X20(c)AI2438

1 General information

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

Each current measurement input has its own sensor supply. The two channels with their respective sensor supplies are electrically isolated from each other. The user can select between the two measurement ranges 4 to 20 mA and 0 to 25 mA.

- 2 analog current measurement inputs
- Integrated HART protocol
- Supports HART variables
- Electrically isolated analog channels
- Electrically isolated sensor supplies
- 16-bit digital converter resolution
- NetTime timestamp: Moment of measurement, HART image

NetTime timestamp of the measurement

For many applications, not only the measured value is important, but also the exact time of the measurement. The module is equipped with a NetTime timestamp function for this that supplies a timestamp for the recorded position and trigger time with microsecond accuracy.

The timestamp function is based on synchronized timers. If a timestamp event occurs, the module immediately saves the current NetTime. After the respective data is transferred to the controller, including this precise moment, the controller can then evaluate the data using its own NetTime (or system time), if necessary.

1.1 Coated modules

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

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

The coating has been certified according to the following standards:

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

1.1.1 Starting temperature

The starting temperature describes the minimum permissible ambient temperature in a voltage-free state at the time the coated module is switched on. This is permitted to be as low as -40°C. During operation, the conditions as specified in the technical data continue to apply.

> **Information:**
>
> **It is important to absolutely ensure that there is no forced cooling by air currents in the closed control cabinet, e.g. due to the use of a fan or ventilation slots.**

1.2 Other applicable documents

For additional and supplementary information, see the following documents.

Other applicable documents

Document name	Title
MAX20	X20 system user's manual
MAEMV	Installation / EMC guide

2 Order data

Order number	Short description	Figure
	Analog inputs	
X20AI2438	X20 analog input module, 2 inputs, 4 to 20 mA, 16-bit converter resolution, single-channel galvanically isolated and with own sensor power supply, supports HART protocol, NetTime function	
X20cAI2438	X20 analog input module, coated, 2 inputs, 4 to 20 mA, 16-bit converter resolution, single-channel galvanically isolated and with own sensor power supply, supports HART protocol, NetTime function	
	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: X20AI2438, X20cAI2438 - Order data

3 Technical description

3.1 Technical data

Order number	X20AI2438	X20cAI2438
Short description		
I/O module	2 analog inputs 4 to 20 mA or 0 to 25 mA	
General information		
B&R ID code	0xB3A9	0xE1EE
Status indicators	I/O function per channel, operating state, module status, sensor power supply per channel, HART	
Diagnostics		
Module run/error	Yes, using LED status indicator and software	
Inputs	Yes, using LED status indicator and software	
Sensor power supply	Yes, using LED status indicator and software	
HART link	Yes, using LED status indicator and software	
HART error	Yes, using LED status indicator and software	
Power consumption		
Bus	0.05 W	
Internal I/O	1.15 W [1)]	
Additional power dissipation caused by actuators (resistive) [W]	-	

Table 2: X20AI2438, X20cAI2438 - Technical data

Order number	X20AI2438	X20cAI2438
Certifications		
CE	Yes	
UKCA	Yes	
ATEX	Zone 2, II 3G Ex nA nC IIA T5 Gc IP20, Ta (see X20 user's manual) FTZÚ 09 ATEX 0083X	
UL	cULus E115267 Industrial control equipment	
HazLoc	cCSAus 244665 Process control equipment for hazardous locations Class I, Division 2, Groups ABCD, T5	
DNV	Temperature: **B** (0 - 55°C) Humidity: **B** (up to 100%) Vibration: **B** (4 g) EMC: **B** (bridge and open deck)	
LR	ENV1	
KR	Yes	
ABS	Yes	
EAC	Yes	
KC	Yes	-
Analog inputs		
Input	4 to 20 mA or 0 to 25 mA configurable using software	
Input type	Differential input	
Digital converter resolution	16-bit	
Data output rate		
With HART	4.7 to 10 samples per second, configurable using software	
Analog	4.7 to 100 samples per second, configurable using software	
Output format	INT	
Output format		
4 to 20 mA	INT 0x0000 - 0x7FFF / 1 LSB = 0x0001 = 488.281 nA	
0 to 25 mA	INT 0x0000 - 0x7FFF / 1 LSB = 0x0001 = 762.939 nA	
0 to 25000 µA	INT 0x0000 - 0x61A8 / 1 LSB = 0x0001 = 1000 nA	
Load	I_IN ≥ 0.1 mA: R < 8000 Ω I_IN ≥ 1 mA: R < 1100 Ω I_IN ≥ 4 mA: R < 510 Ω	
Input protection	Up to 30 VDC, reverse polarity protection (max. 0.1 A)	
Open-circuit detection	Yes, using software	
Permissible input signal	0 to 25 mA	
Output of digital value during overload	Configurable	
Conversion procedure	Sigma-delta	
Max. error		
Gain		
0 to 25 mA	<0.046% [2)]	
4 to 20 mA	<0.046% [2)]	
Offset		
0 to 25 mA	<0.004% [3)]	
4 to 20 mA	<0.013% [3)]	
Common-mode rejection		
DC	80 dB	
50 Hz	Depends on the sampling rate: e.g. >130 dB for 50 samples per second	
Common-mode range	0 to 7 V	
Nonlinearity	<0.003% [3)]	
Input filter		
Hardware	First-order low-pass filter / cutoff frequency 100 Hz	
Software	Sinc⁴ filter	
Max. gain drift		
0 to 25 mA	0.003 %/°C [2)]	
4 to 20 mA	0.003 %/°C [2)]	
Max. offset drift		
0 to 25 mA	0.0002 %/°C [3)]	
4 to 20 mA	0.0007 %/°C [3)]	
Test voltage		
Channel - Channel	1000 VAC	
Channel - Bus	1000 VAC	
Channel - Ground	1000 VAC	
Sensor power supply		
Power consumption	0.75 W per channel	
Nominal voltage	25 V ±2%	
Nominal output current	Max. 30 mA	
Short-circuit proof	Yes, continuous	

Table 2: X20AI2438, X20cAI2438 - Technical data

Order number	X20AI2438	X20cAI2438
Max. voltage ripple		
Up to 100 kHz	≤2.2 mV	
Up to 1 MHz	≤22 mV	
Higher	≤100 mV	
Short-circuit current		
Typical	<50 mA	
Maximum	60 mA	
Behavior on short circuit	Current limiting	
HART		
Transfer rate	1200 bit/s	
Operating frequencies	1200 Hz / 2200 Hz	
Multi-drop operation		
Possible	Yes	
Stations	5 nodes (when using HART slaves with a nominal current of 4 mA) Up to 15 (taking into account the maximum permissible input signal of 25 mA)	
Burst operation possible	Yes	
Transmission amplitude		
Minimum	400 mV_{pp}	
Typical	500 mV_{pp}	
Maximum	600 mV_{pp}	
Receiving amplitude		
Minimum	120 mV_{pp}	
Maximum	800 mV_{pp}	
Electrical properties		
Electrical isolation	Channel isolated from channel and bus Sensor power supply isolated from sensor power supply Sensor power supply not isolated from channel	
Operating conditions		
Mounting orientation		
Horizontal	Yes	
Vertical	Yes	
Installation elevation above sea level		
0 to 2000 m	No limitation	
>2000 m	Reduction of ambient temperature by 0.5°C per 100 m	
Degree of protection per EN 60529	IP20	
Ambient conditions		
Temperature		
Operation		
Horizontal mounting orientation	-25 to 60°C	
Vertical mounting orientation	-25 to 50°C	
Derating	-	
Starting temperature	-	Yes, -40°C
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: X20AI2438, X20cAI2438 - Technical data

1) To reduce power dissipation, B&R recommends leaving unused inputs open.
2) Based on the current measured value.
3) Based on the 25 mA measurement range.

3.2 LED status indicators

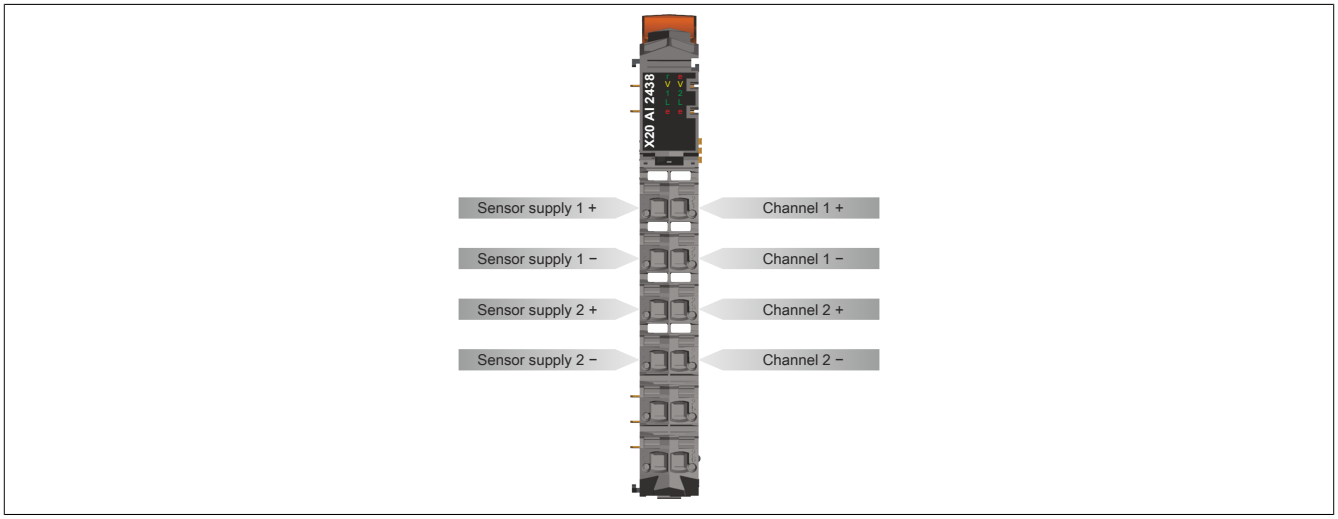
For a description of the various operating modes, see section "Additional information - Diagnostic LEDs" in the X20 system user's manual.

Figure	LED	Color	Status	Description
	**Operating state**			
	r	Green	Off	No power to module
			Single flash	UNLINK mode
			Double flash	BOOT mode (during firmware update)[1]
			Blinking quickly	SYNC mode
			Blinking slowly	Mode PREOPERATIONAL
			On	RUN mode
	Module status			
	e	Red	Off	No power to module or everything OK
			Single flash	A conversion error has occurred. This status is output along with a double flash on the channel LED of the analog input where the error occurs.
			On	Error or reset status
	Sensor supply			
	V	Yellow	Off	Module supply not connected or overload
			On	Sensor supply in its normal operating range
	Analog input			
	1 - 2	Green	Off	Indicates one of the following cases: • No power to module • Channel disabled • Open line
			Single flash	Input signal overflow or underflow
			Double flash	A conversion error has occurred. A single flash is output on the red "e" module status LED.
			On	Analog/digital converter running, value OK
	HART link			
	L	Green	Off	Indicates one of the following cases: • No power to module • HART disabled for the respective channel
			Flickering	Carrier signal active (DCD or RTS)
	HART error			
	e	Red	Off	Indicates one of the following cases: • Communication taking place without errors • No power to module • HART disabled for the respective channel
			On	Communication error

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

3.3 Pinout

Shielded twisted pair cables should be used to minimize coupling disturbances. Use either one cable for each channel or a multiple twisted pair cable for both channels.

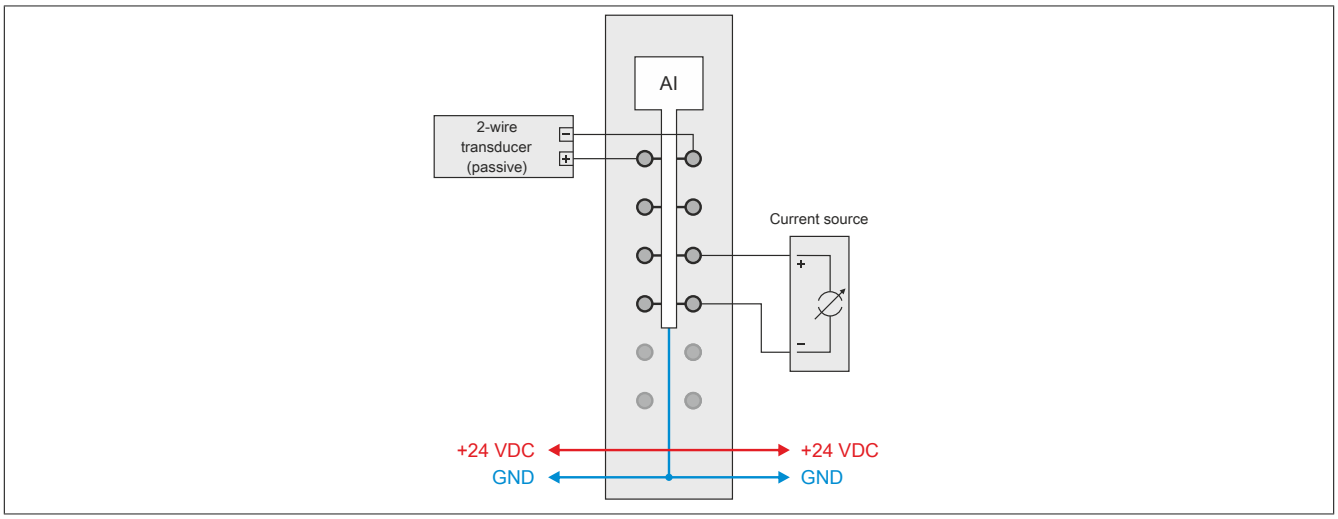


3.4 Connection examples

2-wire connections

A 2-wire connection can be implemented as follows:

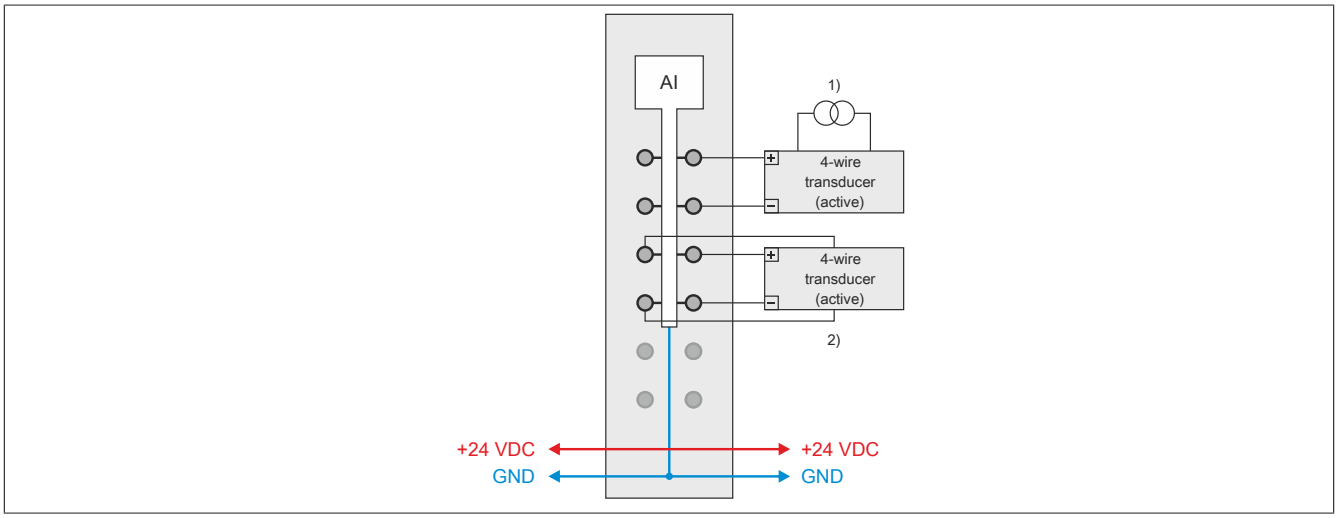
- 2-wire transducer
- Active current source



4-wire connections

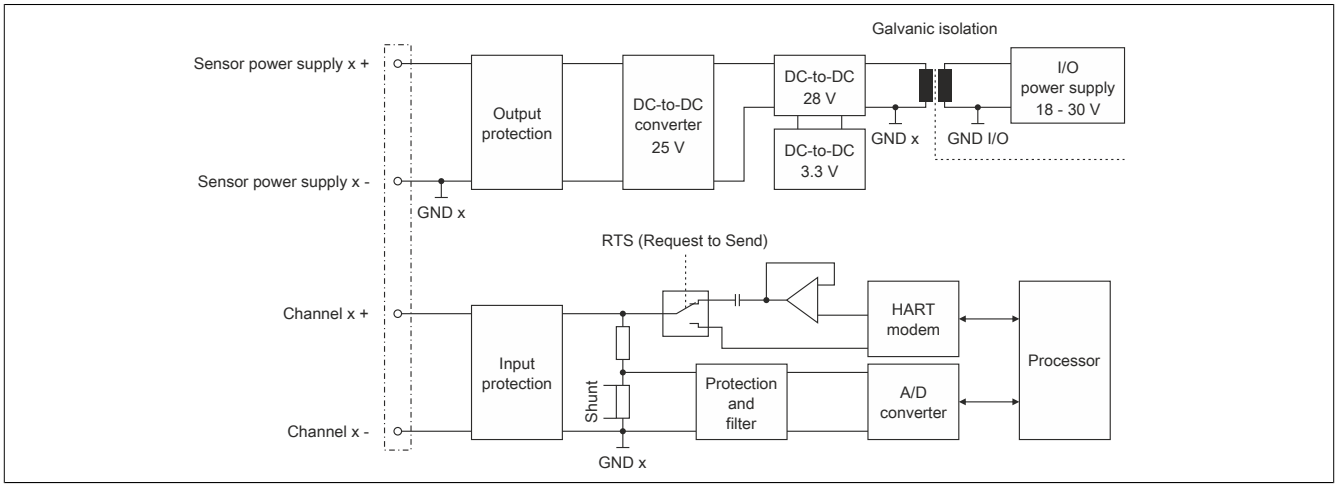
A 4-wire connection can be implemented as follows:

- 4-wire transducer with external supply
- 4-wire transducer supplied by the module



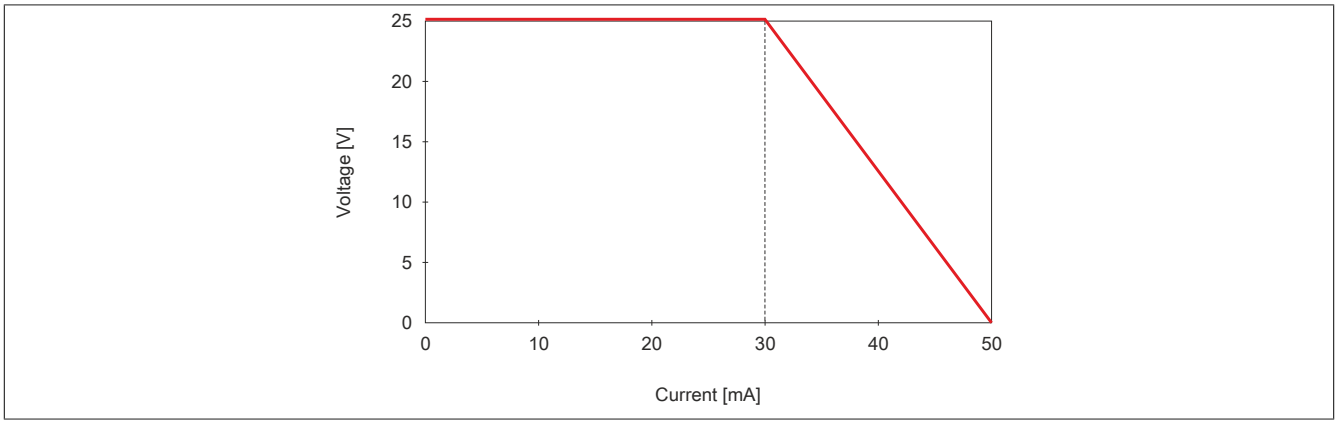
1) With external power supply.
2) With internal power supply. The internal power supply is only permitted to be loaded with max. 30 mA.

3.5 Input circuit diagram



3.6 Behavior in the event of short circuit

In the event of a short circuit, the output current for the sensor supply is limited according to the following diagram.



3.7 Usage after the X20IF1091-1

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

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 Register overview - Function model 0 (standard)

Register	Name	Data type	Read		Write	
			Cyclic	Acyclic	Cyclic	Acyclic
Analog signal - Configuration						
386 426	AnMode_1 AnMode_2	UINT				•
390 430	Samplerate_1 Samplerate_2	UINT				•
394 434	OpenLoopLimit_1 OpenLoopLimit_2	(U)INT				•
398 438	LowerLimit_1 LowerLimit_2	(U)INT				•
402 442	UpperLimit_1 UpperLimit_2	(U)INT				•
406 446	Hysteres_1 Hysteres_2	(U)INT				•
410 450	ReplacementLower_1 ReplacementLower_2	(U)INT				•
414 454	ReplacementUpper_1 ReplacementUpper_2	(U)INT				•
418 458	ErrorDelay_1 ErrorDelay_2	UINT				•
422 462	SumErrorDelay_1 SumErrorDelay_2	UINT				•
466 482	PreparationInterval_1 PreparationInterval_2	UINT				•
Analog signal - Communication						
266 270	AnalogInput01 (if replacement value strategy on) AnalogInput02 (if replacement value strategy on)	(U)INT	•			
258 262	AnalogInput01 (if replacement value strategy off) AnalogInput02 (if replacement value strategy off)	(U)INT	•			
282 290	AnalogSampletime01 (16-bit) AnalogSampletime02 (16-bit)	INT	•			
284 292	AnalogSampletime01 (32-bit) AnalogSampletime02 (32-bit)	DINT	•			
30 31	AnalogStatus01 AnalogStatus02	USINT	•			
	UnderflowAnalogInput01 or 02	Bit 0				
	OverflowAnalogInput01 or 02	Bit 1				
	OpenLineAnalogInput01 or 02	Bit 2				
	ConversionErrorAnalogInput01 or 02	Bit 3				
	SumErrorAnalogInput01 or 02	Bit 4				
	SensorErrorAnalogInput01 or 02	Bit 6				
	IoSuppErrorAnalogInput01 or 02	Bit 7				
HART - Configuration						
1537 1665	HartNodeCnt_1 HartNodeCnt_2	USINT				•
1539 1667	HartMode_1 HartMode_2	USINT				•
1541 1669	HartBurstNode_1 HartBurstNode_2	USINT				•
HART - Extended configuration						
1558 1686	HartNodeDisable_1 HartNodeDisable_2	UINT				•
1546 1674	HartProtTimeOut_1 HartProtTimeOut_2	UINT				•
1550 1678	HartProtRetry_1 HartProtRetry_2	UINT				•
1554 1682	HartPreamble_1 HartPreamble_2	UINT				•
HART - Communication (P2P)						
612 + N*24 1124 + N*24	PvInput01_0N (index N = 1 to 4) PvInput02_0N (index N = 1 to 4)	REAL	•	•		
617 + N*24 1129 + N*24	PvUnit01_0N (index N = 1 to 4) PvUnit02_0N (index N = 1 to 4)	USINT	•	•		
628 1140	PvSampleTime01 PvSampleTime02	DINT	•	•		

Register	Name	Data type	Read		Write	
			Cyclic	Acyclic	Cyclic	Acyclic
626 1138	PvSampleTime01 PvSampleTime02	INT	•			
566 1078	PvNodeComStatus01 PvNodeComStatus02	DINT		•		
HART - Communication (multidrop)						
612 + N*24 1124 + N*24	PvInput01_N (index N = 01 to 15) PvInput02_N (index N = 01 to 15)	REAL	•	•		
617 + N*24 1129 + N*24	PvUnit01_N (index N = 01 to 15) PvUnit02_N (index N = 01 to 15)	USINT	•	•		
604 + N*24 1116 + N*24	PvSampleTime01_N (index N = 01 to 15) PvSampleTime02_N (index N = 01 to 15)	DINT	•	•		
602 + N*24 1114 + N*24	PvSampleTime01_N (index N = 01 to 15) PvSampleTime02_N (index N = 01 to 15)	INT	•			
562 + N*4 1074 + N*4	PvNodeComStatus01_N (index N = 01 to 15) PvNodeComStatus02_N (index N = 01 to 15)	DINT		•		
HART - Extended communication						
522 1034	PvCountHartRequest01 PvCountHartRequest02	UINT	•			
530 1042	PvCountHartTimeout01 PvCountHartTimeout02	UINT	•			
538 1050	PvCountHartRxError01 PvCountHartRxError02	UINT	•			
546 1058	PvCountHartFrameError01 PvCountHartFrameError02	UINT	•			
554 1066	PvNodeFound01 PvNodeFound02	UINT	•			
558 1070	PvNodeError01 PvNodeError02	UINT	•			
Flatstream - Configuration						
1793	OutputMTU	USINT				•
1795	InputMTU	USINT				•
1797	FlatstreamMode	USINT				•
1799	Forward	USINT				•
1801	ForwardDelay	UINT				•
Flatstream - Communication						
1857	InputSequence	USINT	•			
1857 + N*2	RxByteN (index N = 1 to 15)	USINT	•			
1889	OutputSequence	USINT			•	
1889 + N*2	TxByteN (index N = 1 to 15)	USINT			•	

4.3 Register overview - Function model 254 (bus controller)

Register	Offset[1)]	Name	Data type	Read		Write	
				Cyclic	Acyclic	Cyclic	Acyclic
Analog signal - Configuration							
386 426	- -	AnMode_1 AnMode_2	UINT				•
390 430	- -	Samplerate_1 Samplerate_2	UINT				•
394 434	- -	OpenLoopLimit_1 OpenLoopLimit_2	(U)INT				•
398 438	- -	LowerLimit_1 LowerLimit_2	(U)INT				•
402 442	- -	UpperLimit_1 UpperLimit_2	(U)INT				•
406 446	- -	Hysteres_1 Hysteres_2	(U)INT				•
410 450	- -	ReplacementLower_1 ReplacementLower_2	(U)INT				•
414 454	- -	ReplacementUpper_1 ReplacementUpper_2	(U)INT				•
418 458	- -	ErrorDelay_1 ErrorDelay_2	UINT				•
422 462	- -	SumErrorDelay_1 SumErrorDelay_2	UINT				•
466 482	- -	PreparationInterval_1 PreparationInterval_2	UINT				•
Analog signal - Communication							
266 270	0 8	AnalogInput01 (if replacement value strategy on) AnalogInput02 (if replacement value strategy on)	(U)INT	•			
258 262	- -	AnalogInput01 (if replacement value strategy off) AnalogInput02 (if replacement value strategy off)	(U)INT		•		

Register	Offset[1)]	Name	Data type	Read		Write	
				Cyclic	Acyclic	Cyclic	Acyclic
30 31	- -	AnalogStatus01 AnalogStatus02	USINT		•		
		UnderflowAnalogInput01 or 02	Bit 0				
		OverflowAnalogInput01 or 02	Bit 1				
		OpenLineAnalogInput01 or 02	Bit 2				
		ConversionErrorAnalogInput01 or 02	Bit 3				
		SumErrorAnalogInput01 or 02	Bit 4				
		SensorErrorAnalogInput01 or 02	Bit 6				
		IoSuppErrorAnalogInput01 or 02	Bit 7				
HART - Configuration							
1537 1665	- -	HartNodeCnt_1 HartNodeCnt_2	USINT				•
1539 1667	- -	HartMode_1 HartMode_2	USINT				•
1541 1669	- -	HartBurstNode_1 HartBurstNode_2	USINT				•
HART - Extended configuration							
1558 1686	- -	HartNodeDisable_1 HartNodeDisable_2	UINT				•
1546 1674	- -	HartProtTimeOut_1 HartProtTimeOut_2	UINT				•
1550 1678	- -	HartProtRetry_1 HartProtRetry_2	UINT				•
1554 1682	- -	HartPreamble_1 HartPreamble_2	UINT				•
HART - Communication (P2P)							
636 1148	4 12	PvInput01_01 PvInput02_01	REAL	•			
612 + N*24 1124 + N*24	- -	PvInput01_0N (index N = 2 to 4) PvInput02_0N (index N = 2 to 4)	REAL		•		
641 1153	2 10	PvUnit01_01 PvUnit02_01	USINT	•			
617 + N*24 1129 + N*24	- -	PvUnit01_0N (index N = 2 to 4) PvUnit02_0N (index N = 2 to 4)	USINT		•		
566 1078	- -	PvNodeComStatus01 PvNodeComStatus02	DINT		•		
HART - Communication (multidrop)							
636 1148	4 12	PvInput01_01 PvInput02_01	REAL	•			
612 + N*24 1124 + N*24	- -	PvInput01_N (index N = 02 to 15) PvInput02_N (index N = 02 to 15)	REAL		•		
641 1153	2 10	PvUnit01_01 PvUnit02_01	USINT	•			
617 + N*24 1129 + N*24	- -	PvUnit01_N (index N = 02 to 15) PvUnit02_N (index N = 02 to 15)	USINT		•		
562 + N*4 1074 + N*4	- -	PvNodeComStatus01_N (index N = 01 to 15) PvNodeComStatus02_N (index N = 01 to 15)	DINT		•		
HART - Extended communication							
522 1034	- -	PvCountHartRequest01 PvCountHartRequest02	UINT		•		
530 1042	- -	PvCountHartTimeout01 PvCountHartTimeout02	UINT		•		
538 1050	- -	PvCountHartRxError01 PvCountHartRxError02	UINT		•		
546 1058	- -	PvCountHartFrameError01 PvCountHartFrameError02	UINT		•		
554 1066	- -	PvNodeFound01 PvNodeFound02	UINT		•		
558 1070	- -	PvNodeError01 PvNodeError02	UINT		•		

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

4.3.1 Using the module on the bus controller

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

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

4.3.2 CAN I/O bus controller

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

4.4 General information

This module is equipped with two independent electrically isolated channels with integrated HART modems. Both channels can be used to read in an analog signal and handle HART communication. All registers necessary for this have a dual design so that the channels can be configured and operated independently of one another.
The current input signals (0 to 25 mA) can be displayed in various formats and used as conventional analog inputs. The integrated HART modems retrieve digital information from the memory on the HART slave using the same physical lines that modulate the HART signals.

When using the 0 to 25 mA current input variant, the module is conceived as a HART master for 2 channels (loops), with FSK modulation of the HART protocol and sensor supply for up to 15 slaves per channel.

Each channel can use one of the following connection variants:

- Connection of one HART node (point-to-point) with evaluation of the analog signal and output of 4 HART process variables OR
- Connection of up to 15 HART nodes in multidrop mode with output of the primary HART variable from activated nodes

Specific features:

- Channels electrically isolated
- Up to 15 HART input variables per channel
- Configurable sampling rate (input filter) to transfer HART and analog signal without interference (default: 50 Hz or 20 ms)
- Internal supply with short circuit protection <30 mA per channel
- Selective line monitoring can be enabled for: open line (<2 mA), underflow (<3.6 mA) or overflow (>21 mA) of a configurable threshold
- Selectable error strategy (static replacement value or retention of the last permitted value)
- Cyclic "HART status" polling (HART command 0), the status information received is made available for channel diagnostics
- Compatible with an additional secondary master in the HART network (module acts as the primary master)
- "HART communication error bit" (shows loss of HART connection if a connection had already been established successfully)
- Optional: Burst mode for one node per channel
- Optional: Cyclic polling of "HART variables" (HART command 3 or 9)
- Optional: Sensor power supply for max. 15 nodes per channel in the multidrop variant
- Optional: Flatstream functionality (module acts as bridge for HART packets)

Information:

Maximum number of HART nodes per channel:

- **5 nodes (when using HART nodes with a nominal current of 4 mA)**
- **Up to 15 HART nodes (taking into account the maximum permissible input signal or nominal output current of the sensor power supply of 25 mA)**

4.5 Analog signal - Configuration

How the analog signal is displayed can be adapted to the requirements of the application. Separate configuration registers per channel are available to aid in this.

4.5.1 Channel parameters

Name:
AnMode_1 to AnMode_2

These registers are used to predefine the operating parameters that the module will be using for the respective channel. Each channel must be enabled individually and can be configured and operated independently.

> **Information:**
>
> **Different limit values must be configured for any display normalizing that needs to take place.**

Data type	Values	Bus controller default setting
UINT	See bit structure.	29

Bit structure:

Bit	Name	Value	Information
0	Channel	0	Channel 0x turned off
		1	Channel 0x enabled (bus controller default setting)
1	Open line detection	0	Open line monitoring turned off
		1	Open circuit monitoring enabled (bus controller default setting)
2	Underflow detection	0	Underflow detection turned off
		1	Underflow detection enabled (bus controller default setting)
3	Replacement value strategy	0	Use replacement values in the event of error (bus controller default setting)
		1	Keep the last valid converted value
4 - 5	Normalization	00	Displays 0 to 25 mA as 0 to 32767
		01	Display 0 to 25 mA as 0 to 25000 [µA] (bus controller default setting)
		10	Displays 4 to 20 mA as 0 to 32767
		11	Displays 0 to 25 mA as 0 to 65535
6 - 15	Reserved	-	

4.5.2 Sample rate

Name:
Samplerate_1 to Samplerate_2

A conversion rate can be configured independently for the two analog inputs. Based on the desired sampling frequency, the following formula results for this parameter:

Sampling rate for A/D converter = (4920000 / 1024) / Sampling frequency

Data type	Value	Information
UINT	4 to 1023	Sample rate **Examples of configurable values** **Value** **Time** **Frequency** 960 ... 200 ms ... 5 Hz 480 ... 100 ms ... 10 Hz 320 ... 66.7 ms ... 15 Hz 192 ... 40 ms ... 25 Hz 160 ... 33.3 ms ... 30 Hz 96 ... 20 ms ... 50 Hz (bus controller default setting) 80 ... 16.7 ms ... 60 Hz 48 ... 10 ms ... 100 Hz 9 ... 2 ms ... 500 Hz 4 ... 1 ms ... 1000 Hz

The fastest sample rate of 10 ms for the analog inputs is predefined by the cutoff frequency of the hardware filter. When using HART communication, however, a sample rate not faster than 100 ms is recommended.

4.5.3 Limit value for open line detection

Name:
OpenLoopLimit_1 to OpenLoopLimit_2

The limit value for the respective analog input must be set when open circuit monitoring is enabled and if required by the configured normalization.

Data type	Value	Information
INT	-32767 to 32767	Open circuit limit value. Bus controller default setting: 2621
UINT	0 to 65535	Open circuit limit value

If limit value monitoring is active, the corresponding error status is output after a configured delay when falling below this value. Using a default value of 2000 µA, the following values and formulas result for this parameter:

- Displays 0 to 25 mA as 0 to 25000: 2000
- Displays 0 to 25 mA as 0 to 32767: 2621, limit value = ([µA] * 32767) / 25000
- Displays 4 to 20 mA as 0 to 32767: -4096, limit value = (([µA] * 1.31068) - 5242.72) * 1.5625
- Displays 0 to 25 mA as 0 to 65535: 5243, limit value = ([µA] * 65535) / 25000

4.5.4 Preparation time for the measured values

Name:
PreparationInterval01 to PreparationInterval02

If the last valid measured value should be kept when violating the limit value, then PreparationInterval must be defined. The measured values continue to be acquired and converted according to the configured I/O update time. They are then checked and discarded if they do not meet the specifications. When an error does not occur, therefore, the measured value acquired 2 preparation intervals ago is constantly output.

Data type	Value	Information
UINT	0 to 65535	In 0.1 ms. Bus controller default setting: 0

Functionality:
Measured values are continuously converted and stored to measured value memory. The current contents of the measured value memory are checked within the configured interval. If a permissible value is present, then the contents of the buffer memory are passed to output memory and the contents of the measured value memory are passed to the buffer.
If the check turns up an impermissible value, then the contents of the measured value memory are discarded. The copy direction between output and buffer memory reverses and the last valid value continues to be output.

Information:

If configured to keep the last valid value, the delay time from measuring to outputting the value will be at least twice the preparation interval. In the worst case scenario, this can also take twice the interval time plus the configured conversion rate of the A/D converter.

"Application" Value being measured (analog)	
↓	Condition: - Conversion interval (A/D converter) elapsed
"Measured value memory" Measured value (digital)	
↓	Condition: - PreparationInterval elapsed - Measured value permissible
"Buffer" Last valid value	
↓	Condition: - PreparationInterval elapsed - Measured value permissible
"Output memory" Next-to-last valid/ displayed value	

4.5.5 Lower replacement value

Name:
ReplacementLower_1 to ReplacementLower_2

This register is used to define the lower static values to be displayed instead of the current measured value when the lower limit is violated.

Data type	Value	Information
INT	-32767 to 32767	Bus controller default setting: 4718
UINT	0 to 65535	

If the replacement strategy "Use replacement values when an error occurs" is enabled, the replacement value must be set for the respective analog input taking the configured normalization into account as well. When an overflow or underflow error status occurs, the "AnalogInput0x" on page 16 channel is replaced with the corresponding value. Using a default value of 3600 µA, the following values and formulas result for this parameter:

- Displays 0 to 25 mA as 0 to 25000: 3600
- Displays 0 to 25 mA as 0 to 32767: 4718, limit value = ([µA] * 32767) / 25000
- Displays 4 to 20 mA as 0 to 32767: -819, limit value = (([µA] * 1.31068) - 5242.72) * 1.5625
- Displays 0 to 25 mA as 0 to 65535: 9437, limit value = ([µA] * 65535) / 25000

4.5.6 Upper replacement value

Name:
ReplacementUpper_1 to ReplacementUpper_2

These registers are used to specify the upper static values that are displayed instead of the current measured value when a limit value is exceeded.

Data type	Value	Information
INT	-32767 to 32767	Bus controller default setting: 27524
UINT	0 to 65535	

If the replacement strategy "Use replacement values when an error occurs" is activated, the replacement value must be set for the respective analog input taking the configured normalization into account as well. When an overflow or underflow error status occurs, the "AnalogInput0x" on page 16 channel is replaced with the corresponding value. Using a default value of 21000 µA, the following values and formulas result for this parameter:

- Displays 0 to 25 mA as 0 to 25000: 21000
- Displays 0 to 25 mA as 0 to 32767: 27524, limit value = ([µA] * 32767) / 25000
- Displays 4 to 20 mA as 0 to 32767: 32767, limit value = (([µA] * 1.31068) - 5242.72) * 1.5625
- Displays 0 to 25 mA as 0 to 65535: 55049, limit value = ([µA] * 65535) / 25000

4.5.7 Lower limit value

Name:
LowerLimit_1 to LowerLimit_2

If the value range needs to be restricted further, this register can be used to enter new user-specific lower limit values.

Data type	Value	Information
INT	-32767 to 32767	Bus controller default setting: 4718
UINT	0 to 65535	

The limit value must be set for the respective analog input depending on the configured normalization. After the configured delay time has passed, the corresponding error status is given if the respective value is overrun or underrun. When this error state occurs, the "AnalogInput0x" on page 16 channel is evaluated according to the replacement value strategy. Using a default value of 3600 µA, the following values and formulas result for this parameter:

- Displays 0 to 25 mA as 0 to 25000: 3600
- Displays 0 to 25 mA as 0 to 32767: 4718, limit value = ([µA] * 32767) / 25000
- Displays 4 to 20 mA as 0 to 32767: -819, limit value = (([µA] * 1.31068) - 5242.72) * 1.5625
- Displays 0 to 25 mA as 0 to 65535: 9437, limit value = ([µA] * 65535) / 25000

4.5.8 Upper limit value

Name:
UpperLimit_1 to UpperLimit_2

If the value range needs to be restricted further, this register can be used to enter new user-specific upper limit values.

Data type	Value	Information
INT	-32767 to 32767	Bus controller default setting: 27524
UINT	0 to 65535	

The limit value must be set for the respective analog input depending on the configured normalization. After the configured delay time has passed, the corresponding error status is given if the respective value is overrun or underrun. When this error state occurs, the "AnalogInput0x" on page 16 channel is evaluated according to the replacement value strategy. Using a default value of 21000 µA, the following values and formulas result for this parameter:

- Displays 0 to 25 mA as 0 to 25000: 21000
- Displays 0 to 25 mA as 0 to 32767: 27524, limit value = ([µA] * 32767) / 25000
- Displays 4 to 20 mA as 0 to 32767: 32767, limit value = (([µA] * 1.31068) - 5242.72) * 1.5625
- Displays 0 to 25 mA as 0 to 65535: 55049, limit value = ([µA] * 65535) / 25000

4.5.9 Hysteresis

Name:
Hysteres_1 to Hysteres_2

If the user-specific limit values are being used, then a hysteresis range should also be defined. These registers configure how far a limit value can be exceeded before a response is triggered.

Data type	Value	Information
INT	-32767 to 32767	Bus controller default setting: 131
UINT	0 to 65535	

The hysteresis value must be set for the respective analog input depending on the configured normalization. The error status is cleared if the actual analog value changes by at least this hysteresis value from the limit value in the allowed direction. Using a default value of 100 µA, the following values and formulas result for this parameter:

- Displays 0 to 25 mA as 0 to 25000: 100
- Displays 0 to 25 mA as 0 to 32767: 131, limit value = ([µA] * 32767) / 25000
- Displays 4 to 20 mA as 0 to 32767: 156, limit value = [µA] * 1.5625
- Displays 0 to 25 mA as 0 to 65535: 262, limit value = ([µA] * 65535) / 25000

4.5.10 Delaying error messages

Name:
ErrorDelay_1 to ErrorDelay_2

This register specifies the number of consecutive conversion procedures where an error is pending until the corresponding individual error status bit is set. The delay applies to underflow, overflow and open circuit errors. This delay can be used to hide temporary measured value deviations, for example.

Data type	Value	Information
UINT	0 to 10	Error formation delay in conversion cycles. Bus controller default setting: 2

4.5.11 Time for composite error bit

Name:
SumErrorDelay_1 to SumErrorDelay_2

This register specifies the time in milliseconds that one of the individual error bits must be pending until the composite error status bit is set.

Data type	Value	Information
UINT	0 to 65535	Composite error bit delay in ms. Bus controller default setting: 4000

4.6 Analog signal - Communication

4.6.1 Analog input values

Name:
AnalogInput01 to AnalogInput02

The analog input value is mapped in this register.

Data type	Value	Information
INT	0 to 25000	Normalizing option 0 to 25 mA
	0 to 32,767	Normalizing option 0 to 25 mA
	-8192 to 32767	Normalizing option 4 to 20 mA (value 0 corresponds to 4 mA)
UINT	0 to 65535	Normalizing option 0 to 25 mA

Predefining values and timing
If a replacement value strategy was configured, value "0" (zero) is output from the beginning until a valid measured value has been calculated.

The timing of the measured value acquisition is determined by the converter hardware and the set sampling rate. The two channels are converted independently and not synchronized with the X2X Link network.

Conversion time
Channel 0x sampling rate

4.6.2 Sample time

Name:
AnalogSampletime01 to AnalogSampletime02

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

For additional information about NetTime and timestamps, see "NetTime Technology" on page 53.

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

4.6.3 Status of the inputs

Name:
AnalogStatus01 to AnalogStatus02
UnderflowAnalogInput01 to UnderflowAnalogInput02
OverflowAnalogInput01 to OverflowAnalogInput02
OpenLineAnalogInput01 to OpenLineAnalogInput02
ConversionErrorAnalogInput01 to ConversionErrorAnalogInput02
SumErrorAnalogInput01 to SumErrorAnalogInput02
SensorErrorAnalogInput01 to SensorErrorAnalogInput02
IoSuppErrorAnalogInput01 to IoSuppErrorAnalogInput02

The current error state of the module channels is indicated in this register regardless of the configured replacement value strategy. Some error information is delayed according to the previously configured condition.

Setting "Format of status information" makes it possible to define in Automation Studio whether the status information is transferred as USINT or bitwise.

Data type	Values
USINT	See bit structure.

Bit structure:

Bit	Name	Values	Information
0	UnderflowAnalogInput01 or 02	0	No error
		1	Lower limit value undershot
1	OverflowAnalogInput01 or 02	0	No error
		1	Upper limit value overshot
2	OpenLineAnalogInput01 or 02	0	No error
		1	Open circuit determined
3	ConversionErrorAnalogInput01 or 02	0	No error
		1	Conversion error determined
4	SumErrorAnalogInput01 or 02	0	No error
		1	Composite error determined
5	Reserved	-	
6	SensorErrorAnalogInput01 or 02	0	Sensor voltage OK
		1	Sensor load too high
7	IoSuppErrorAnalogInput01 or 02	0	I/O power supply OK
		1	Error in I/O power supply determined

UnderflowAnalogInput

The signal underflow error state is represented here based on the configuration. This error information is enabled as a multiple of the conversion cycles only after the configurable delay time has passed (see register "ErrorDelay" on page 15).

OverflowAnalogInput

The signal overflow error status state is represented here based on the configuration. This error information is enabled as a multiple of the conversion cycles only after the configurable delay time has passed (see register "ErrorDelay" on page 15).

OpenLineAnalogInput

Based on the configuration, the measurement information is checked for values <2 mA (register "OpenLoopLimit" on page 13) to detect a failure signal. Open circuit detection is performed using a configurable hysteresis value (default: 100 µA, register "Hysteresis" on page 15). It is possible to disable open circuit detection (register "AnalogMode" on page 12) to suppress the generation of alarms when hardware is missing. This error information is enabled as a multiple of the conversion cycles only after the configurable delay time has passed (register "ErrorDelay" on page 15).

ConversionErrorAnalogInput

This error state is triggered when the hardware overshoots the conversion time.

SumErrorAnalogInput

This error information is derived from the state of individual errors and enabled only after the configurable delay time has passed [ms] (see register "SumErrorDelay" on page 15). Linking this error information in an application makes it possible to hide temporary overshoots or undershoots of the temperature value, for example.

SensorErrorAnalogInput

This error is enabled immediately after a fault is detected in the internal sensor power supply.

IoSuppErrorAnalogInput

This error is enabled immediately after a supply voltage undershoot is detected (<20 VDC).

4.7 HART

HART (Highway Addressable Remote Transducer) is a protocol for communicating with intelligent field devices. It was developed in order to more efficiently use the infrastructure for transferring analog signals. The digital HART notifications are modulated to the analog signal using Frequency Shift Keying (FSK). HART can thus use the same physical line as the analog signal without influencing the original function.
HART slaves are able to determine different process data independently and prepare HART concordantly. This protocol supports polling of the value of a process variable as well as its unit and status. Field devices usually supply their information after the master requests it. In newer revisions, it is also possible to transfer configuration data.

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

4.7.1 HART - Configuration

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

The number of HART slaves must be specified in the configuration.
If only one slave is connected to the HART channel, then it is part of a point-to-point network. The module can then prepare up to 4 process variables from the connected slave.
Multidrop mode allows up to 15 HART slaves to be connected. The primary process variable from each slave is then retrieved.

4.7.1.1 HartBurstNode

Name:
HartBurstNode_1 to HartBurstNode_2

In addition to the type of network, the user can also choose from 2 different types of communication behavior. Conventional HART communication relies on polling. The module queries the data from the HART slave individually and receives the corresponding information from the slave as a response. If a HART node should be queried in short time intervals, the user can configure burst mode for a node on each channel. In this case, the slave transmits the information from this node cyclically without a new request by the master.
The node numbers (short address) whose information should be queried using burst mode are entered by channel in the "HartBurstNode" registers. Burst mode is enabled using register "HartMode" on page 19.

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

4.7.1.2 HartMode

Name:
HartMode_1 to HartMode_2

The user can use these registers to configure the communication behavior of each of the HART channels. Generally, the HART nodes are polled individually. This register can still be used to start or stop burst mode when needed. In burst mode, a node transmits its information cyclically instead of continuously. As a result, the HART standard allows the simultaneous usage of both burst mode and polling.

Information:

Register "HartBurstNode" on page 18 must be configured correctly for burst queries.

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

Bit structure:

Bit	Name	Value	Information
0	Slave polling mode	0	Polling mode enabled (bus controller default setting)
		1	Polling mode disabled
1	Start slave burst mode	0	No response to burst (bus controller default setting)
		1	Enables burst mode in the "HartBurstNode" on page 18 node
2	Stop slave burst mode	0	No response to burst (bus controller default setting)
		1	Disables burst mode, if enabled
3 - 7	Reserved	-	

4.7.1.3 HartNodeCnt

Name:
HartNodeCnt_1 to HartNodeCnt_2

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

Information:

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

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

4.7.2 HART - Communication

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

Overview of internal module mapping

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

The HART specifications stipulates that information from a HART node be split into various pieces. The value of a process variable is stored to the respective "PvInput" on page 20 register and has a size of 4 bytes (REAL) per the HART specification. Due to the length limitation of 30 bytes on the X2X Link network, there are limitations to the number of possible cyclic variables. It is recommended to transfer a maximum of 2 "PvInput" on page 20 registers cyclically to the X2X master. All other information should be read in a different way. To access HART information, the user can choose between the following methods:

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

  > **Information:**
  >
  > **This mode cannot be used when using the module after a bus controller.**
  >
  > **"Multiplexed" data transfer is used exclusively for HART data points.**
  >
  > **Information from the analog inputs/outputs is always transferred cyclically (see above).**

- **Flatstream** - HART modules are equipped with a Flatstream interface. When using Flatstream communication, the module is used as a bridge between the X2X master and HART slave, i.e. the X2X master communicates directly with the HART slave (see "Flatstream communication" on page 28).
  Flatstream communication is also not real-time capable. It allows unrestricted access to the HART slave. The user must have sufficient knowledge of the HART protocol command set as well as the capabilities of the corresponding HART slave.

4.7.2.1 PvInput

Name:
PvInput01_01 to PvInput01_15
PvInput02_01 to PvInput02_15

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

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

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

4.7.2.2 PvUnit

Name:
PvUnit01_01 to PvUnit01_15
PvUnit02_01 to PvUnit02_15

These registers return a HART-specific code that specifies the unit for the measured value. The coding for this is established in the HART specification.

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

4.7.2.3 PvSampleTime

Name:
PvSampleTime01 to PvSampleTime02
PvSampleTime01_01 to PvSampleTime01_15
PvSampleTime02_01 to PvSampleTime02_15

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

For additional information about NetTime and timestamps, see "NetTime Technology" on page 53.

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

This refers to the point in time when the HART master receives the slave's response. This is a way to check whether new HART information has been read since the last X2X cycle.

Information:

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

4.7.2.4 PvNodeComStatus

Name:
PvNodeComStatus01 to PvNodeComStatus02
PvNodeComStatus01_01 to PvNodeComStatus01_15
PvNodeComStatus02_01 to PvNodeComStatus02_15

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

These registers can be checked before further processing information in temporary storage. If the current value is 0x0000, an error was not detected during the HART transfer and the information from the checked node can be used. If a different value is present, the situation in the HART network should be checked. This can be done using an extension register, for example.

Data type	Values
USINT	See the bit structure.

Bit structure:

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

HART-specific response code (excerpt):

0x82 … Receive buffer overflow 0x88 … Checksum incorrect 0x90 … Faulty protocol structure 0xA0 … Overrun 0xC0 … Parity not allowed 0xFF … Timeout	If a HART communication error occurs, the response code is written. Bit 15 is always set.

Retrieving information that has been read

After the node data has been transferred to the module registers, the information can be retrieved from the module. A separate register in the module is implemented for each piece of information.

4.7.2.5 PvCountHartRequest

Name:
PvCountHartRequest01 to PvCountHartRequest02

This register is increased as soon as the module is ready to transmit a message on the corresponding channel.

Data type	Value
UINT	0 to 65535

4.7.2.6 PvCountHartTimeout

Name:
PvCountHartTimeout01 to PvCountHartTimeout02

This register is increased if the slave exceeds the maximum permissible time to respond a request from the module.

Data type	Value
UINT	0 to 65535

4.7.2.7 PvCountHartRxError

Name:
PvCountHartRxError01 to PvCountHartRxError02

This register is increased if communication errors occur on layer 1 of the OSI model (e.g. transfer error according to the parity bit).

Data type	Value
UINT	0 to 65535

4.7.2.8 PvCountHartFrameError

Name:
PvCountHartFrameError01 to PvCountHartFrameError02

This register is increased if communication errors occur on layer 2 of the OSI model (e.g. invalid telegram structure).

Data type	Value
UINT	0 to 65535

4.7.2.9 PvNodeFound

Name:
PvNodeFound01 to PvNodeFound02

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

Data type	Values
UINT	See the bit structure.

Bit structure:

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

4.7.2.10 PvNodeError

Name:
PvNodeError01 to PvNodeError02

These registers contain the HART communications error bits. These bits are set if the connection to a node was established successfully but the node at some point no longer responds as it should (e.g. the HART slave exceeds the configured timeout / number of retries).

Data type	Values
UINT	See the bit structure.

Bit structure:

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

4.7.3 Extended configuration

The additional configuration registers are specified values when the module is started. In most systems, the user does not need to make any adjustments here. Register values should only be changed if HART network communication is not taking place satisfactorily.

4.7.3.1 HartNodeDisable

Name:
HartNodeDisable_1 to HartNodeDisable_2

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

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

Bit structure:

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

4.7.3.2 HartProtTimeOut

Name:
HartProtTimeOut_1 to HartProtTimeOut_2

These registers specify the time span within which the slave must respond for the response to be valid.

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

4.7.3.3 HartProtRetry

Name:
HartProtRetry_1 to HartProtRetry_2

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

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

4.7.3.4 HartPreamble

Name:
HartPreamble_1 to HartPreamble_2

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

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

4.7.4 HART with Flatstream

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

HART is considered a master-slave network where half-duplex communication takes place asynchronously. Various features have been included to ensure that signals are transmitted without errors.
For example, the user can increase the length of the preamble, thus making the transmission more secure. However, this also has an effect on the percentage of payload data and overhead.

Additional information about HART can be found at www.HARTcomm.org.

How it works

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

Input/Output sequence	Tx/Rx bytes		
(unchanged)	Control byte (unchanged)	Channel number	HART frame (without preamble and checksum)

HART frame with Flatstream					
Startup	ADDR	CMD	BCNT	(STS)	(DATA)

Startup Start identification
ADDR Address within the HART network
CMD HART command
BCNT Byte counters (number of remaining bytes)
*STS Status of the last command received. Information about the working mode of the HART Slave and communication errors (if supported, return data from the HART Slave)
*DATA Data (if necessary for the command)

Examples of HART commands

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

4.8 Flatstream registers

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

> **Information:**
>
> **For detailed information about Flatstream, see "Flatstream communication" on page 28.**

4.8.1 Number of enabled Tx and Rx bytes

Name:
OutputMTU
InputMTU

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

Data type	Values
USINT	See the register overview.

4.8.2 Transport of 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

4.8.3 Communication status of the controller

Name:
OutputSequence

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

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0 - 2	OutputSequenceCounter	0 - 7	Counter for the sequences issued in the output direction
3	OutputSyncBit	0	Output direction disabled
		1	Output direction enabled
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.

4.8.4 Communication status of the module

Name:
InputSequence

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

Data type	Values
USINT	See the bit structure.

Bit structure:

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

InputSequenceCounter

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

InputSyncBit

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

OutputSequenceAck

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

OutputSyncAck

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

4.8.5 Flatstream mode

Name:
FlatstreamMode

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

Data type	Values
USINT	See the bit structure.

Bit structure:

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

4.8.6 Number of unacknowledged sequences

Name:
Forward

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

Recommendation:
X2X Link: Max. 5
POWERLINK: Max. 7

Data type	Values
USINT	1 to 7 Default: 1

4.8.7 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

4.9 Flatstream communication

4.9.1 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.

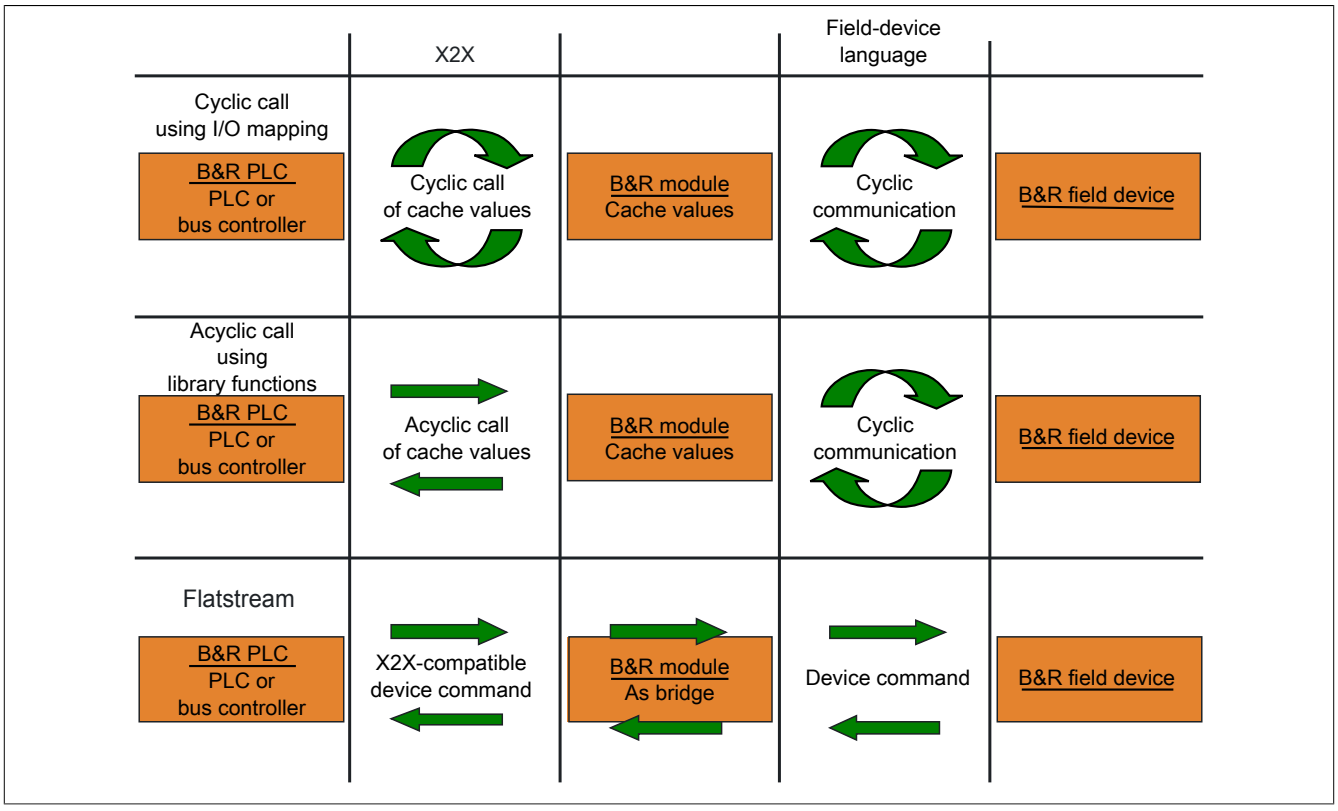


Figure 1: 3 types of communication

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.

4.9.2 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 put back 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.

4.9.3 The Flatstream principle

Requirement

Before Flatstream can be used, the respective communication direction must be synchronized, i.e. both communication partners cyclically query the sequence counter on the opposite 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 opposite 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.

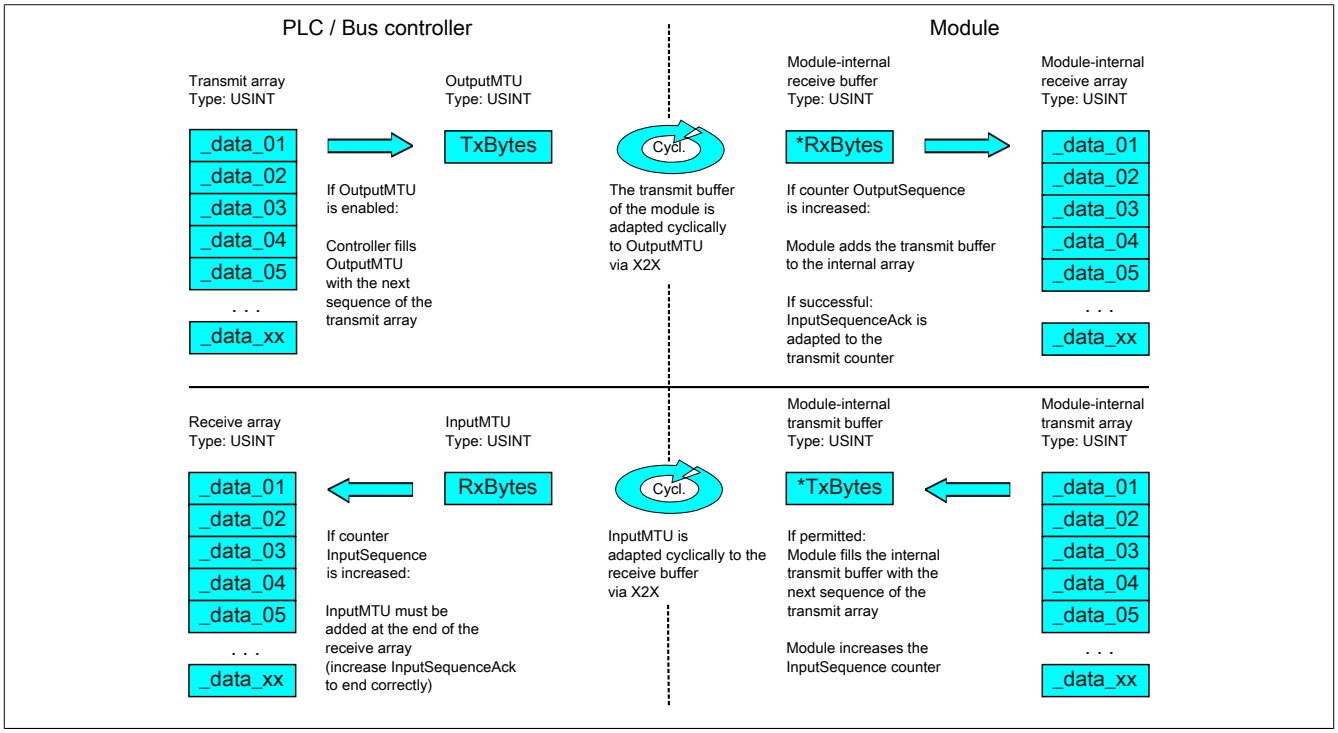


Figure 2: Flatstream communication

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 opposite 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.

4.9.4 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 Rx bytes. For this reason, the user needs to have sufficient knowledge of the communication protocol being used on the field device.

4.9.4.1 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:

Registers are described in section "Flatstream registers" on page 25.

4.9.4.2 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 and ensure that communication is taking place properly, i.e. the transmitter issues the directive that the data should be accepted and the receiver acknowledges that a sequence has been transferred successfully.

Information:

Registers are described in section "Flatstream registers" on page 25.

4.9.4.2.1 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.

4.9.4.2.2 Transport of 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.

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 multi-segment MTUs, 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

4.9.4.2.3 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.

Relationship between OutputSequence and InputSequence

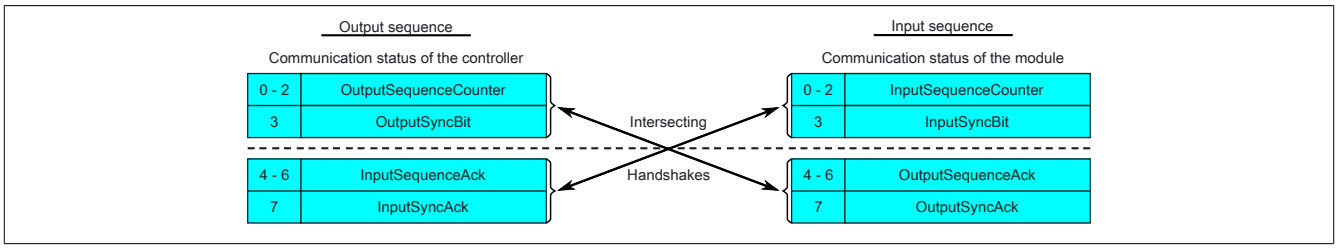


Figure 3: Relationship between OutputSequence and InputSequence

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 opposite 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 opposite 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.

4.9.4.3 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 has to 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 has to 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.

4.9.4.4 Transmitting and receiving

If a channel is synchronized, then the opposite station is ready to receive messages from the transmitter. Before the transmitter can send data, it needs to 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.

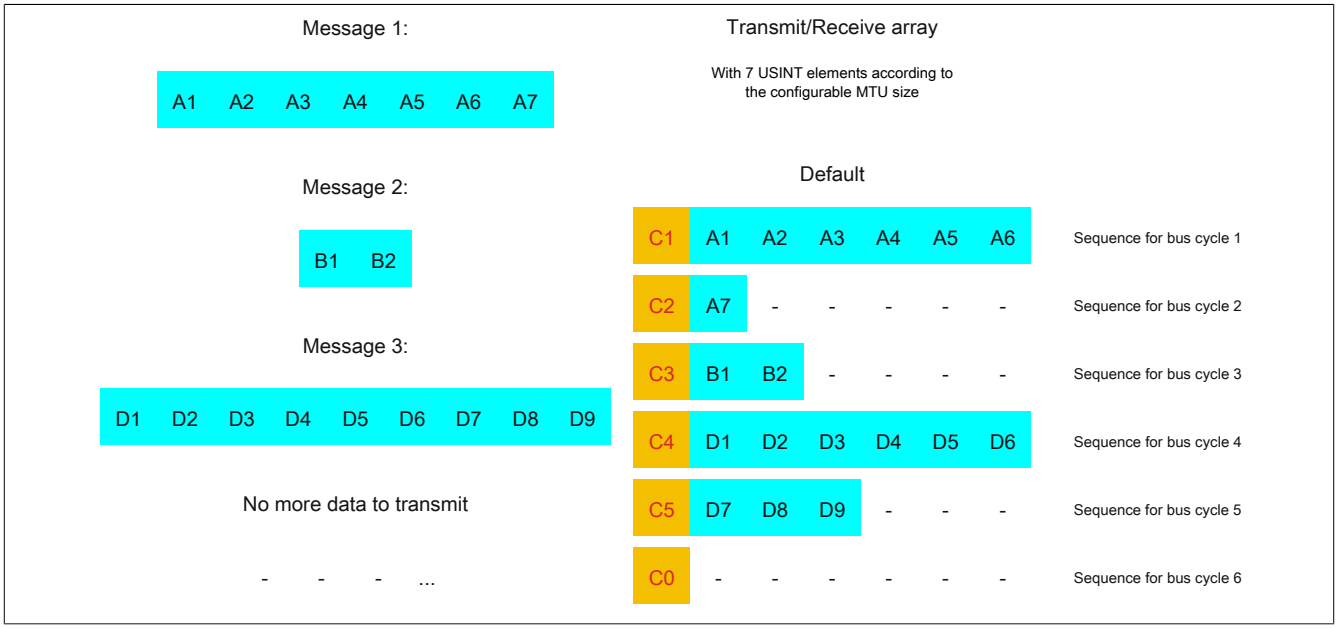


Figure 4: Transmit/Receive array (default)

First, the messages must 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)

4.9.4.4.1 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.

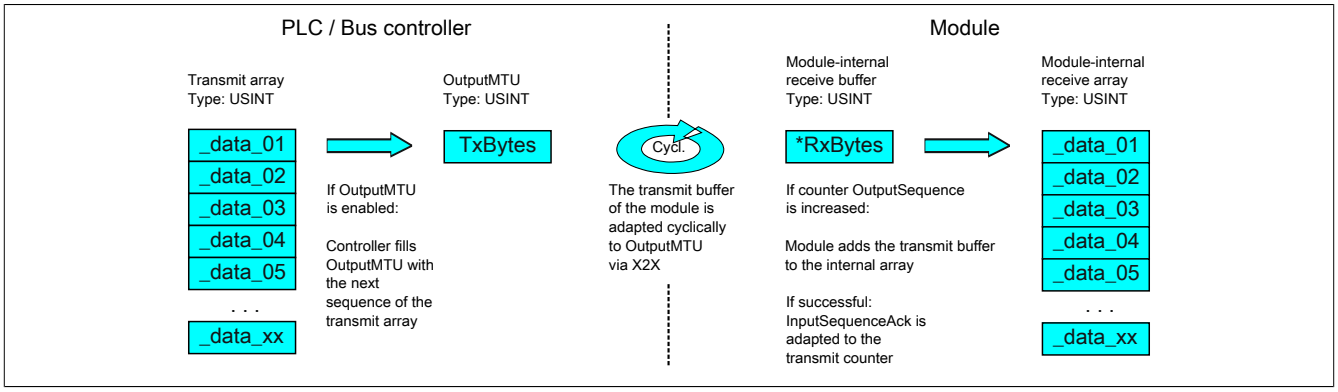


Figure 5: Flatstream communication (output)

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 has to 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

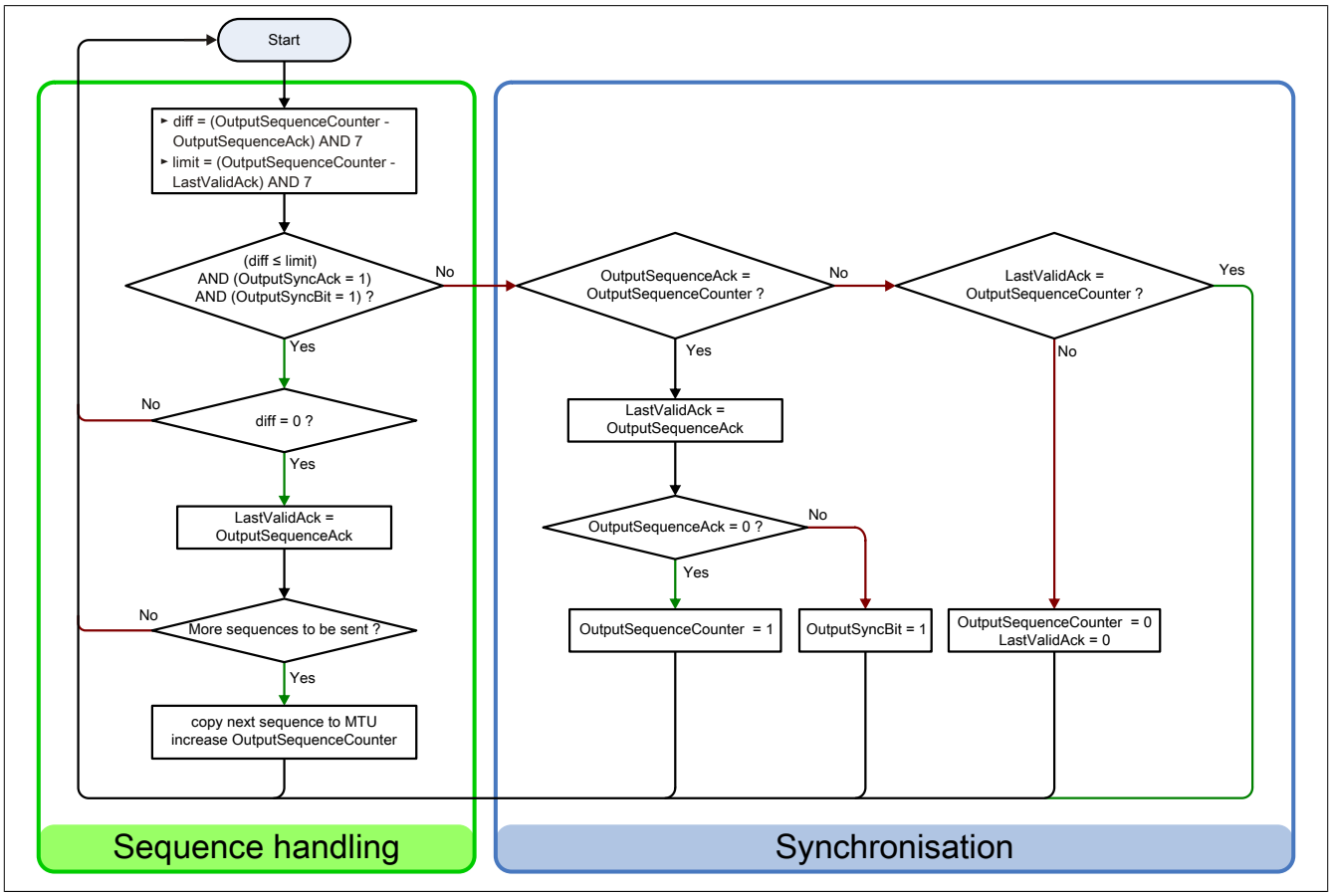


Figure 6: Flowchart for the output direction

4.9.4.4.2 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.

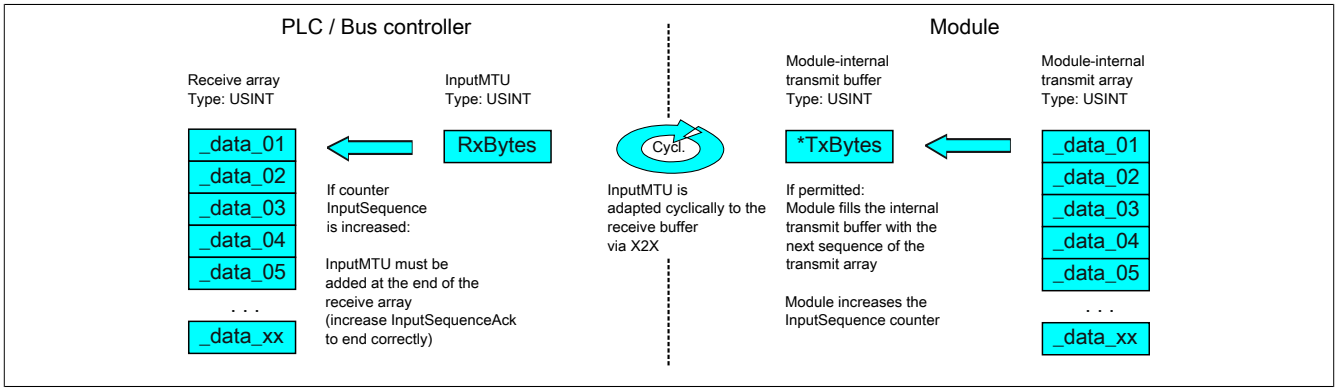


Figure 7: Flatstream communication (input)

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

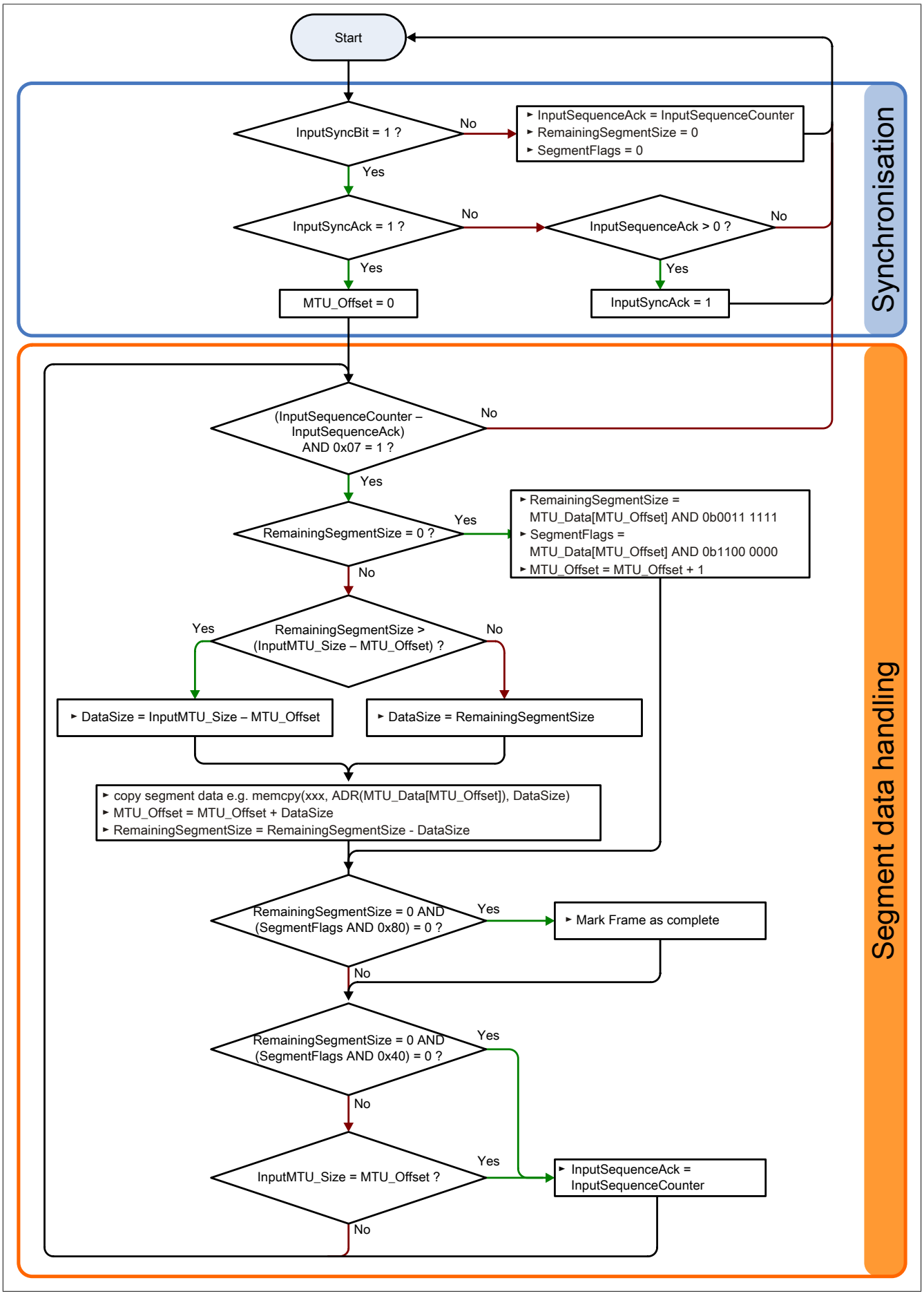


Figure 8: Flowchart for the input direction

4.9.4.4.3 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 opposite 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.

4.9.4.5 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
- MultiSegmentMTUs 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 "MultiSegmentMTUs" 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.

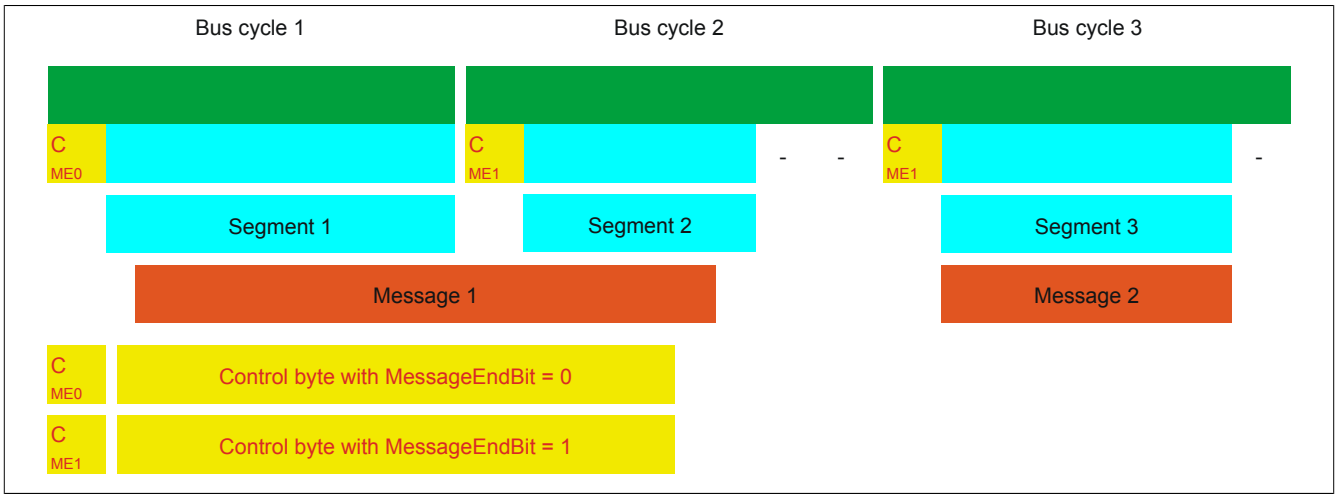


Figure 9: Message arrangement in the MTU (default)

MultiSegmentMTUs 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.

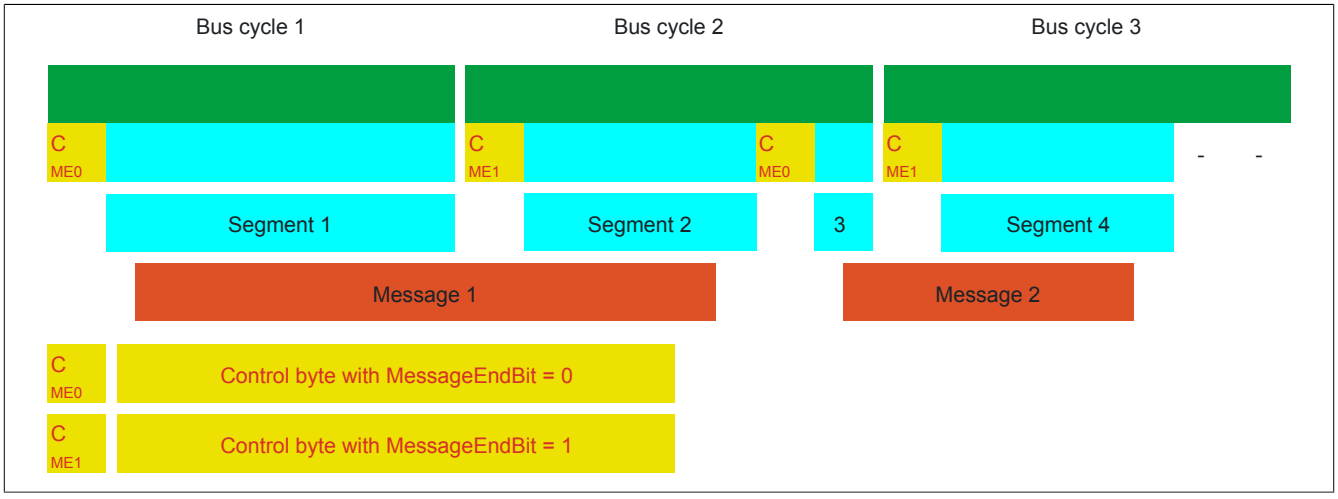


Figure 10: Arrangement of messages in the MTU (MultiSegmentMTUs)

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.

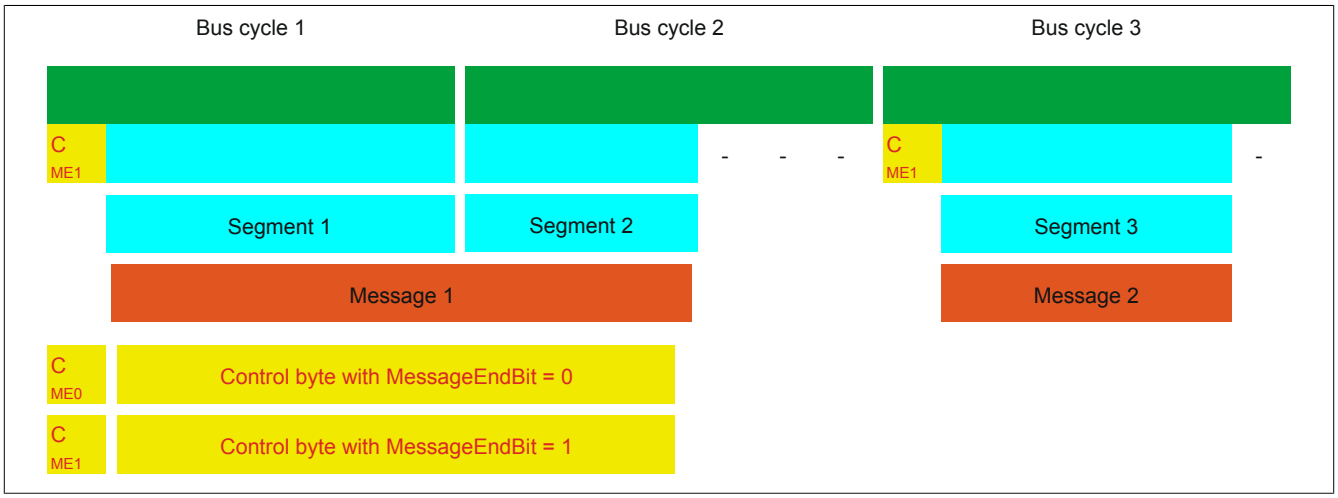


Figure 11: Arrangement of messages in the MTU (large segments)

Using both options

Using both options at the same time is also permitted.

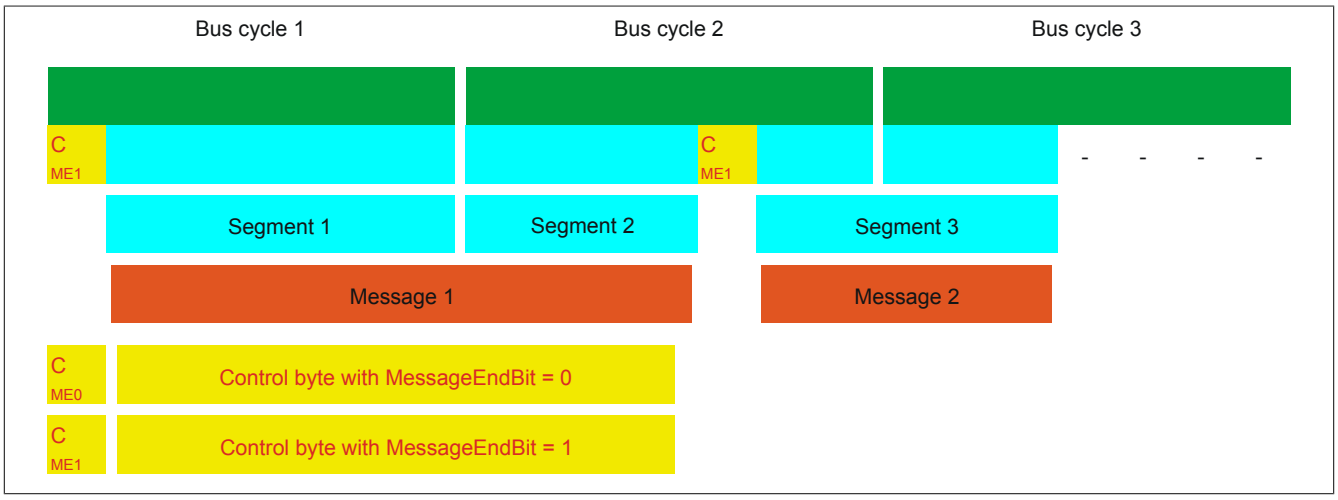


Figure 12: Arrangement of messages in the MTU (large segments and MultiSegmentMTUs)

4.9.4.6 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.

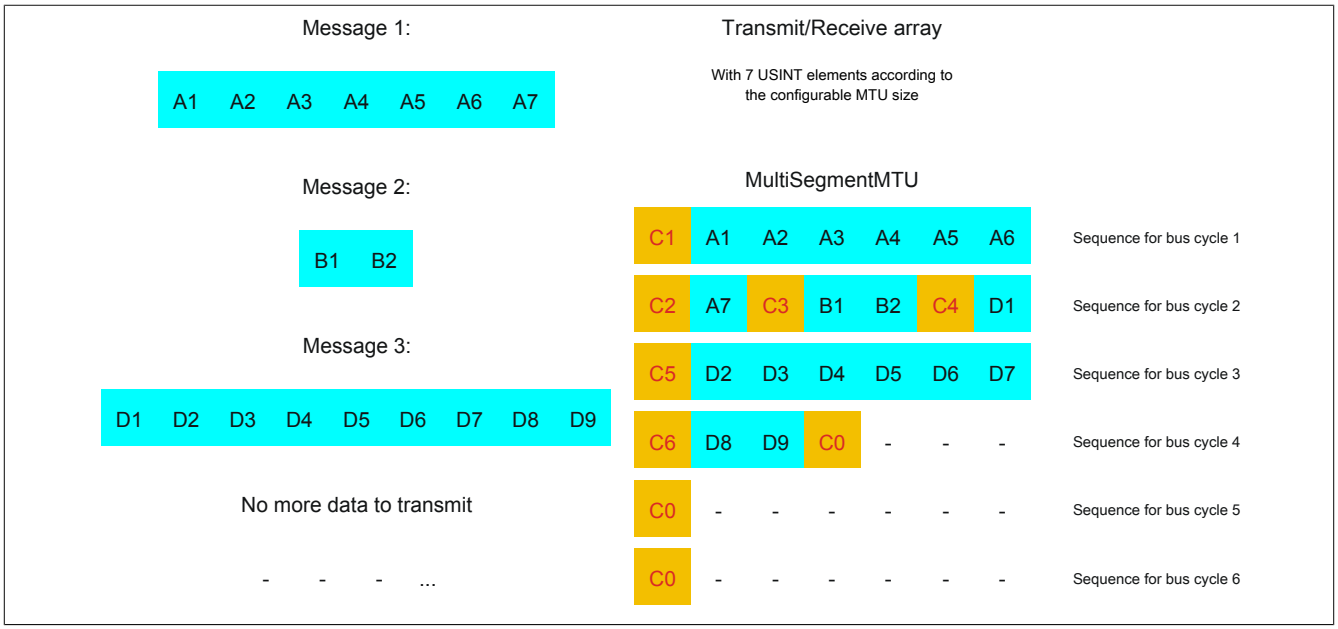


Figure 13: Transmit/receive array (MultiSegmentMTUs)

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

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

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

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

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

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

Warning!

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

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

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

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.

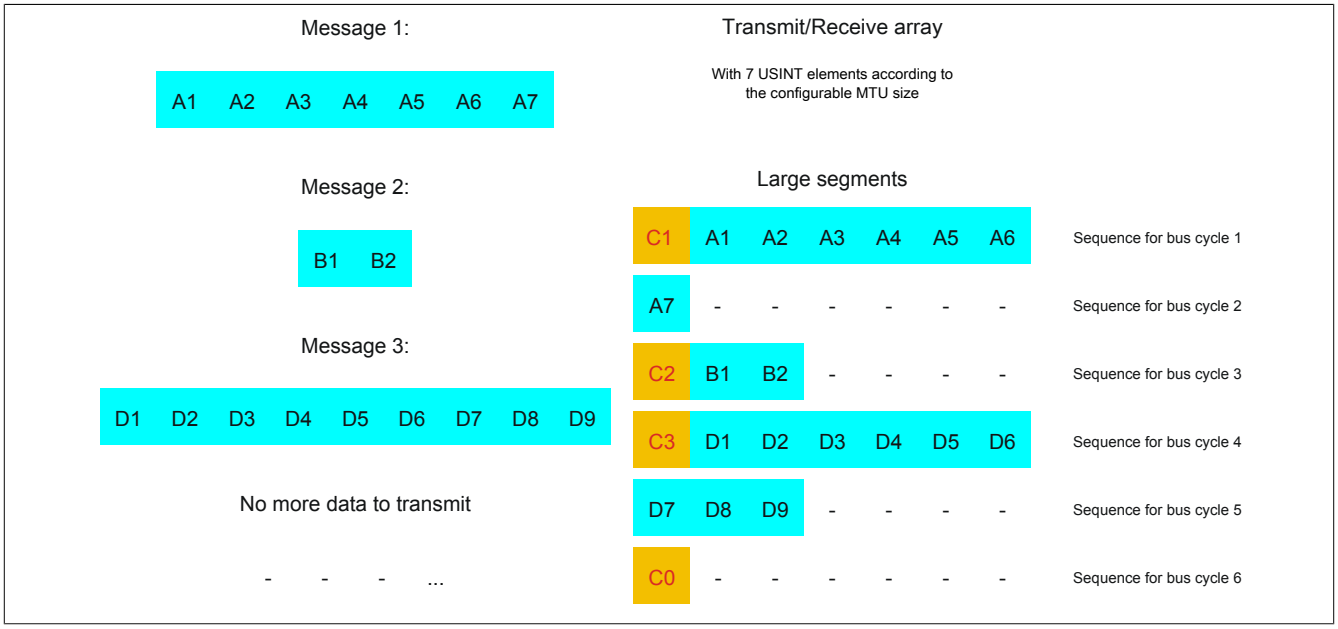


Figure 14: Transmit/receive array (large segments)

First, the messages must 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.

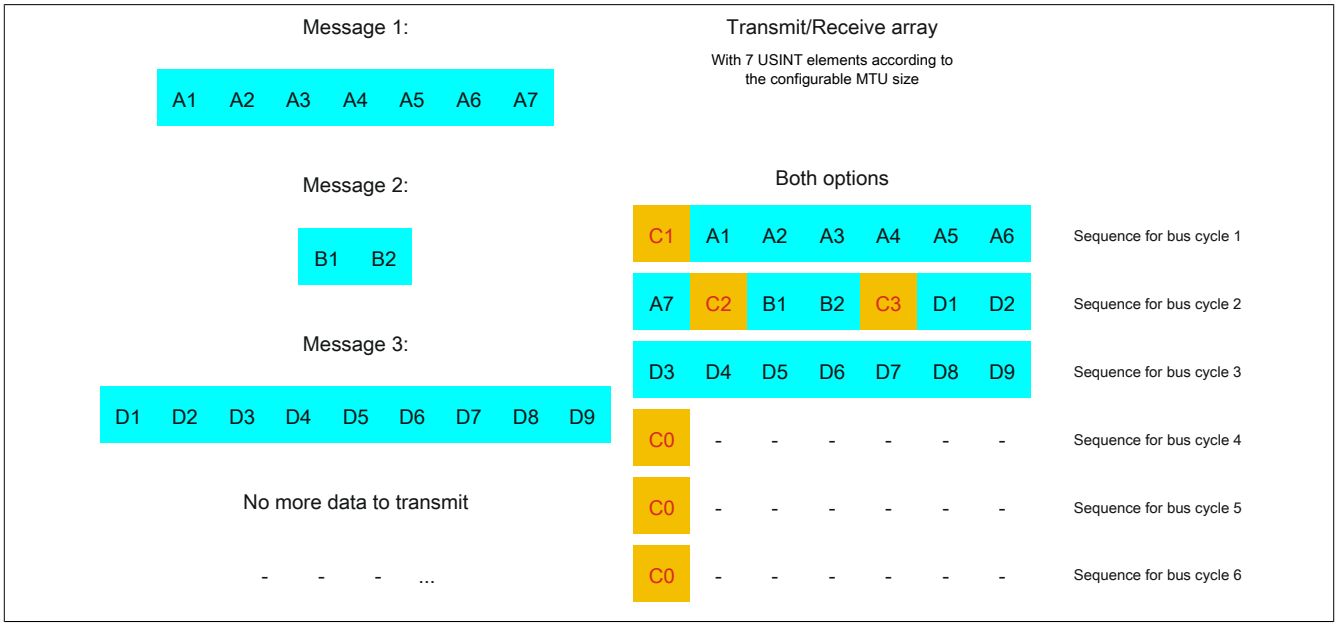


Figure 15: Transmit/receive array (large segments and MultiSegmentMTUs)

First, the messages must 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

4.9.5 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.

4.9.5.1 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 matching 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)	Receiver (task to receive)	Bus system (direction 2)	Transmitter (task for Ack checking)

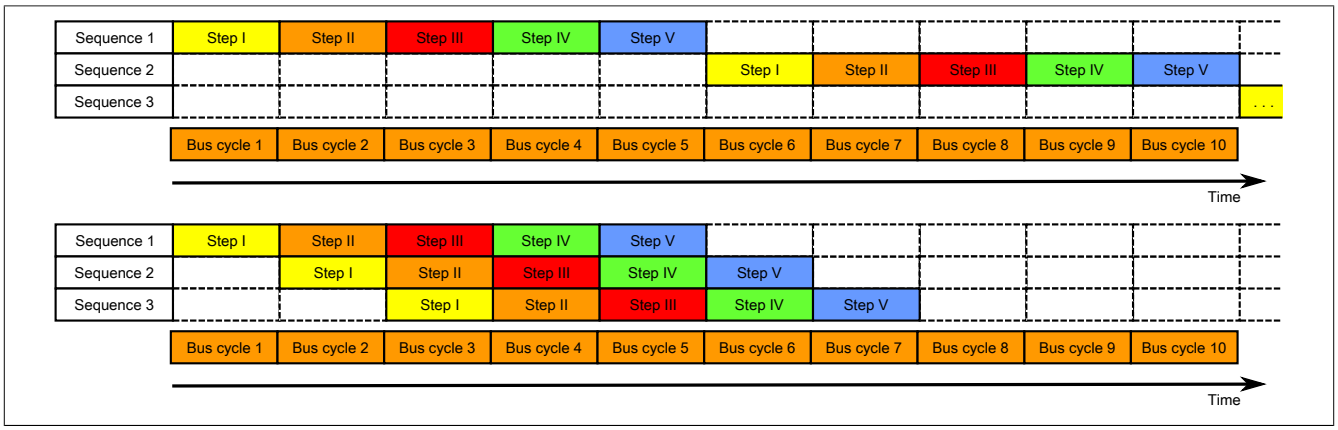


Figure 16: Comparison of transfer without/with Forward

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 still has to acknowledge every sequence received. Only when SequenceAck has been changed and checked by the transmitter is the sequence considered as having been transferred successfully.

4.9.5.2 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:

Registers are described in section "Flatstream registers" on page 25.

4.9.5.2.1 Delay time

The delay time is specified in microseconds. This is the amount of time the module has to 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.

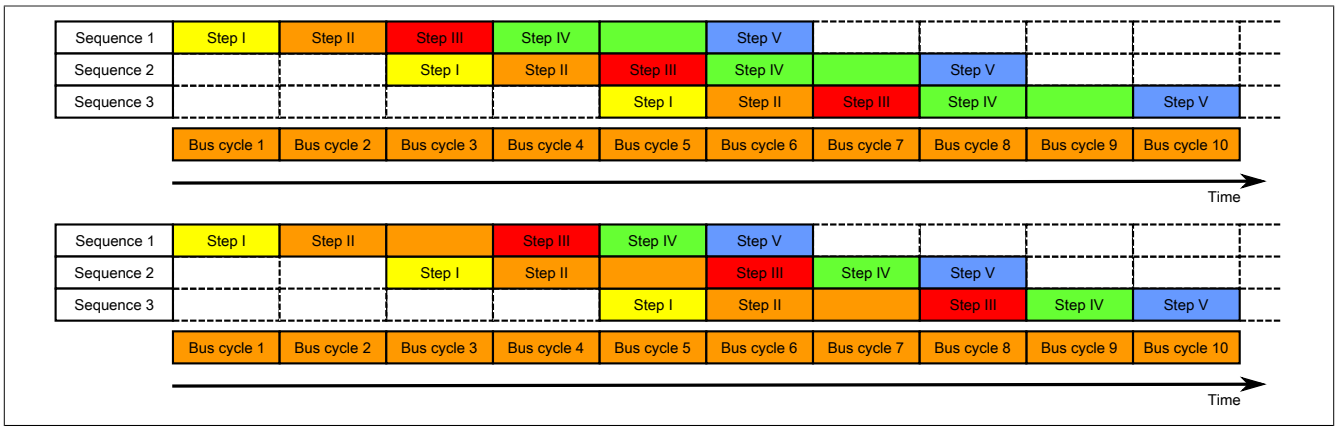


Figure 17: Effect of ForwardDelay when using Flatstream communication with the Forward function

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.

4.9.5.3 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 Sequence-Counter 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.

4.9.5.4 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 X2X Link transfers if this type of interference occurs. 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).

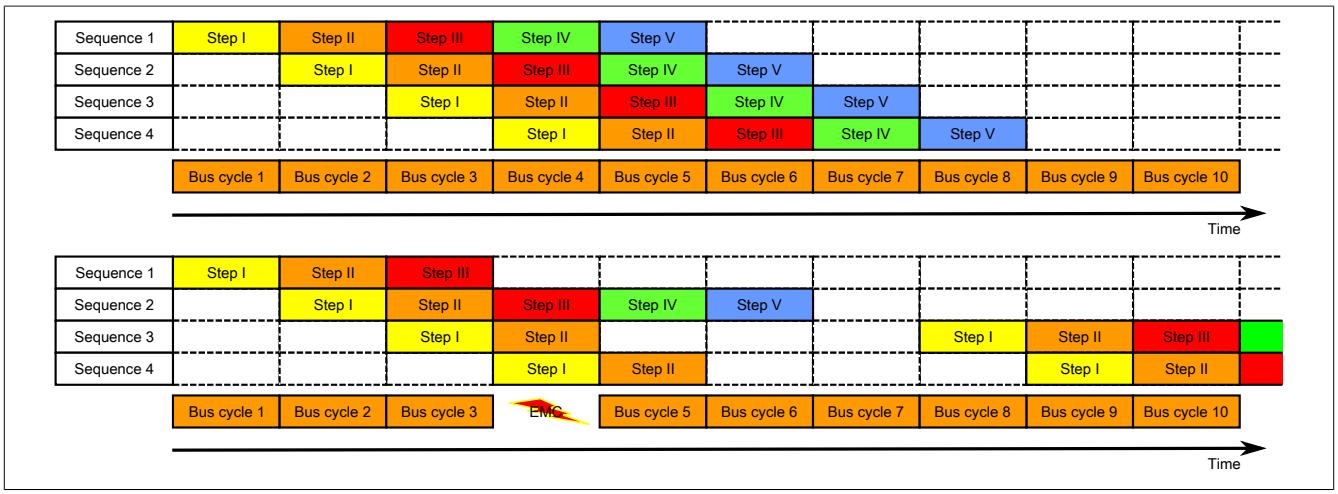


Figure 18: Effect of a lost bus cycle

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.

4.10 NetTime Technology

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

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

4.10.1 Time information

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

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

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

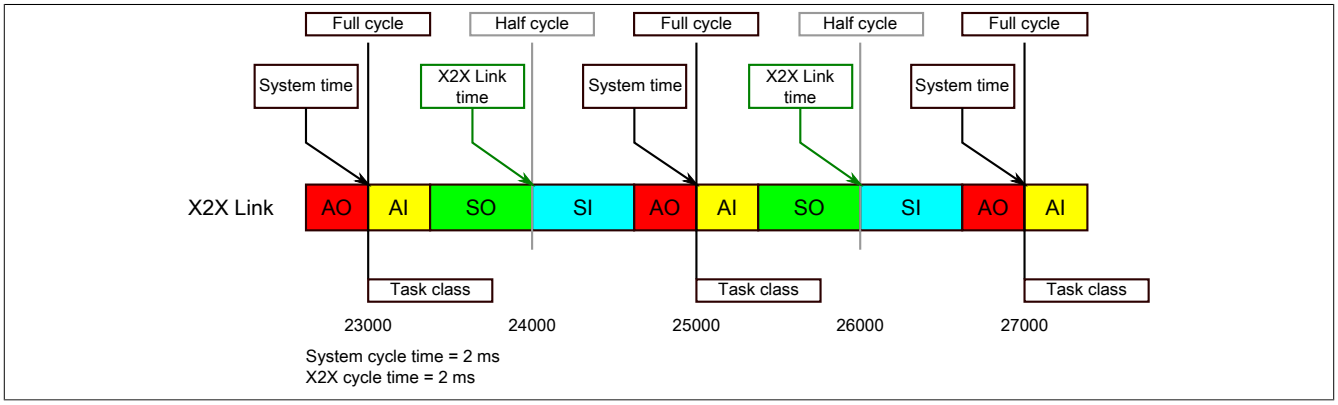
The initialization of the times is based on the system time during the startup of the X2X Link, the I/O modules or the POWERLINK interface.

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

4.10.1.1 Controller data points

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

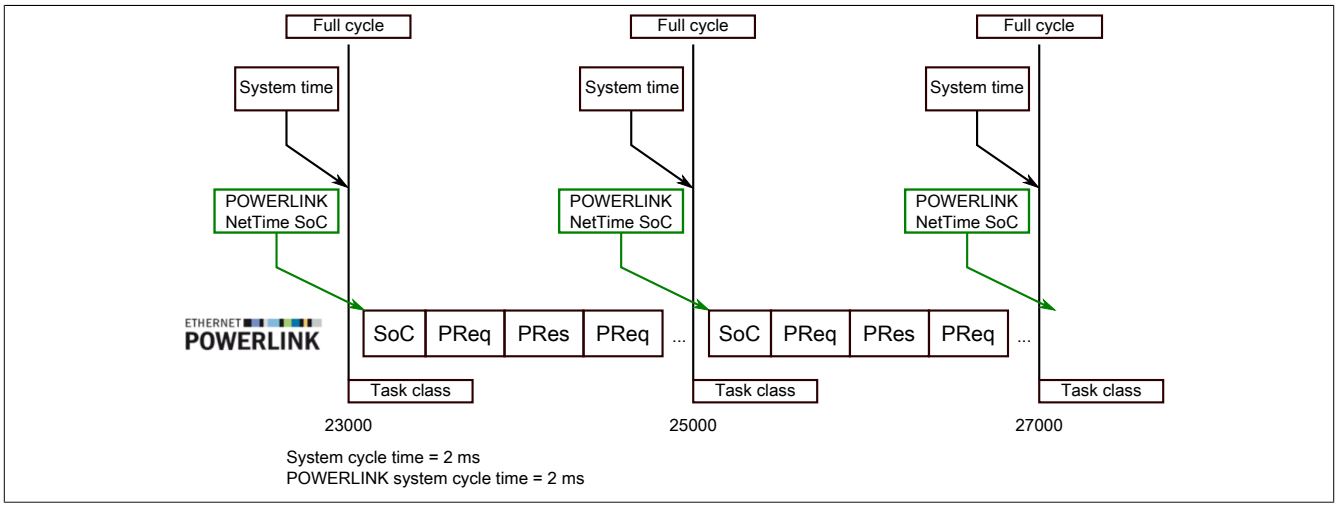
4.10.1.2 X2X Link - Reference time point



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

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

4.10.1.3 POWERLINK - Reference time point

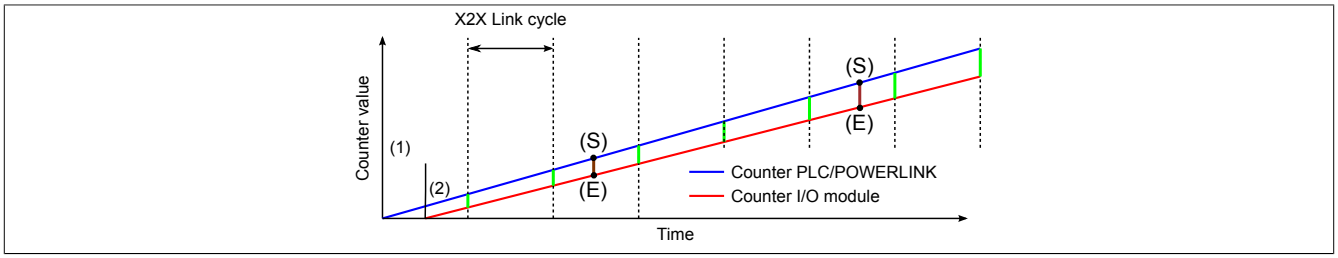


The reference time point on the POWERLINK network is always calculated at the start of cycle (SoC) of the POWERLINK network. The SoC starts 20 µs after the system clock due to the system. This results in the following difference between the system time and the POWERLINK reference time:

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

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

4.10.1.4 Synchronization of system time/POWERLINK time and I/O module



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

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

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

Note

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

4.10.2 Timestamp functions

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

4.10.2.1 Time-based inputs

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

> **Information:**
>
> **The determined moment always lies in the past.**

4.10.2.2 Time-based outputs

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

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

4.10.2.3 Time-based measurements

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

4.11 Minimum cycle time

The minimum cycle time defines how far the bus cycle can be reduced without causing a communication error or impaired functionality. It should be noted that very fast cycles decrease 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 defines how far the bus cycle can be reduced while still allowing an I/O update to take place in each cycle.

Minimum I/O update time	
Analog inputs	1 ms

Minimum I/O update time for HART communication	
Point-to-point	500 ms
Multidrop	500 ms * Number of stations