1.6 Histogram for acceleration

A histogram for acceleration is recorded in the module. The measurement range for the acceleration is divided into 8 areas:

Area	Acceleration	Register
1	-16 to <-12 g	AccHist0N01Entry AccHist0N01Time
2	-12 to <-8 g	AccHist0N02Entry AccHist0N02Time
3	-8 to <-4 g	AccHist0N03Entry AccHist0N03Time
4	-4 to <0 g	AccHist0N04Entry AccHist0N04Time
5	0 to <4 g	AccHist0N05Entry AccHist0N05Time
6	4 to <8 g	AccHist0N06Entry AccHist0N06Time
7	8 to <12 g	AccHist0N07Entry AccHist0N07Time
8	12 to 16 g	AccHist0N08Entry AccHist0N08Time

Legend: N = 1 to 3

As soon as the acceleration reaches one of the predefined areas, a delay time of 3 s begins. After the delay time has expired, the entry counter is increased by 1 and the dwell time begins. The delay time prevents the counter from constantly being incremented in the crossover area.

Information:

The values are saved to module-internal FRAM.

If needed, the registers can be reset.

Registers are described in section "Acceleration" on page 56.

3.1.7 Histogram for rotation

A histogram for rotation is recorded in the module. The measurement range for the rotation is divided into 8 areas:

Area	Rotation	Register
1	-2000 to <-1500 dps	RotationHist0N01Entry RotationHist0N01Time
2	-1500 to <-1000 dps	RotationHist0N02Entry RotationHist0N02Time
3	-1000 to <-500 dps	RotationHist0N03Entry RotationHist0N03Time
4	-500 to <0 dps	RotationHist0N04Entry RotationHist0N04Time
5	0 to <500 dps	RotationHist0N05Entry RotationHist0N05Time
6	500 to <1000 dps	RotationHist0N06Entry RotationHist0N06Time
7	1000 to <1500 dps	RotationHist0N07Entry RotationHist0N07Time
8	1500 to 2000 dps	RotationHist0N08Entry RotationHist0N08Time

Legend: N = 1 to 3

As soon as the rotation reaches one of the predefined areas, a delay time of 3 s begins. After the delay time has expired, the entry counter is increased by 1 and the dwell time begins. The delay time prevents the counter from constantly being incremented in the crossover area.

Information:

The values are saved to module-internal FRAM.

If needed, the registers can be reset.

Registers are described in section "Rotation" on page 56.

3.2 Recording operating data

The following operating data is collected by the module:

- Operating time with active connection to network master
- Operating time without active connection to network master (blackout mode)
- Total operating time
- Number of power-on cycles

> **Information:**
>
> **The values are saved to module-internal FRAM.**
>
> **If needed, the operating data can be reset.**
>
> **Registers are described in section "Operating data" on page 54.**

3.3 Internal module memory for user data

3.3.1 General information

The module is equipped with 512 kB nonvolatile internal flash memory that can be used by the application. Data can be saved directly to the module and then read back. This makes it possible to store recipe data or production information about the machine on the module, for example.

3.3.2 Operation

The module's memory interface is based on Flatstream communication. Operation takes place using library "AsFltGen".

> **Information:**
>
> **For additional information about library "AsFltGen", see Automation Help.**

> **Information:**
>
> **The following points must be observed:**
>
> - **Up to 256 bytes can be read or written per read or write command. These 256 bytes represent a page. If more than 256 bytes must be read or written, then a consecutive sequence of commands and memory management must be implemented in the application.**
> - **The erase command is based on sectors. One sector is 64 kB. This corresponds to 256 pages. The entire sector containing the specified address is erased. The flash memory is divided into a total of 8 sectors (8 x 64 kB = 512 kB).**
> - **In order to overwrite data, the corresponding sector must first be erased. Only then can the new data be saved.**
> - **Memory can be arranged as needed. A separate sector should be used for data that is overwritten regularly.**

3.3.3 Commands

3.3.3.1 Protocol

A header precedes each command. The data follows the header and depends on the command.

3.3.3.2 Header

Each request or response begins with a 16-byte header. The following elements must be defined in the header:

Element	Data type	Activity	Description
Code	USINT	Requirement	Defines the command: "r" ... Read data (ASCII code 0x72) "w" ... Write data (ASCII code 0x77) "e" ... Erase data (ASCII code 0x65)
		Response	The command code contained in the request is sent back.
Consecutive number	USINT	Requirement	Unrestricted use. The consecutive number is important, for example, if more than 256 bytes must be read or written. In this case, the user must implement a sequential series of commands and memory management in the application.
		Response	The number contained in the request is sent back.
Status	UINT	Requirement	Not used: The byte is not evaluated.
		Response	Status response: 0x0000 ... Command executed successfully 0x8001 ... Invalid: General fault 0x8002 ... Invalid address 0x8003 ... Invalid size 0x8004 ... Flash memory busy 0x8006 ... Flash memory timeout
Address	UDINT	Requirement	Starting address from which data should be read or written.
		Response	The starting address contained in the request is sent back.
Data size	UDINT	Requirement	Size of the data to be read or written.
		Response	The data size contained in the request is sent back.
Reserve	UDINT	Reserved	

3.3.3.3 Write data

Action	Description
Requirement	In order to save data to the module, the header must be prepared for communication. The data is appended directly to the header. The header and data must be specified to function block "fltWrite" as a transmit buffer. Header | Data
Response	The module's response – the returned header – is stored in the receive buffer using function block "fltRead" and can be evaluated by the application. Header

3.3.3.4 Read data

Action	Description
Requirement	In order to read data from the module, the header must be prepared for communication. The header must be specified to function block "fltWrite" as a transmit buffer. Header
Response	The module's response – the returned header and data – is stored in the receive buffer using function block "fltRead" and can be evaluated by the application. Header | Data

3.3.3.5 Erasing a sector

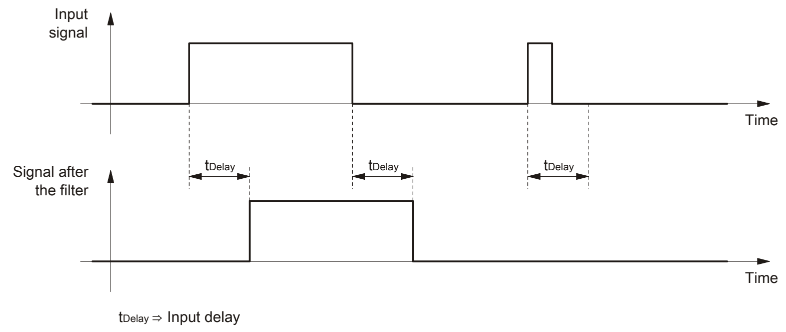
Action	Description
Requirement	In order to erase an area of the module's flash memory, the header must be prepared for communication. The entire 64 kB sector containing the specified address is erased. The header must be specified to function block "fltWrite" as a transmit buffer. Header
Response	The module's response – the returned header – is stored in the receive buffer using function block "fltRead" and can be evaluated by the application. Header

3.4 Digital inputs

This module is equipped with 2 digital inputs. This makes it possible to use a door contact to determine whether the control cabinet door is open or closed, for example.

3.4.1 Input filter

An input filter is available for each input. Disturbance pulses that are shorter than the input delay are suppressed by the input filter.



The input delay can be set in steps of 100 µs. It makes sense to enter values in steps of 2, however, since the input signals are sampled every 200 µs.

Values	Filter
0	No software filter
2	0.2 ms
...	...
250	25 ms - Higher values are limited to this value.

Information:

The register is described in section "Digital input filter" on page 49.

3.4.2 Record input state

Unfiltered

The input state is collected with a fixed offset to the network cycle and transferred in the same cycle.

Filtered

The filtered state is collected with a fixed offset to the network cycle and transferred in the same cycle. Filtering takes place asynchronously to the network in multiples of 200 µs with a network-related jitter of up to 50 µs.

3.5 Analog inputs

Using 2 PT1000 temperature inputs, the control cabinet temperature can be determined at critical locations.

3.5.1 Filter level

The filter level determines the number of measurements for averaging. The external temperature sensors are measured every 1 ms. This means that by default, a new value (100 x 1 ms = 100 ms) with 100 measurements is available for averaging every 100 ms.

Information:

The register is described in section "Filter level and input ramp limiting" on page 50.

3.5.2 Input ramp limiting

Input ramp limiting can only be performed in conjunction with filtering. Input ramp limiting is performed before filtering.

The difference of the input value change is checked for exceeding the specified limit. In the event of overshoot, the tracked input value is equal to the old value ± the limit value. The input value is monitored with the scanning grid every 1 ms.

Input ramp limiting is well suited for suppressing disturbances (spikes).

Information:

The register is described in section **"Filter level and input ramp limiting" on page 50****.**

3.5.3 Limiting the analog value

When an error state occurs, the analog value is set by default to the values listed below.

Error state	Digital value on error
Open circuit	32767 (0x7FFF)
Upper limit value overshot	32767 (0x7FFF)
Lower limit value undershot	-32767 (0x8001)
Invalid value	-32768 (0x8000)[1]

1) After switching off the channel during operation or when the channel is disabled

3.5.4 Output value

In order for the user to always be supplied with a defined output value, the following must be taken into consideration:

- Up to the first conversion, 0x8000 is output.
- If the input is not switched on, 0x8000 is output.

3.6 Blackout mode

Blackout mode allows users to continue execution of the application in lower-level subsystems if components of the B&R system fail. In this way, the B&R system – independently of redundancy technology – makes it possible to respond to system-critical situations based on the specific application.

The use of blackout-capable modules is recommended for the following requirements:

- Exit routines on system failure, e.g. to enable the opening of a press if the system fails.
- Stopping or controlled setting of an output on system failure, e.g. to automatically close inflow valves.
- Deceleration sequences on system failure, e.g. to reduce motor speeds before transmitting a stop command.

If blackout-capable modules are configured accordingly, blackout mode will be carried out if the network connection to the higher-level controller is interrupted.

As soon as the network disturbance has been corrected, blackout mode is stopped by the modules and bumpless synchronization with the network takes place.

Requirements for operation

The following requirements must be met in order to use blackout mode:

- The module being used must support blackout mode.
- Parameter "Blackout mode" must be enabled in Automation Studio.

3.6.1 Areas of use

Through the use of blackout-capable modules, a part of the control system can also remain functional if a disturbance in the network or X2X Link connection between the modules occurs.

3.6.1.1 Loss of POWERLINK connection

Initial situation

Several stations in an application are connected to the controller via network cables. A fault occurs that interrupts data transfer between the controller and stations.

Effect

Non-blackout modules are reset and operated according to their default behavior.

Blackout-capable modules show the following behavior:

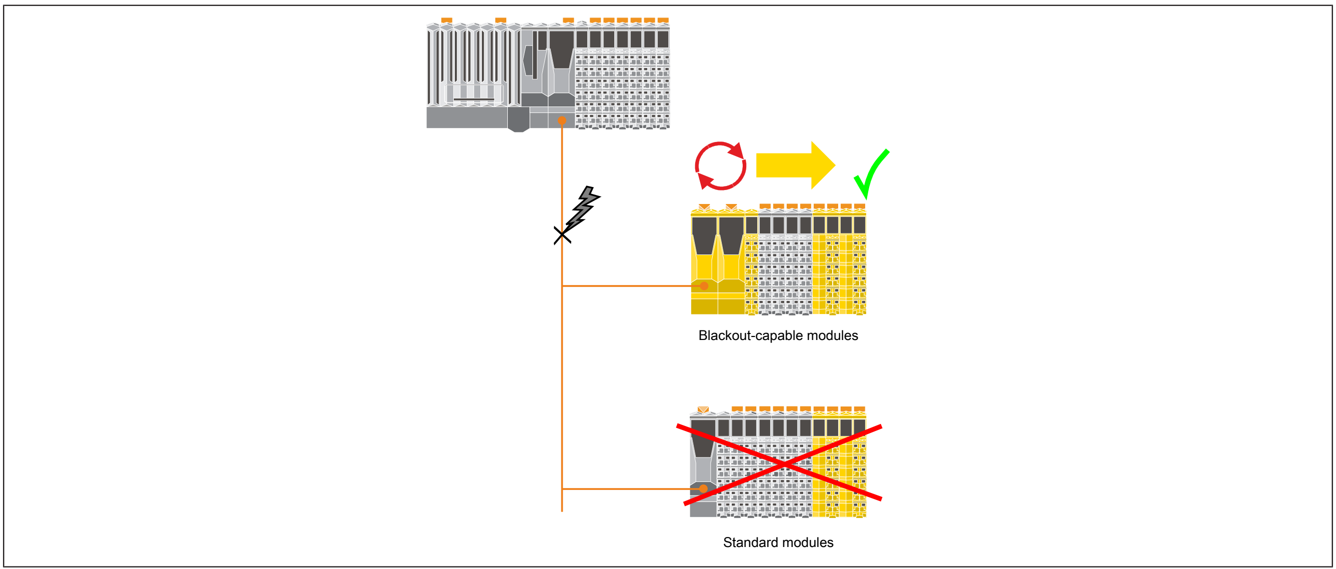
- The programmed function continues to be executed.
- Subordinate networks continue to work.
- Data from the controller is initialized with "0".
- After the disturbance has been corrected, the module bumplessly returns to the higher-level network.

Warning!

Blackout mode causes data from the controller to be initialized with "0". If blackout mode is used in combination with "output inversion", this can result in the unintentional setting of outputs.

Mise en garde !

Le mode « Blackout » entraîne l'initialisation des données du contrôleur avec « 0 ». Si le mode « Blackout » est utilisé en combinaison avec « l'inversion de sortie », il peut en résulter un réglage involontaire des sorties.



3.6.1.2 Loss of X2X Link connection

Initial situation

Modules in an application are connected to the network via X2X Link cables. A defect in the X2X Link cable causes the data transfer between the controller and modules to be interrupted.

Effect

Non-blackout modules are reset and operated according to their default behavior.

Blackout-capable modules show the following behavior:

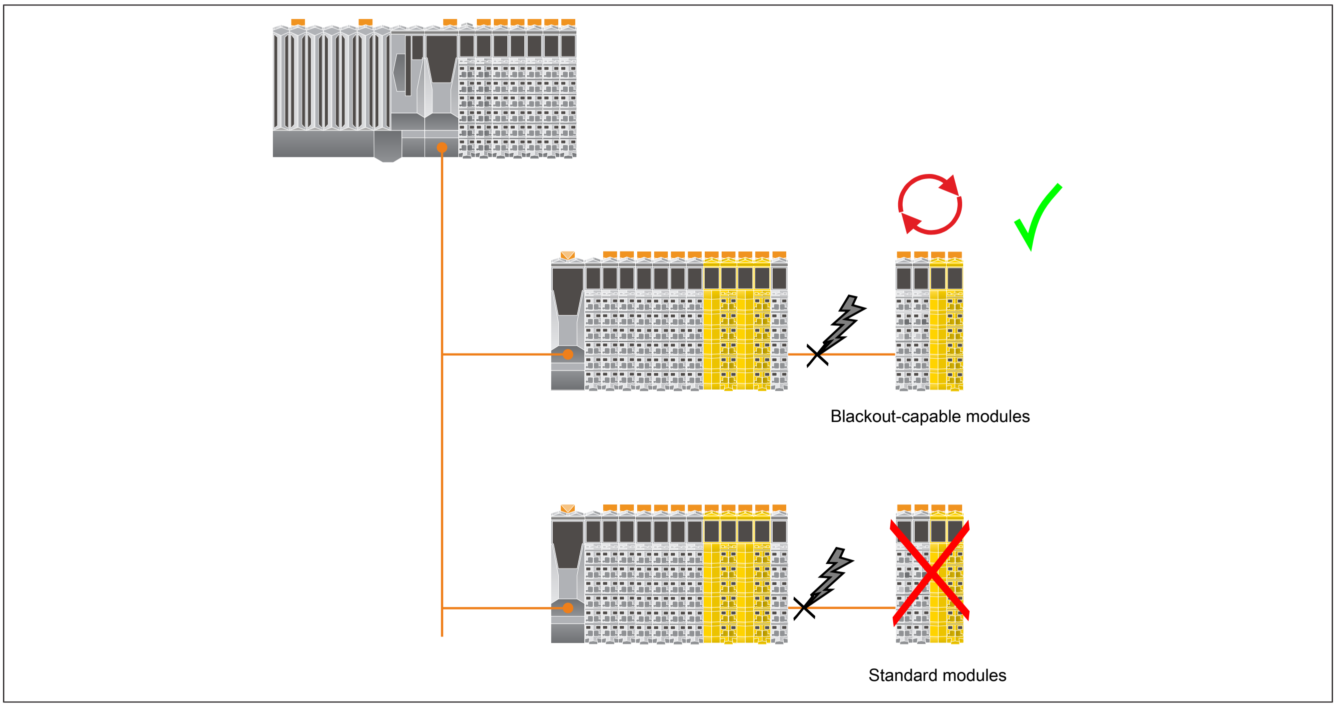
- The programmed function continues to be executed.
- Subordinate networks continue to work.
- Data from the controller is initialized with "0".
- After the disturbance has been corrected, the module bumplessly returns to the higher-level network.

Warning!

Blackout mode causes data from the controller to be initialized with "0". If blackout mode is used in combination with "output inversion", this can result in the unintentional setting of outputs.

Mise en garde !

Le mode « Blackout » entraîne l'initialisation des données du contrôleur avec « 0 ». Si le mode « Blackout » est utilisé en combinaison avec « l'inversion de sortie », il peut en résulter un réglage involontaire des sorties.



3.6.2 Programming blackout mode

Blackout mode cannot be detected by the blackout-capable modules themselves. If it is necessary to program specific blackout behavior in an application, an indirect method must therefore be chosen.

One possibility is to implement a counter in the blackout-capable module's higher-level controller and query it cyclically. Blackout mode would make itself noticeable in this case by a counter value that no longer changes or a counter value of zero.

Blackout-capable modules can be divided into 2 categories:

- **Programmable modules**
  The blackout function is programmed using existing function blocks. In other words, the existing technologies for application programming or reACTION Technology are used.
  The blackout function is executed largely independently of other system components.
- **Standard function modules**
  These modules are not programmable and maintain their default behavior in blackout mode.

3.7 Flatstream communication

3.7.1 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.

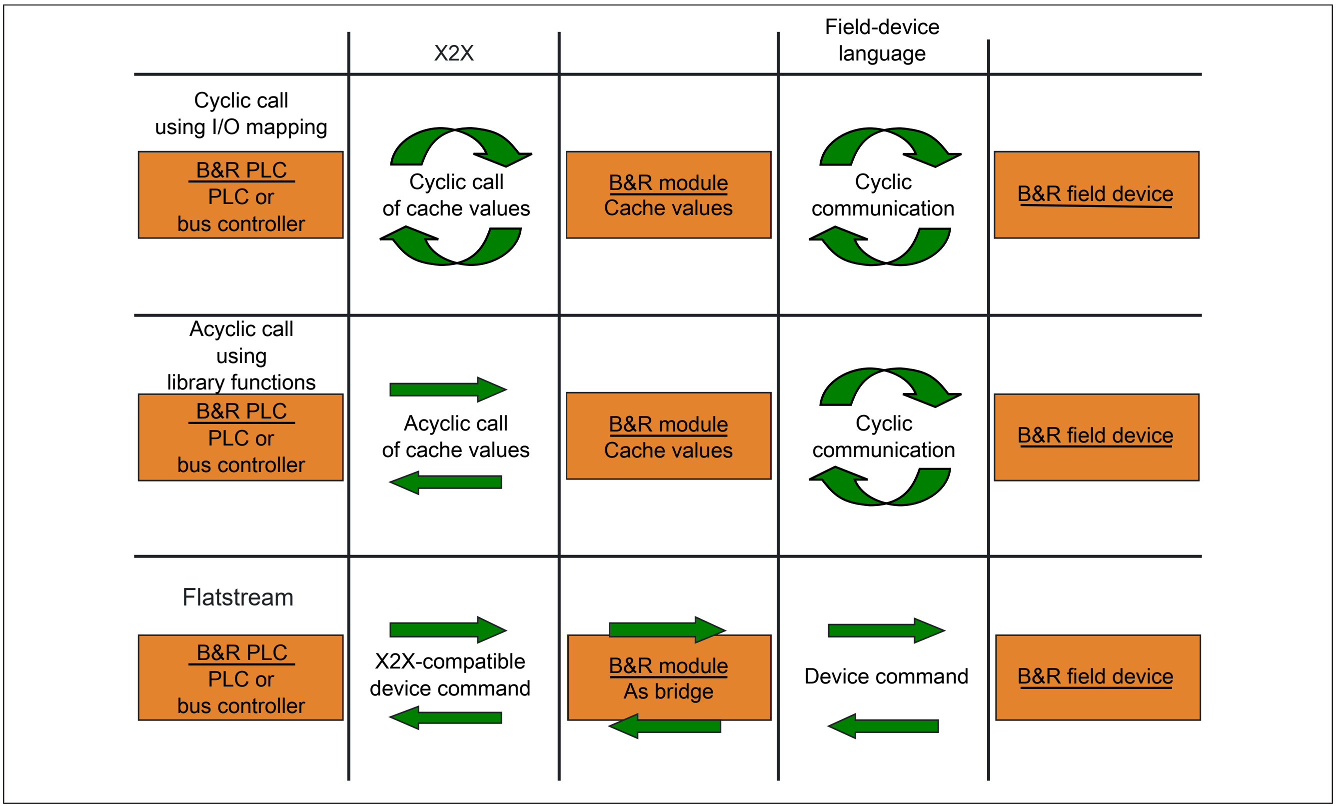


Figure 1: 3 types of communication

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.

3.7.2 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.

3.7.3 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.

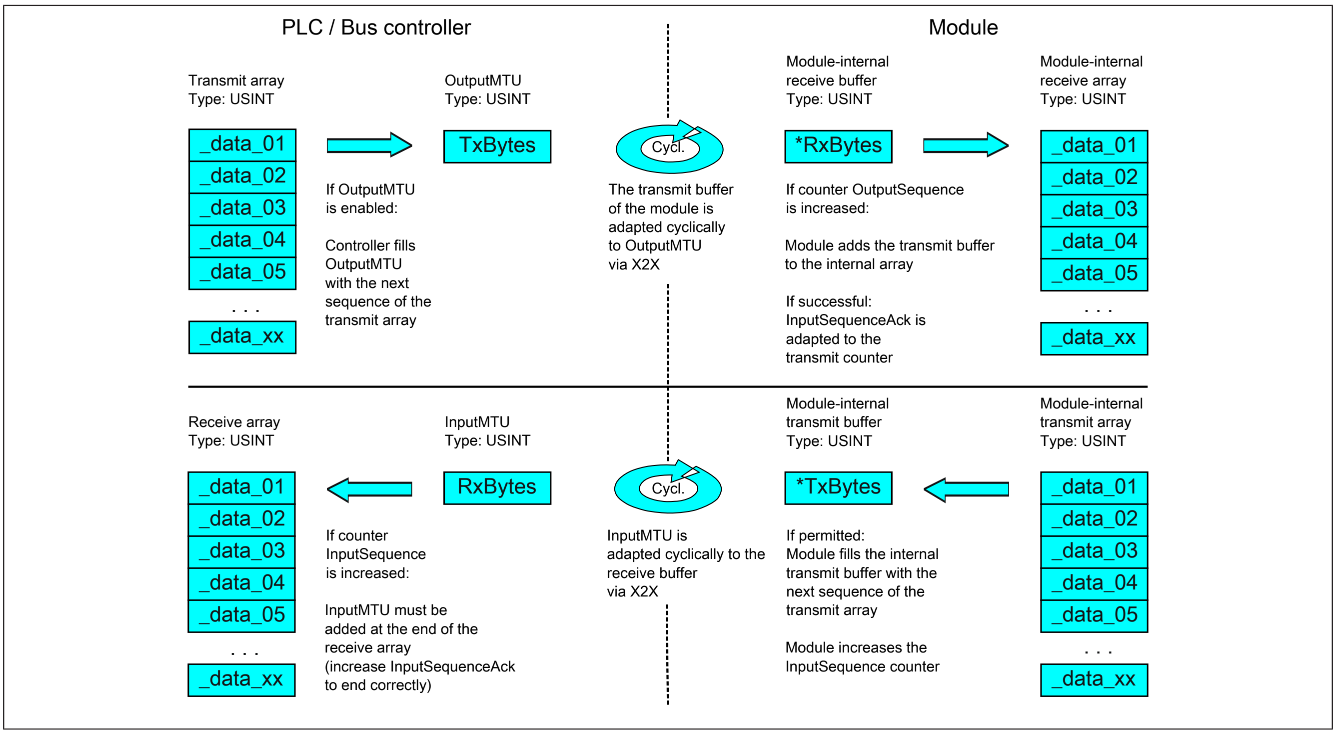


Figure 2: Flatstream communication

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 refilled. 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.

3.7.4 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.**

3.7.4.1 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 57.**
>
> **Registers are described in section "Flatstream communication" in the respective data sheets.**

3.7.4.2 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 57.**
>
> **Registers are described in section "Flatstream communication" in the respective data sheets.**

3.7.4.2.1 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.

3.7.4.2.2 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.

3.7.4.2.2.1 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

3.7.4.2.3 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.

3.7.4.2.3.1 Relationship between OutputSequence and InputSequence

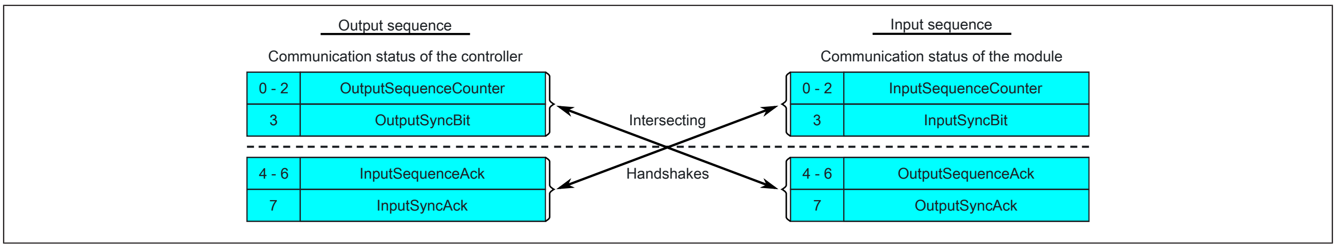


Figure 3: Relationship between OutputSequence and InputSequence

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.

3.7.4.3 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). **Note:** 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. **Note:** 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.

3.7.4.4 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:

Position (of the next control byte) = Current position + 1 + 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.

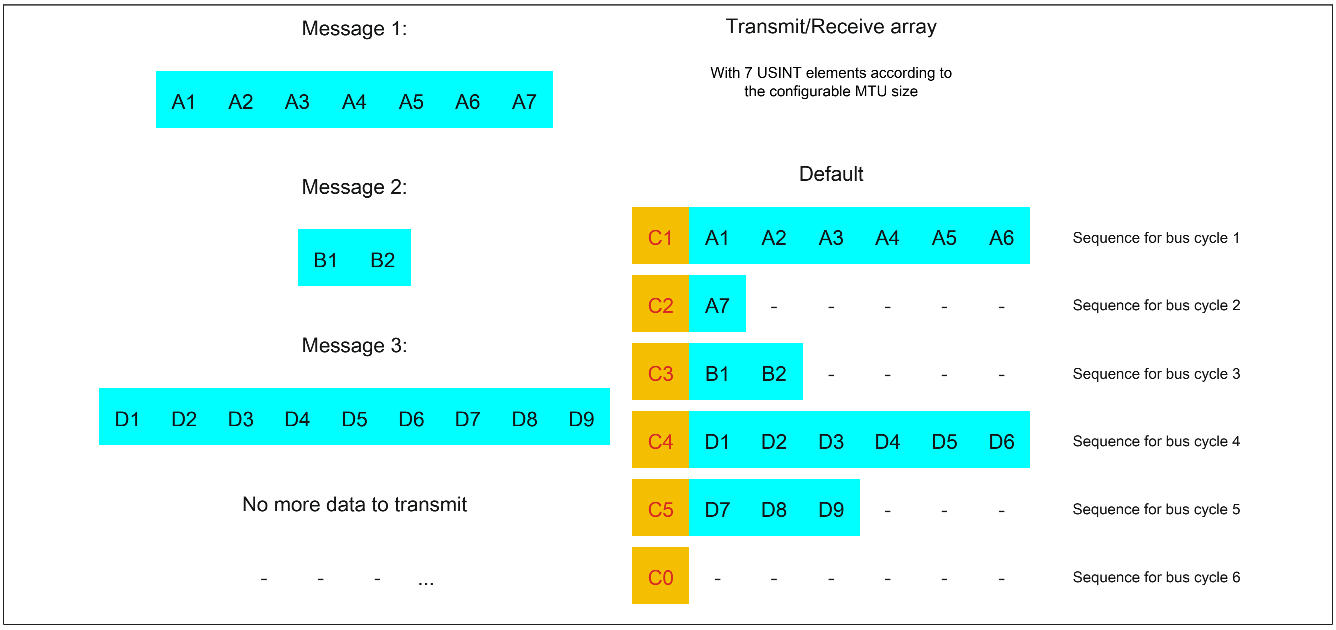


Figure 4: Transmit/Receive array (default)

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)

3.7.4.4.1 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.

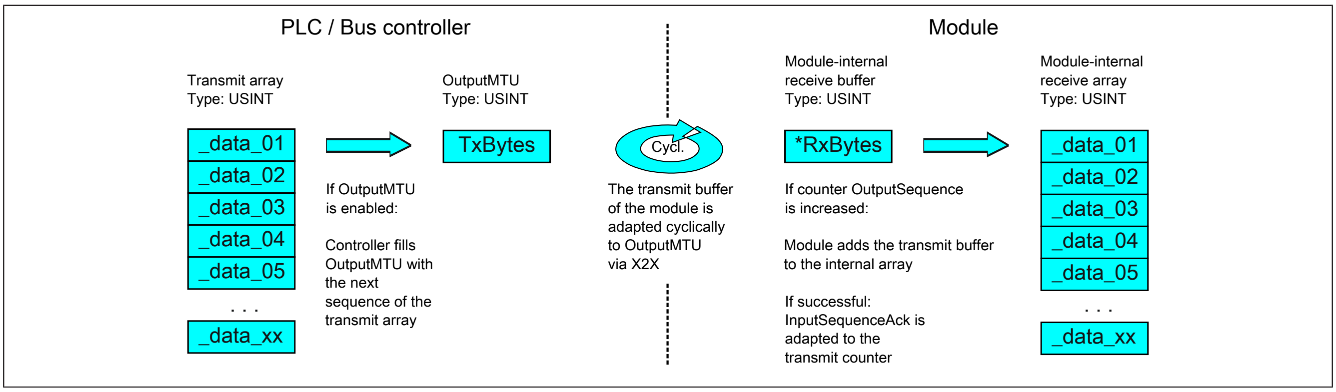


Figure 5: Flatstream communication (output)

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. **Note:** 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.

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

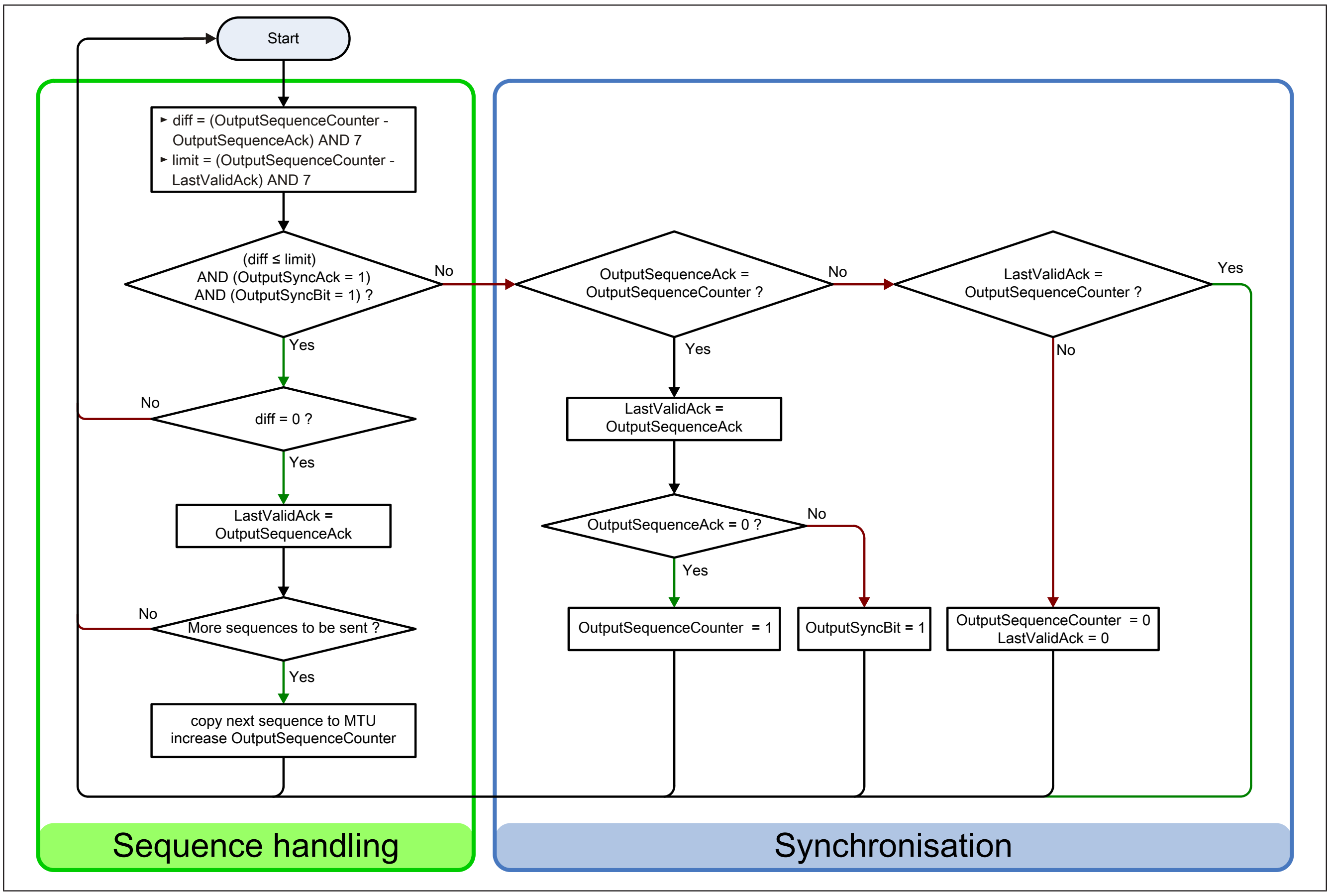


Figure 6: Flowchart for the output direction

3.7.4.4.2 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.

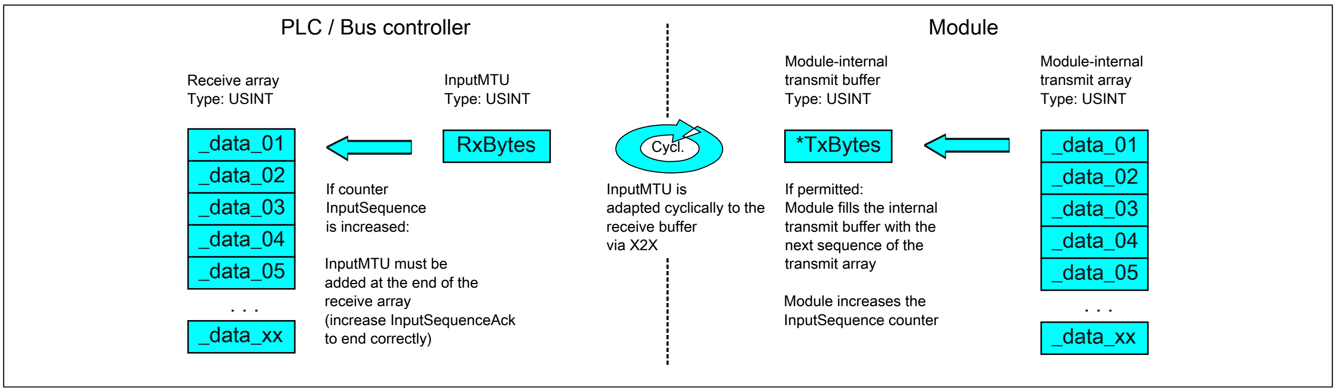


Figure 7: Flatstream communication (input)

Algorithm

0) Cyclic status query: **- 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.
1) Receiving (as soon as InputSequenceCounter is increased): **- 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.

General flowchart

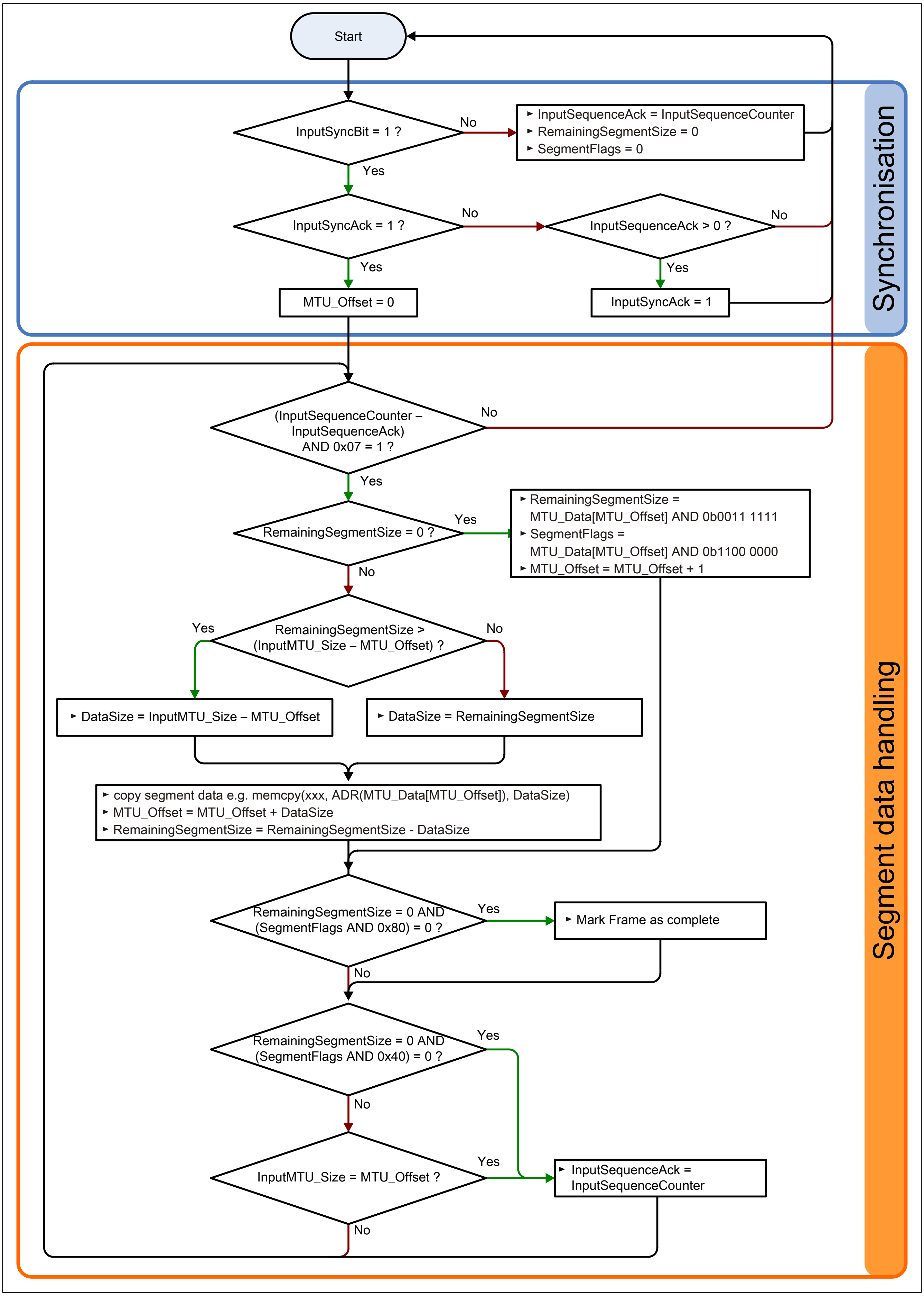


Figure 8: Flowchart for the input direction

3.7.4.4.3 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.

3.7.4.5 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.

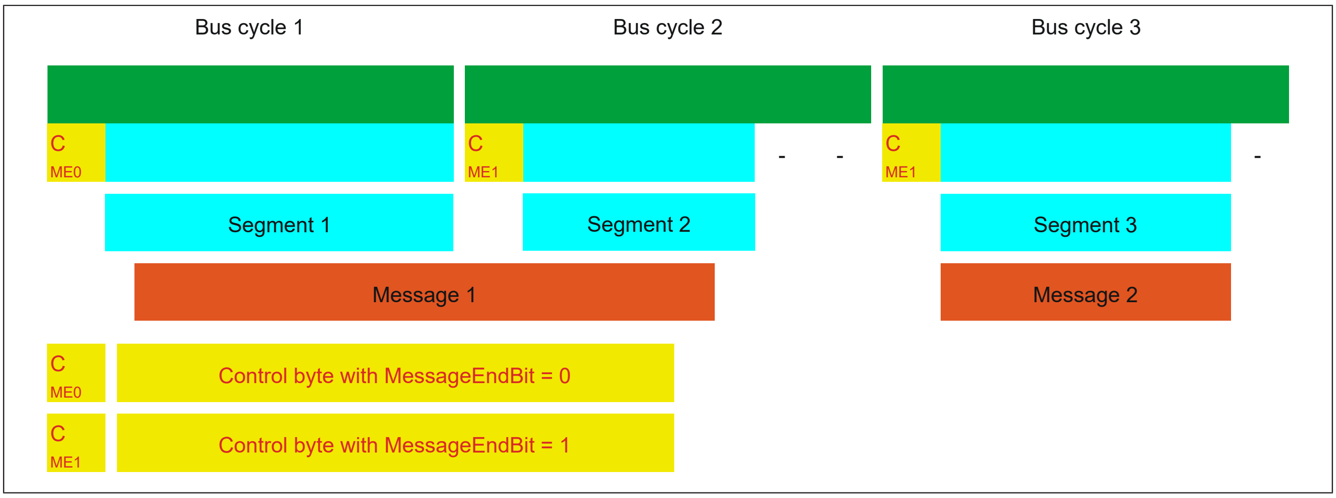


Figure 9: Message arrangement in the MTU (default)

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.

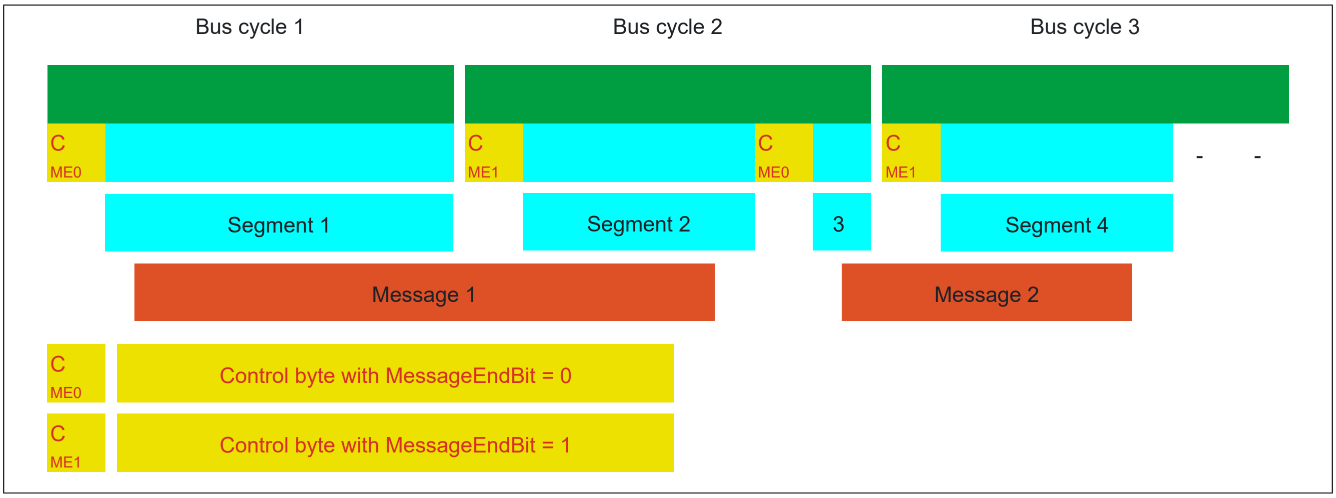


Figure 10: Arrangement of messages in the MTU (MultiSegmentMTU)

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.

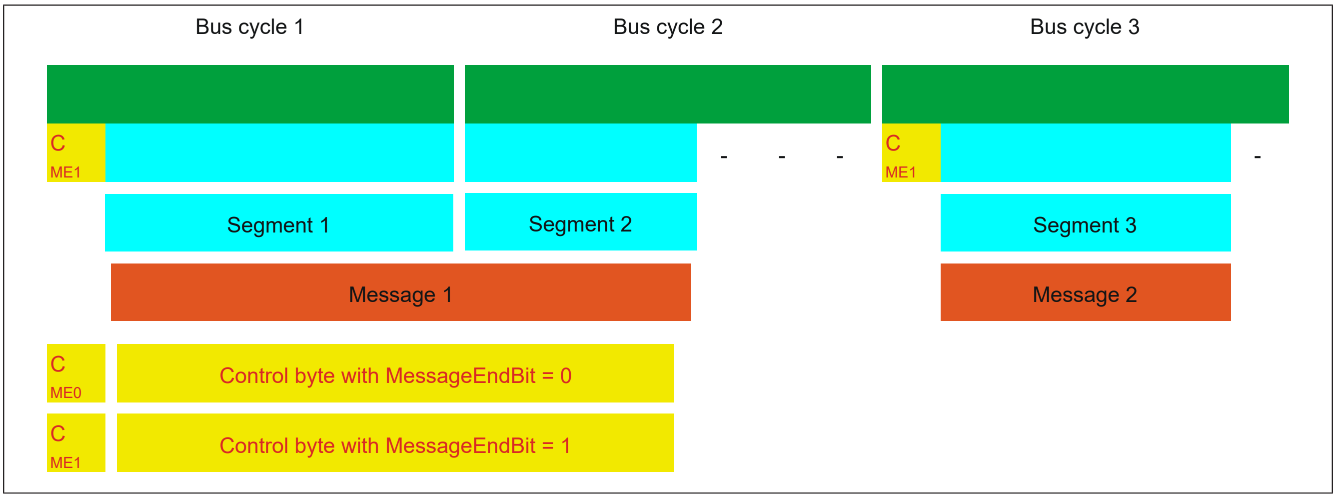


Figure 11: Arrangement of messages in the MTU (large segments)

Using both options

Using both options at the same time is also permitted.

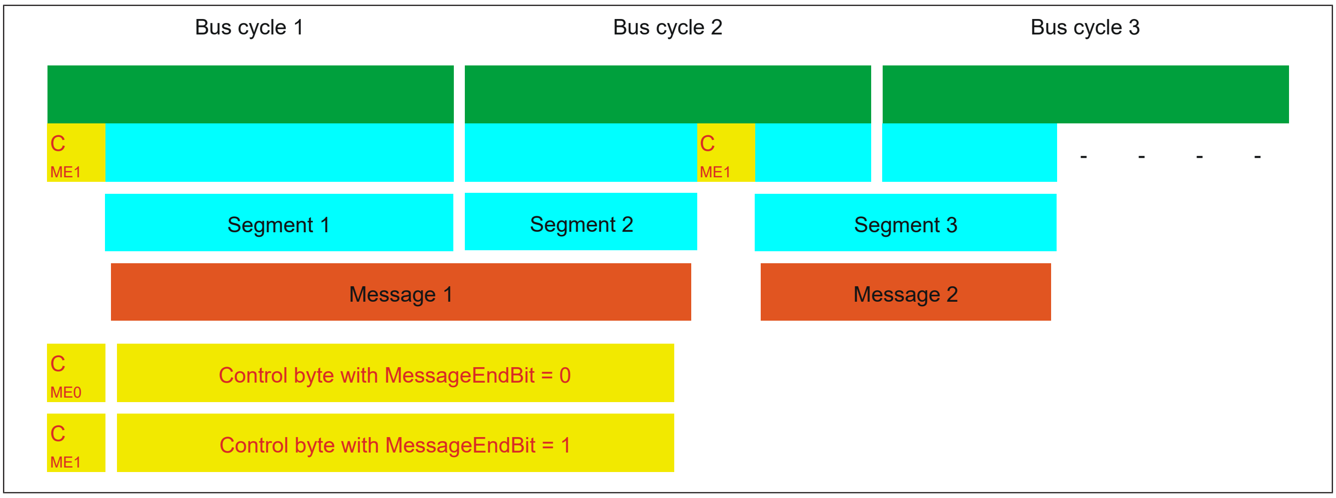


Figure 12: Arrangement of messages in the MTU (large segments and MultiSegmentMTU)

3.7.4.6 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 "nextCBPos" 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.

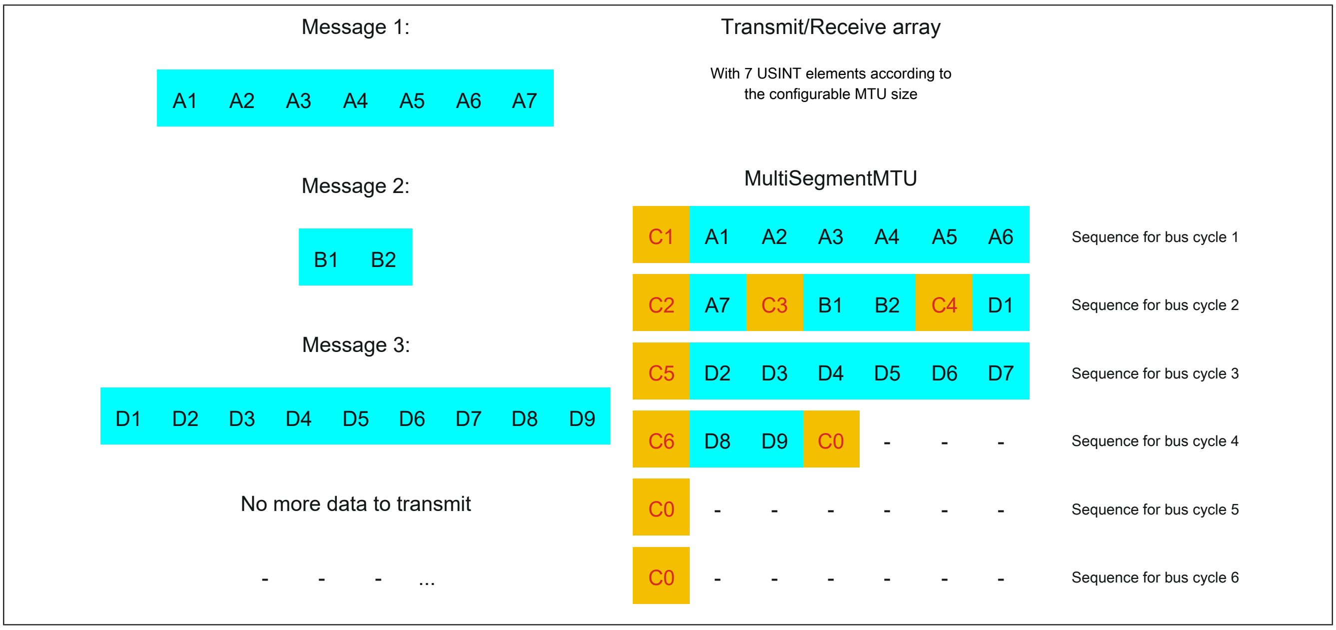


Figure 13: Transmit/Receive array (MultiSegmentMTU)

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.

Mise en garde !

La deuxième séquence ne peut être acquittée via SequenceAck que si elle a été entièrement traitée. Dans cet exemple, il y a 3 segments différents dans la deuxième séquence, c'est-à-dire que le programme doit inclure suffisamment de tableaux de réception pour gérer cette 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)

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.

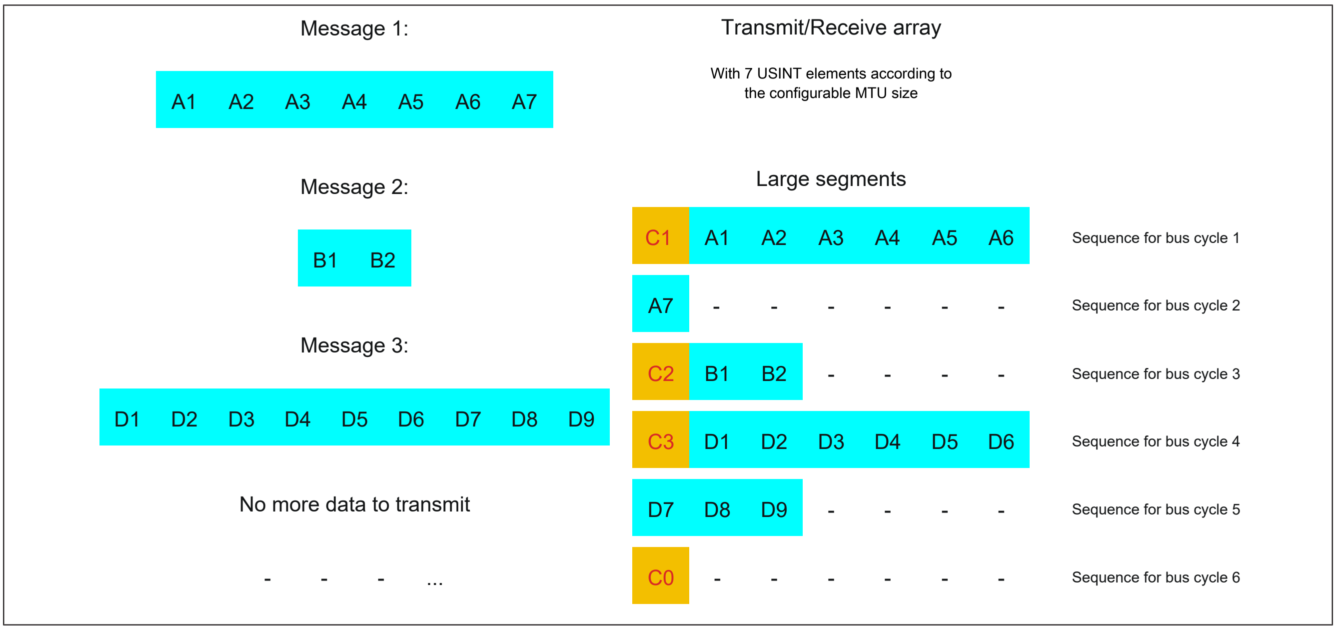


Figure 14: Transmit/receive array (large segments)

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.

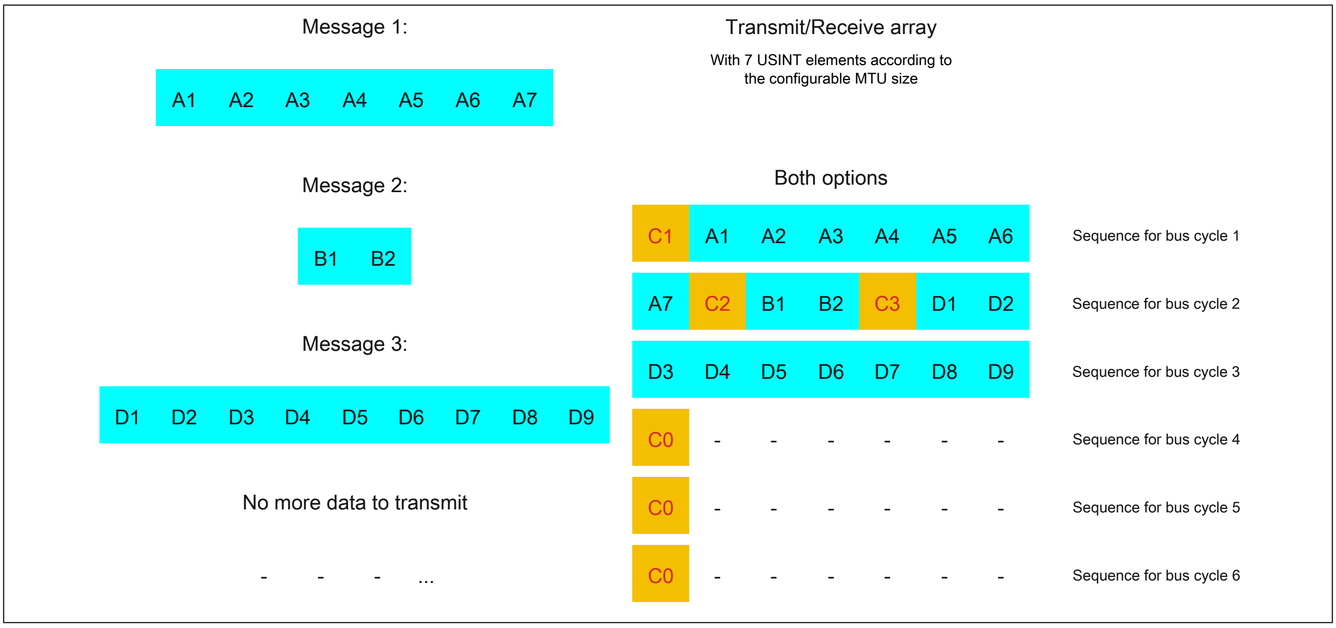


Figure 15: Transmit/Receive array (large segments and MultiSegmentMTU)

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

3.7.5 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.

3.7.5.1 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
Actions	Transfer sequence from transmit array, increase SequenceCounter	Cyclic synchronization of MTU and module buffer	Append sequence to receive array, adjust SequenceAck	Cyclic synchronization MTU and module buffer	Check SequenceAck
Resource	Transmitter (task to transmit)	Bus system (direction 1)	Recipients (task to receive)	Bus system (direction 2)	Transmitter (task for Ack checking)

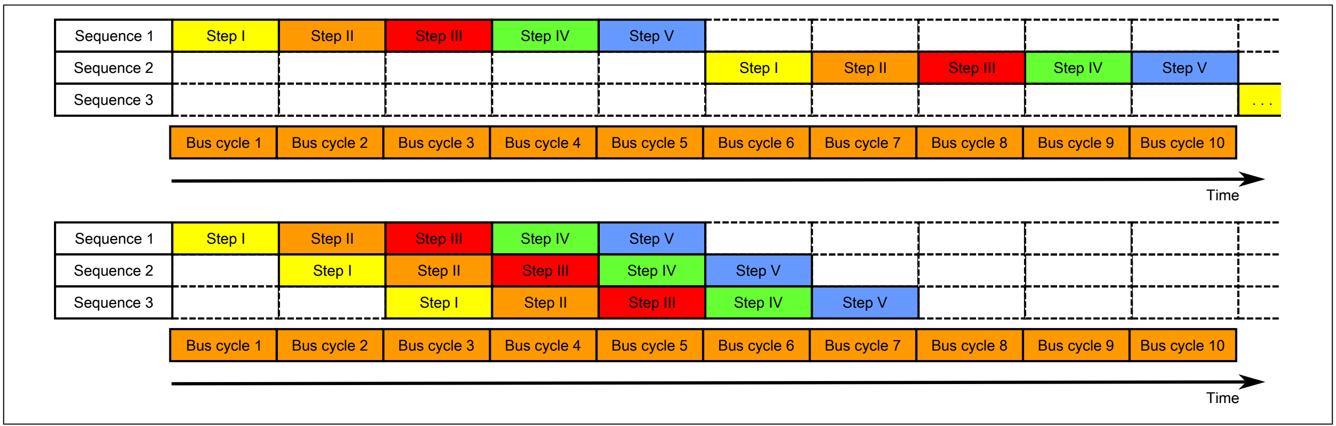


Figure 16: Comparison of transfer without/with Forward

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.

3.7.5.2 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 57.**
>
> **Registers are described in section "Flatstream communication" in the respective data sheets.**

3.7.5.2.1 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.

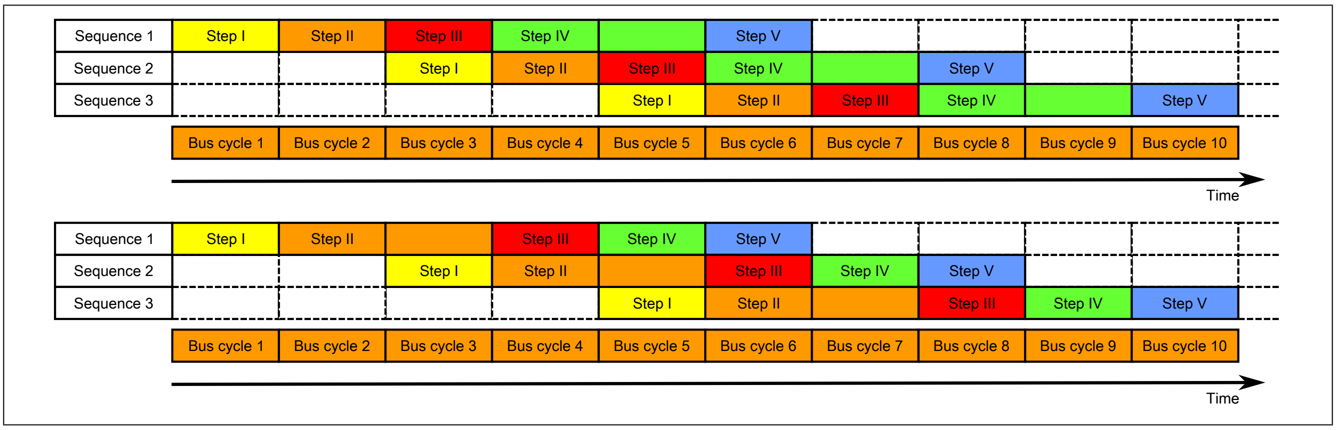


Figure 17: Effect of ForwardDelay when using Flatstream communication with the Forward function

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.

3.7.5.3 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. **Note:** 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.

3.7.5.4 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).

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).

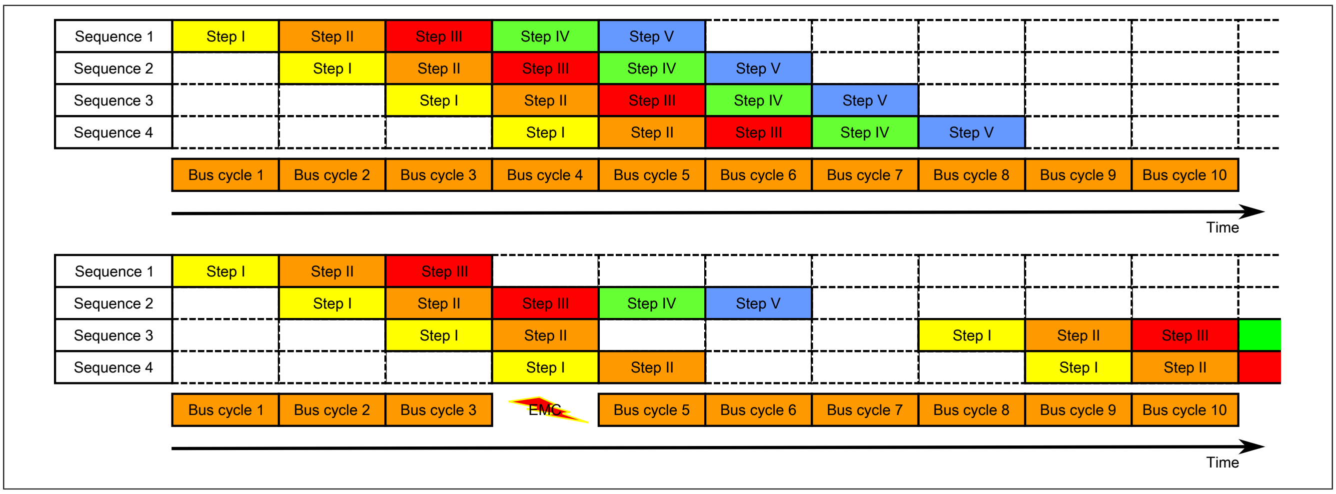


Figure 18: Effect of a lost bus cycle

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 Register description

4.1 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.

4.2 Function model 0 - Standard

Register	Name	Data type	Read		Write	
			Cyclic	Acyclic	Cyclic	Acyclic
Module - Configuration						
794	CfO_DiFilter	UINT				•
798	CfO_AnFilter	UINT				•
802	CfO_AnEnable	UINT				•
Module - Control						
129	Switching state of the digital output	USINT			•	
	DigitalOutput01	Bit 0				
134	Reset additional information and data point histograms	UINT			•	
	ClrStatistics_OperatingData	Bit 0				
	ClrStatistics_RelHumidity	Bit 1				
	ClrStatistics_Temperature	Bit 2				
	ClrStatistics_Acceleration01	Bit 8				
	ClrStatistics_Acceleration02	Bit 9				
	ClrStatistics_Acceleration03	Bit 10				
	ClrStatistics_Rotation01	Bit 11				
	ClrStatistics_Rotation02	Bit 12				
	ClrStatistics_Rotation03	Bit 13				
Module communication						
41	Status of digital inputs, digital output and I/O power supply	USINT	•			
	DigitalInput01	Bit 0				
	DigitalInput02	Bit 1				
	StateDigitalOutput01	Bit 4				
	PowerSupply	Bit 7				
46	ModuleState01	UINT	•			
Measured values						
2	RelHumidity	INT	•			
6	Temperature	INT	•			
10	Acceleration01	INT	•			
14	Acceleration02	INT	•			
18	Acceleration03	INT	•			
22	Rotation01	INT	•			
26	Rotation02	INT	•			
30	Rotation03	INT	•			
34	TempExt01	INT	•			
38	TempExt02	INT	•			
Additional information						
4100	OnTimeConnected	UDINT	•			
4108	OnTimeDisconnected	UDINT	•			
4116	OnTimeCombined	UDINT	•			
4124	PowerCycles	UDINT	•			
4134	RelHumidityMin	INT	•			
4138	RelHumidityMax	INT	•			
4150	TemperatureMin	INT	•			
4154	TemperatureMax	INT	•			
4166 + N*16	Acceleration0NMin (index N = 1 to 3)	INT	•			
4170 + N*16	Acceleration0NMax (index N = 1 to 3)	INT	•			
4198 + N*16	Rotation0NMin (index N = 1 to 3)	INT	•			
4202 + N*16	Rotation0NMax (index N = 1 to 3)	INT	•			

Register description

Register	Name	Data type	Read		Write	
			Cyclic	Acyclic	Cyclic	Acyclic
Data point histogram						
4244 + N*16	RelHumHist0NEntry (index N = 1 to 10)	UDINT	•			
4252 + N*16	RelHumHist0NTime (index N = 1 to 10)	UDINT	•			
4404 + N*16	TempHist0NEntry (index N = 1 to 12)	UDINT	•			
4412 + N*16	TempHist0NTime (index N = 1 to 12)	UDINT	•			
4596 + N*16	AccHist010NEntry (index N = 1 to 8)	UDINT	•			
4604 + N*16	AccHist010NTime (index N = 1 to 8)	UDINT	•			
4724 + N*16	AccHist020NEntry (index N = 1 to 8)	UDINT	•			
4732 + N*16	AccHist020NTime (index N = 1 to 8)	UDINT	•			
4852 + N*16	AccHist030NEntry (index N = 1 to 8)	UDINT	•			
4860 + N*16	AccHist030NTime (index N = 1 to 8)	UDINT	•			
4980 + N*16	RotationHist010NEntry (index N = 1 to 8)	UDINT	•			
4988 + N*16	RotationHist010NTime (index N = 1 to 8)	UDINT	•			
5108 + N*16	RotationHist020NEntry (index N = 1 to 8)	UDINT	•			
5116 + N*16	RotationHist020NTime (index N = 1 to 8)	UDINT	•			
5236 + N*16	RotationHist030NEntry (index N = 1 to 8)	UDINT	•			
5244 + N*16	RotationHist030NTime (index N = 1 to 8)	UDINT	•			
Flatstream - Configuration (access to internal flash memory)						
513	OutputMTU	USINT				•
515	InputMTU	USINT				•
517	FlatstreamMode	USINT				•
519	Forward	USINT				•
522	ForwardDelay	UINT				•
Flatstream - Communication (access to internal flash memory)						
577	InputSequence	USINT	•			
577 + N*2	RxByteN (index N = 1 to 27)	USINT	•			
641	OutputSequence	USINT			•	
641 + N*2	TxByteN (index N = 1 to 15)	USINT			•	

4.3 Function model 254 - Bus controller

Register	Offset[1]	Name	Data type	Read		Write	
				Cyclic	Acyclic	Cyclic	Acyclic
Module - Configuration							
794	-	CfO_DiFilter	UINT				•
798	-	CfO_AnFilter	UINT				•
802	-	CfO_AnEnable	UINT				•
Module - Control							
129	2	Switching state of the digital output	USINT			•	
		DigitalOutput01	Bit 0				
134	-	Reset additional information and data point histograms	UINT				•
		ClrStatistics_OperatingData	Bit 0				
		ClrStatistics_RelHumidity	Bit 1				
		ClrStatistics_Temperature	Bit 2				
		ClrStatistics_Acceleration01	Bit 8				
		ClrStatistics_Acceleration02	Bit 9				
		ClrStatistics_Acceleration03	Bit 10				
		ClrStatistics_Rotation01	Bit 11				
		ClrStatistics_Rotation02	Bit 12				
		ClrStatistics_Rotation03	Bit 13				
Module communication							
41	20	Status of digital inputs, digital output and I/O power supply	USINT	•			
		DigitalInput01	Bit 0				
		DigitalInput02	Bit 1				
		StateDigitalOutput01	Bit 4				
		PowerSupply	Bit 7				
46	22	ModuleState01	UINT	•			
Measured values							
2	0	RelHumidity	INT	•			
6	2	Temperature	INT	•			
10	4	Acceleration01	INT	•			
14	6	Acceleration02	INT	•			
18	8	Acceleration03	INT	•			
22	10	Rotation01	INT	•			
26	12	Rotation02	INT	•			
30	14	Rotation03	INT	•			
34	16	TempExt01	INT	•			
38	18	TempExt02	INT	•			

Register	Offset[1]	Name	Data type	Read		Write	
				Cyclic	Acyclic	Cyclic	Acyclic
Additional information							
4100	-	OnTimeConnected	UDINT		•		
4108	-	OnTimeDisconnected	UDINT		•		
4116	-	OnTimeCombined	UDINT		•		
4124	-	PowerCycles	UDINT		•		
4134	-	RelHumidityMin	INT		•		
4138	-	RelHumidityMax	INT		•		
4150	-	TemperatureMin	INT		•		
4154	-	TemperatureMax	INT		•		
4166 + N*16	-	Acceleration0NMin (index N = 1 to 3)	INT		•		
4170 + N*16	-	Acceleration0NMax (index N = 1 to 3)	INT		•		
4198 + N*16	-	Rotation0NMin (index N = 1 to 3)	INT		•		
4202 + N*16	-	Rotation0NMax (index N = 1 to 3)	INT		•		
Data point histogram							
4244 + N*16	-	RelHumHist0NEntry (index N = 1 to 10)	UDINT		•		
4252 + N*16	-	RelHumHist0NTime (index N = 1 to 10)	UDINT		•		
4404 + N*16	-	TempHist0NEntry (index N = 1 to 12)	UDINT		•		
4412 + N*16	-	TempHist0NTime (index N = 1 to 12)	UDINT		•		
4596 + N*16	-	AccHist010NEntry (index N = 1 to 8)	UDINT		•		
4604 + N*16	-	AccHist010NTime (index N = 1 to 8)	UDINT		•		
4724 + N*16	-	AccHist020NEntry (index N = 1 to 8)	UDINT		•		
4732 + N*16	-	AccHist020NTime (index N = 1 to 8)	UDINT		•		
4852 + N*16	-	AccHist030NEntry (index N = 1 to 8)	UDINT		•		
4860 + N*16	-	AccHist030NTime (index N = 1 to 8)	UDINT		•		
4980 + N*16	-	RotationHist010NEntry (index N = 1 to 8)	UDINT		•		
4988 + N*16	-	RotationHist010NTime (index N = 1 to 8)	UDINT		•		
5108 + N*16	-	RotationHist020NEntry (index N = 1 to 8)	UDINT		•		
5116 + N*16	-	RotationHist020NTime (index N = 1 to 8)	UDINT		•		
5236 + N*16	-	RotationHist030NEntry (index N = 1 to 8)	UDINT		•		
5244 + N*16	-	RotationHist030NTime (index N = 1 to 8)	UDINT		•		

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

4.4 Configuration

4.4.1 Digital input filter

Name:
CfO_DiFilter

The filter value for all digital inputs can be configured in this register.

Data type	Values	Filter
USINT	0	No software filter (bus controller default setting)
	2	0.2 ms
	...	...
	250	25 ms - Higher values are limited to this value.

4.4.2 Filter level and input ramp limiting

Name:
CfO_AnFilter

The filter level and input ramp limiting of the input filter of the analog inputs are set in this register.

Data type	Values
UINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0 - 7	Filter level	0	Filter switched off
		1	1 measurement
		2	2 measurements
		3	3 measurements
		:	:
		100	100 measurements: Default
		:	:
		255	255 measurements
8 - 10	Input ramp limiting	000	The input value is applied without limiting.
		001	Limit value = 0x3FFF (16383)
		010	Limit value = 0x1FFF (8191)
		011	Limit value = 0x0FFF (4095)
		100	Limit value = 0x07FF (2047)
		101	Limit value = 0x03FF (1023)
		110	Limit value = 0x01FF (511)
		111	Limit value = 0x00FF (255): Default
11 - 15	Reserved	0	

4.4.3 Enable analog input

Name:
CfO_AnEnable

These registers enable the analog inputs.

Data type	Values
UINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	Channel 1	0	Not enabled
		1	Enabled
1 - 7	Reserved	0	
8	Channel 2	0	Not enabled
		1	Enabled
9 - 15	Reserved	0	

4.5 Controller

4.5.1 Switching state of the digital output

Name:
DigitalOutput01

This register controls the digital output.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	DigitalOutput01	0	Reset output
		1	Set output
1 - 7	Reserved	0	

4.5.2 Reset additional information and data point histograms

Name:
ClrStatistics_OperatingData
ClrStatistics_RelHumidity
ClrStatistics_Temperature
ClrStatistics_Acceleration01 to ClrStatistics_Acceleration03
ClrStatistics_Rotation01 to ClrStatistics_Rotation03

Setting the respective bit in the register resets operating data, information and histograms. Procedure:

- Set the bit for resetting the desired data
- The bit must remain set until the registers have been reset
- As soon as the user has determined that the data has been reset, then the bit for resetting the data can be deleted.
- If the bit for resetting the data is not deleted, the data will be permanently set to 0.

Information:

It can take up to 1 s until the delete operation for the data is executed.

Data type	Values
UINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	ClrStatistics_OperatingData Reset operating data	0	Do not reset
		1	Reset
1	ClrStatistics_RelHumidity Resets information and histograms for relative humidity	0	Do not reset
		1	Reset
2	ClrStatistics_Temperature Resets information and histograms for ambient temperature	0	Do not reset
		1	Reset
3 - 7	Reserved	0	
8	ClrStatistics_Acceleration01 Resets information and histograms for Acceleration01	0	Do not reset
		1	Reset
9	ClrStatistics_Acceleration02 Resets information and histograms for Acceleration02	0	Do not reset
		1	Reset
10	ClrStatistics_Acceleration03 Resets information and histograms for Acceleration03	0	Do not reset
		1	Reset
11	ClrStatistics_Rotation01 Resets information and histograms for Rotation01	0	Do not reset
		1	Reset
12	ClrStatistics_Rotation02 Resets information and histograms for Rotation02	0	Do not reset
		1	Reset
13	ClrStatistics_Rotation03 Resets information and histograms for Rotation03	0	Do not reset
		1	Reset
14 - 15	Reserved	0	

4.6 Communication

4.6.1 Status of digital inputs, digital output and I/O power supply

Name:
DigitalInput01
DigitalInput02
StateDigitalOutput01
PowerSupply

This register is used to indicate the status of the digital inputs, the digital output and the I/O power supply voltage.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	DigitalInput01	0 or 1	Input state - Digital input 1
1	DigitalInput02	0 or 1	Input state - Digital input 2
2 - 3	Reserved	-	
4	StateDigitalOutput01	0 or 1	Output state digital output 1
5 - 6	Reserved	-	
7	PowerSupply State of I/O power supply voltage	0	I/O power supply in the permissible range: 24 VDC -15% / +20%
		1	I/O power supply outside the permissible range

4.6.2 Module status

Name:
ModuleState01

Status register for monitoring analog inputs.

Data type	Values
UINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0 - 1	Channel 1	00	No error
		01	Range undershoot
		10	Range overshoot
		11	Open circuit
2 - 3	Channel 2	00	No error
		01	Range undershoot
		10	Range overshoot
		11	Open circuit
4 - 7	Reserved	-	

4.7 Measured values

4.7.1 Relative humidity

Name:
RelHumidity

An internal sensor measures the relative humidity in the area.

Data type	Values	Information
INT	0 to 100	Relative humidity [%], resolution 1%

4.7.2 Ambient temperature

Name:
Temperature

An internal sensor measures the ambient temperature.

Data type	Values	Information
INT	-250 to 1250	Ambient temperature [°C], resolution 0.1°C

4.7.3 Acceleration

Name:
Acceleration01 to Acceleration03

An internal sensor measures the acceleration.

Data type	Values	Information
INT	-32768 to 32767	Acceleration as raw value. The conversion must be carried out in the application: • 16 g = 32767 • -16 g = -32768

4.7.4 Rotation

Name:
Rotation01 to Rotation03

An internal sensor measures the rotation.

Data type	Values	Information
INT	-32768 to 32767	Rotation as raw value. The conversion must be carried out in the application: • 2000 dps = 32767 • -2000 dps = -32768

4.7.5 Analog inputs

Name:
TempExt01 to TempExt02

This register contains the analog input values. The input filter and input ramp limiting are set with register "CfO_AnFilter" on page 50.

Data type	Digital value	Input signal
INT	-400 to 1250 (resolution 0.1°C)	Sensor type PT1000, Temperature measurement -40.0 to 125.0°C

4.8 Additional information

Information:

The following points must be observed:

- **Data recorded on the module is saved in intervals of 10 s.**
- **When reseting the values, it can take up to 1 s until the delete operation is executed (see register "ClrStatistics" on page 51).**

4.8.1 Operating data

Name:
OnTimeConnected
OnTimeDisconnected
OnTimeCombined
PowerCycles

The respective operating data is output in these registers. If needed, the values can be reset using register "ClrStatistics" on page 51.

Register	Data type	Values	Information
OnTimeConnected	UDINT	0 to 4,294,967,295	Operating time during which the module was actively connected to the network master [s], resolution 1 s
OnTimeDisconnected	UDINT	0 to 4,294,967,295	Operating time during which the module was not actively connected to the network master [s] (blackout mode), resolution 1 s
OnTimeCombined	UDINT	0 to 4,294,967,295	Total operating time of the module [s], resolution 1 s
PowerCycles	UDINT	0 to 4,294,967,295	Number of power-on cycles

4.8.2 Relative humidity

Name:
RelHumidityMin
RelHumidityMax

Information about the relative humidity is output in these registers. The sampling interval is 1 s. If needed, the values can be reset using register "ClrStatistics" on page 51.

Register	Data type	Values	Information
RelHumidityMin	INT	0 to 100	Smallest value that occurred [%], resolution 1%
RelHumidityMax	INT	0 to 100	Largest value that occurred [%], resolution 1%

4.8.3 Ambient temperature

Name:
TemperatureMin
TemperatureMax

Information about the ambient temperature is output in these registers. The sampling interval is 1 s. If needed, the values can be reset using register "ClrStatistics" on page 51.

Register	Data type	Values	Information
TemperatureMin	INT	-250 to 1250	Lowest value that occurred [°C], resolution 0.1°C
TemperatureMax	INT	-250 to 1250	Highest value that occurred [°C], resolution 0.1°C

4.8.4 Acceleration

Name:
Acceleration01Min to Acceleration03Min
Acceleration01Max to Acceleration03Max

Information about acceleration is output in these registers. The sampling interval is typ. 10 ms. If needed, the values can be reset using register "ClrStatistics" on page 51.

The acceleration is provided as a raw value. The conversion must be performed in the application.

Register	Data type	Values	Information
Acceleration01Min to Acceleration03Min	INT	-32768 to 32767	Smallest value occurred
Acceleration01Max to Acceleration03Max	INT	-32768 to 32767	Largest value occurred

4.8.5 Rotation

Name:
Rotation01Min to Rotation03Min
Rotation01Max to Rotation03Max

Information about rotation is output in these registers. The sampling interval is typ. 10 ms. If needed, the values can be reset using register "ClrStatistics" on page 51.

The rotation is provided as a raw value. The conversion must be performed in the application.

Register	Data type	Values	Information
Rotation01Min to Rotation03Min	INT	-32768 to 32767	Smallest value occurred
Rotation01Max to Rotation03Max	INT	-32768 to 32767	Largest value occurred

4.9 Data point histogram

4.9.1 Relative humidity

Name:
RelHumHist01Entry to RelHumHist10Entry
RelHumHist01Time to RelHumHist10Time

The relative humidity histogram data recorded on the module is displayed on these registers. If needed, the values can be reset using register "ClrStatistics" on page 51.

Register	Data type	Values	Information
RelHumHist01Entry to RelHumHist10Entry	UDINT	0 to 4,294,967,295	Entry counter for the corresponding area
RelHumHist01Time to RelHumHist10Time	UDINT	0 to 4,294,967,295	Dwell time in corresponding area [s], resolution 1 s

4.9.2 Ambient temperature

Name:
TempHist01Entry to TempHist12Entry
TempHist01Time to TempHist12Time

The ambient temperature histogram data recorded on the module is displayed on these registers. If needed, the values can be reset using register "ClrStatistics" on page 51.

Register	Data type	Values	Information
TempHist01Entry to TempHist12Entry	UDINT	0 to 4,294,967,295	Entry counter for the corresponding area
TempHist01Time to TempHist12Time	UDINT	0 to 4,294,967,295	Dwell time in corresponding area [s], resolution 1 s

4.9.3 Acceleration

Name:
AccHist0101Entry to AccHist0108Entry
AccHist0101Time to AccHist0108Time
AccHist0201Entry to AccHist0208Entry
AccHist0201Time to AccHist0208Time
AccHist0301Entry to AccHist0308Entry
AccHist0301Time to AccHist0308Time

The acceleration histogram data recorded on the module is displayed on these registers. If needed, the values can be reset using register "ClrStatistics" on page 51.

Register	Data type	Values	Information
AccHist0N01Entry to AccHist0N08Entry	UDINT	0 to 4,294,967,295	Entry counter for the corresponding area
AccHist0N01Time to AccHist0N08Time	UDINT	0 to 4,294,967,295	Dwell time in corresponding area [s], resolution 1 s

Legend: N = 1 to 3

4.9.4 Rotation

Name:
RotationHist0101Entry to RotationHist0108Entry
RotationHist0101Time to RotationHist0108Time
RotationHist0201Entry to RotationHist0208Entry
RotationHist0201Time to RotationHist0208Time
RotationHist0301Entry to RotationHist0308Entry
RotationHist0301Time to RotationHist0308Time

The rotation histogram data recorded on the module is displayed on these registers. If needed, the values can be reset using register "ClrStatistics" on page 51.

Register	Data type	Values	Information
RotationHist0N01Entry to RotationHist0N08Entry	UDINT	0 to 4,294,967,295	Entry counter for the corresponding area
RotationHist0N01Time to RotationHist0N08Time	UDINT	0 to 4,294,967,295	Dwell time in corresponding area [s], resolution 1 s

Legend: N = 1 to 3

4.10 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.

4.10.1 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.

4.10.2 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

4.10.3 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.

4.10.4 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.

4.10.5 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		

4.10.6 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

4.10.7 Delay time

Name:
ForwardDelay

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

Data type	Values
UINT	0 to 65535 [µs] Default: 0

4.11 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

4.12 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	
Temperature and relative humidity	1 s
Acceleration and rotation	Typ. 10 ms
Digital inputs	100 µs without filtering 200 µs with filtering
Digital output	Equal to the minimum cycle time
Analog inputs	1 ms
State of I/O power supply voltage	<10 ms
User flash Flatstream communication	<10 ms