X20CS1012

1 General information

The M-Bus master is designed as a single-width module and can be connected anywhere within the X20 I/O system. It can therefore be used decentrally for distributed topologies. The M-Bus master supports transfer rates of 300, 2400 and 9600 bit/s; up to 64 slaves supplied via M-Bus can be connected.

M-Bus (Meter-Bus) is a relatively simple fieldbus for recording consumption data, such as from electricity or heat meters. It is based on a reverse polarity protected two-wire line and works according to the master-slave principle.

- Power supply for up to 64 slaves on the M-Bus
- Decentralized use of the communication interface

1.1 Other applicable documents

For additional and supplementary information, see the following documents.

Other applicable documents

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

2 Order data

Order number	Short description	Figure
	X20 electronics module communication	
X20CS1012	X20 interface module, 1 M-Bus master interface, integrated slave supply	
	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	
	Terminal blocks	
X20TB12	X20 terminal block, 12-pin, 24 VDC keyed	

Table 1: X20CS1012 - Order data

3 Technical description

3.1 Technical data

Order number	X20CS1012
Short description	
Communication module	1 M-Bus master for controlling up to 64 slaves
General information	
B&R ID code	0xCABF
Status indicators	Data transfer, M-Bus power supply, operating state, module status
Diagnostics	
Module run/error	Yes, using LED status indicator and software
Data transfer	Yes, using LED status indicator
M-Bus power supply	Yes, using LED status indicator and software
Power consumption	
Bus	0.2 W
Internal I/O	0.35 W + (Number of slaves * 0.08 W)
Module power dissipation	0.55 W + (Number of slaves * 0.006 W)
Additional power dissipation caused by actuators (resistive) [W]	-
Insulation voltage between M-Bus and X2X Link	500 VDC, 1 min
Certifications	
CE	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
EAC	Yes
Interfaces	
Interface	
Type	M-Bus master
Variant	Connection via 12-pin terminal block X20TB12
Max. distance	See section "M-Bus".
Transfer rate	300, 2400 or 9600 bit/s
Number of slaves	Max. 64
Internal resistance of master	Max. 6 Ω
Bus voltage mark at 0 mA	I/O supply voltage + (11.5 to 13.5) V
Bus voltage drop with space	12 to 13.5 V
Overload shutdown	250 mA ±10%
Bit threshold	6 to 9 mA
Collision threshold	24 to 36 mA
Received readjustment time	Max. 10 s [1)]
Bus cable	Shielded or unshielded
Electrical properties	
Electrical isolation	M-Bus isolated from bus M-Bus not isolated from I/O power supply
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	-
Storage	-40 to 85°C
Transport	-40 to 85°C

Table 2: X20CS1012 - Technical data

Order number	X20CS1012
Relative humidity	
Operation	5 to 95%, non-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.
Pitch	12.5$^{+0.2}$ mm

Table 2: X20CS1012 - Technical data

1) After each load change on M-Bus (e.g. switching slaves on or off)

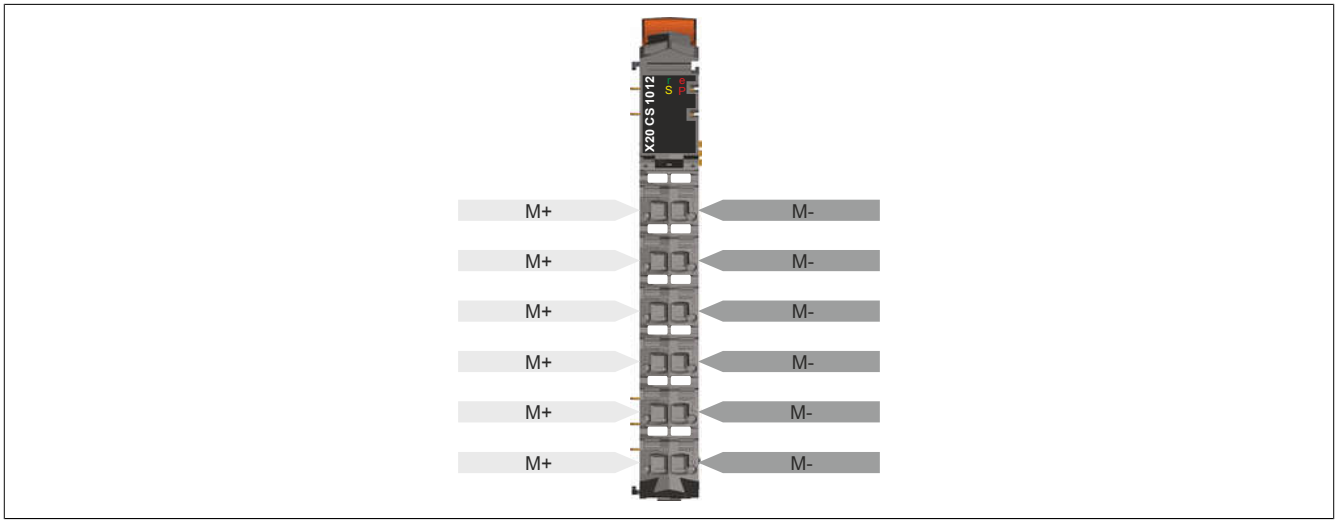
3.2 LED status indicators

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

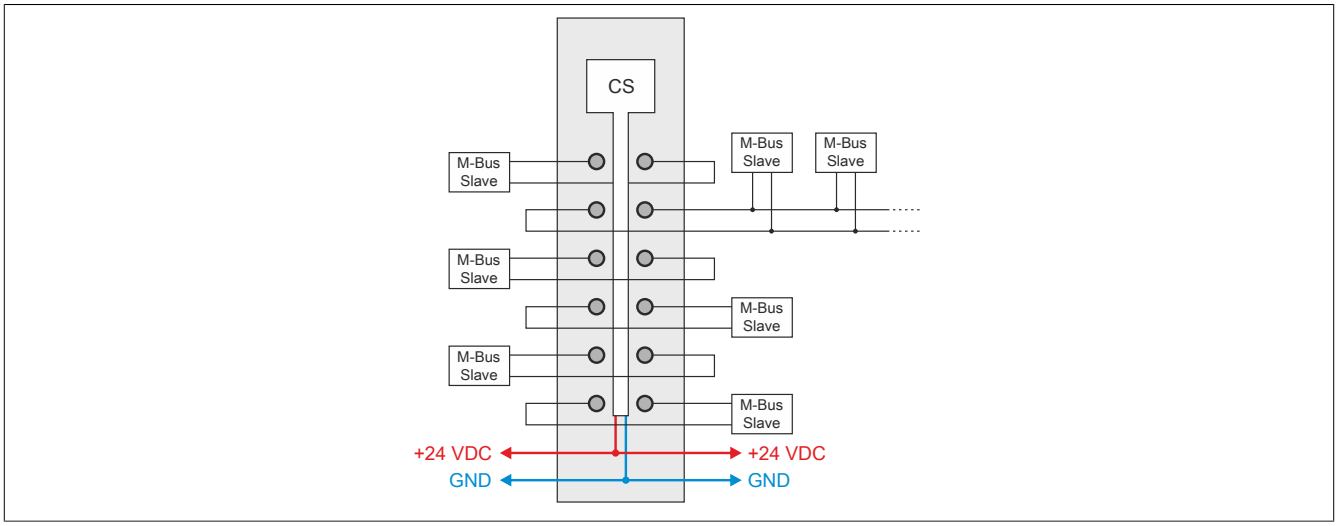
Figure	LED	Color	Status	Description
	r	Green	Off	No power to module
			Single flash	UNLINK mode
			Double flash	Mode BOOT (during firmware update)[1]
			Blinking	PREOPERATIONAL mode
			On	RUN mode
	e	Red	Off	No power to module or everything OK
			On	Error or reset status
	e + r	Red on / Green single flash		Invalid firmware
	S	Yellow	Off	No slaves sending data
			On	At least one slave is sending data via the M-Bus
	P	Red	Off	M-Bus supply ok
			On	Short-circuit or overload on M-Bus

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

3.3 Pinout



3.4 Connection example



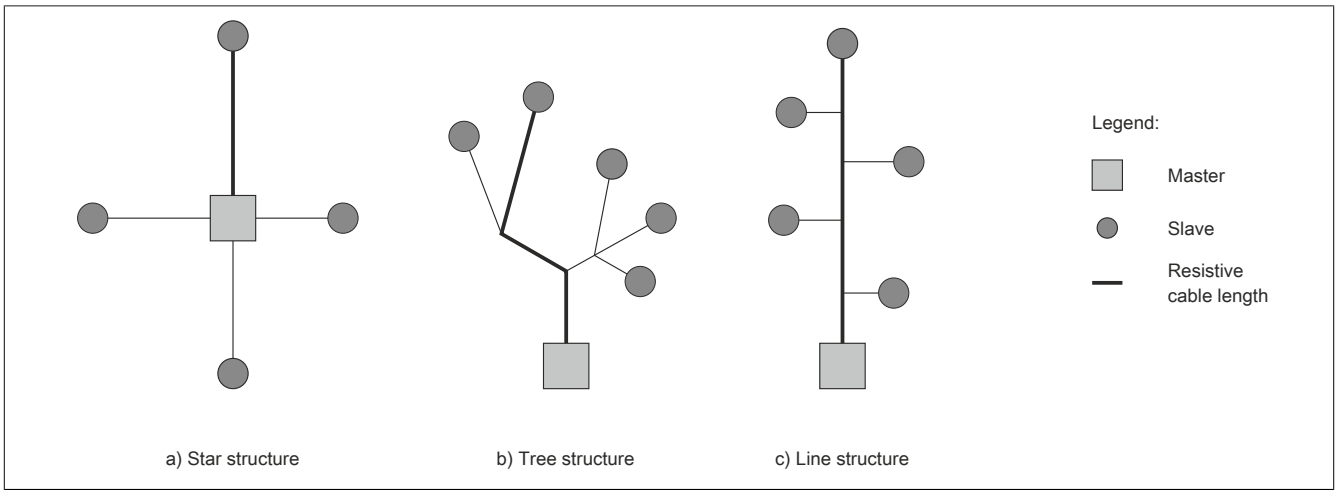
3.5 Usage after the X20IF1091-1

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 M-Bus

4.1 Bus topology

The bus topology has a significant influence on the maximum load of an M-Bus network. In general a star structure is preferred over a tree structure and in turn a tree structure is preferred over a line structure. Furthermore, connecting the slaves to the bus the same way provides better values than connecting them all at the end of the branches after all other parameters have been determined.



4.2 Cable cross section

The cable being used has a specific capacity and resistance, which in turn has an effect on the operation of the bus. The resistive influence of the cable means a loss of voltage on the line, which is subsequently not available for supplying the bus. In order to guarantee sufficient power, the voltage on the slaves must never be less than 12 V neither when sending from the master to the slave, nor in the opposite direction. The deciding factor in this case is the longest branch of the network whose length is referred to as the resistive cable length.

The cable's capacity causes signal distortion during data transfer because the slew rates of the rising and falling edges are slowed down. For example, replacing a 3 km branch in a network with two 1.5 km branches will improve the signal. The total distance of the network is referred to as the capacitive cable length (the sum of all segment lengths).

Information:

The maximum permissible line resistance (for the longest loop) is 250 Ω.

The maximum permissible line capacitance for the entire bus is 500 nF.

4.3 Transmission current and bit threshold

The bit threshold on the master is typically 7.5 mA. Therefore, a slave transmission current of 15 mA results in the least amount of signal distortion while the highest amount occurs at 11 or 20 mA.

4.4 Transfer rate

A lower transfer rate decreases the influence of the signal distortion caused by cable capacity and bit threshold.

Information:

Starting with a total bus length >1 km, the slaves must be operated at a baud rate <9600 bit/s.

4.5 Calculating the resistive bus length

The resistive cable length must be calculated in order to ensure a sufficient power supply of 12 V on the M-Bus. What matters most here is the longest segment between the master and slave.

The resistive bus length is calculated using the following formula without taking an increased bus current caused by a defective receiver into account:

$$L_{res} = \frac{V_{I/O} - (n * 0.0015 + 0.02) * 6 - 12.6}{(n * 0.0015 + 0.02) * R_L} * 1000$$

L_{res} ... Resistive bus length [m]
n ... Number of slaves (all at end of line)
R_L ... Line resistance (loop resistance [Ω/km])
$V_{I/O}$ ... I/O supply voltage [V]

Examples for calculating the maximum resistive bus length:

No.	Example	Maximum resistive bus length
1	• 64 slaves (all at end of line) • 19.2 V I/O supply voltage • 0.5 mm² wire cross-section	675 m
2	• 64 slaves (all at end of line) • 28.8 V I/O supply voltage • 1.5 mm² wire cross-section	5340 m

4.6 Accounting for the capacitive bus length

The total distance of the network is referred to as the capacitive bus length (the sum of all segment lengths). The capacitive bus length depends on two factors:

- Distributed capacitance of cable
- Transfer rate

Distributed capacitance of cable

A lower distributed capacitance on a cable means a higher capacitive bus length.

Transfer rate

A lower transfer rate on an M-Bus system means a higher capacitive bus length.

Example of a cable with a distributed capacitance of 50 nF/km:

Transfer rate	Capacitive bus length
9600 bit/s	1 km
2400 bit/s	4 km
300 bit/s	10 km

4.7 Bus installation

Cables with twisted pair wires and a cross-section of 0.5 mm² to 1.5 mm² are normally used for bus installation (according to standard: J-Y(ST)Y nx2x0.8). The shield on shielded cables only has to be grounded to the module on one side. On the slaves, the shielding must be high resistance for DC and low frequency signals.

4.8 Repeater

Repeaters can be used to further expand the M-Bus network.

5 Register description

5.1 General data points

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

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

5.2 Function model 0 - Standard

Register	Name	Data type	Read		Write	
			Cyclic	Non-cyclic	Cyclic	Non-cyclic
Module configuration						
774	CfO_FunctionModel	UINT				•
M-Bus - Configuration						
Index * 16 + 767	CfO_LengthData1 to CfO_LengthData8	USINT				•
Index * 16 + 775	CfO_BaudData1 to CfO_BaudData8	USINT				•
Index * 16 + 761	CfO_PAdrData1 to CfO_PAdrData8	USINT				•
Index * 16 + 765	CfO_IndexData1 to CfO_IndexData8	USINT				•
Index * 16 + 773	CfO_ReqTimeData1 to CfO_ReqTimeData8	USINT				•
Index * 16 + 770	CfO_MBusModeData1 to CfO_MBusModeData8	UINT				•
Index * 16 + 763	CfO_ToutOffData1 to CfO_ToutOffData8	USINT				•
Index * 8 + 1009	CfO_ReplData1 to CfO_ReplData8	(U)SINT				•
Index * 8 + 1010	CfO_ReplData1 to CfO_ReplData8	(U)INT				•
Index * 8 + 1012	CfO_ReplData1 to CfO_ReplData8	(U)DINT REAL				•
M-Bus - Communication						
513	MBusCommand	USINT			•	•
263	MBusOperation	USINT	•			
257	MBusState	USINT	•			
259	ValidDataByte	USINT	•			
	ValidData1	Bit 0	•			
	...	...	•			
	ValidData8	Bit 7	•			
261	InvalidDataByte	USINT	•			
	InvalidData1	Bit 0	•			
	...	...	•			
	InvalidData8	Bit 7	•			
Index * 8 + 265	Data1 to Data8	(U)SINT	•			
Index * 8 + 266	Data1 to Data8	(U)INT	•			
Index * 8 + 268	Data1 to Data8	(U)DINT REAL	•			
337	ChangedSNByte	USINT	•			
Index * 8 + 900	SNData1 to SNData8	UDINT		•		
FlatStream						
2051	InputMTU	USINT				•
2049	OutputMTU	USINT				•
2113	InputSequence	USINT	•			
Index * 2 + 2113	RxByte1 to RxByte15	USINT	•			
2177	OutputSequence	USINT			•	
Index * 2 + 2177	TxByte1 to TxByte15	USINT			•	
2053	FlatstreamMode	USINT				•
2055	Forward	USINT				•
2057	ForwardDelay	UINT				•

5.3 Function model 254 - Bus controller

Register	Offset [1]	Name	Data type	Read		Write	
				Cyclic	Non-cyclic	Cyclic	Non-cyclic
Module configuration							
774	-	CfO_FunctionModel	UINT				•
M-Bus - Configuration							
Index * 16 + 767	-	CfO_LengthData1 to CfO_LengthData8	USINT				•
Index * 16 + 775	-	CfO_BaudData1 to CfO_BaudData8	USINT				•
Index * 16 + 761	-	CfO_PAdrData1 to CfO_PAdrData8	USINT				•
Index * 16 + 765	-	CfO_IndexData1 to CfO_IndexData8	USINT				•
Index * 16 + 773	-	CfO_ReqTimeData1 to CfO_ReqTimeData8	USINT				•
Index * 16 + 770	-	CfO_MBusModeData1 to CfO_MBusMode-Data8	UINT				•
Index * 16 + 763	-	CfO_ToutOffData1 to CfO_ToutOffData8	USINT				•
Index * 8 + 1009	-	CfO_ReplData1 to CfO_ReplData8	(U)SINT				•
Index * 8 + 1010	-	CfO_ReplData1 to CfO_ReplData8	(U)INT				•
Index * 8 + 1012	-	CfO_ReplData1 to CfO_ReplData8	(U)DINT REAL				•
M-Bus - Communication							
8	8	MBusCommand	USINT			•	•
11	11	MBusOperation	USINT	•			
8	8	MBusState	USINT	•			
9	9	ValidDataByte	USINT	•			
10	10	InvalidDataByte	USINT	•			
Index * 4 + 5	Index * 4 + 8	Data1 to Data8	(U)SINT	•			
Index * 4 + 6	Index * 4 + 8	Data1 to Data8	(U)INT	•			
Index * 4 + 8	Index * 4 + 8	Data1 to Data8	(U)DINT REAL	•			
337	-	ChangedSNByte	USINT		•		
Index * 8 + 900	-	SNData1 to SNData8	UDINT		•		
FlatStream							
2051	-	InputMTU	USINT				•
2049	-	OutputMTU	USINT				•
0	0	InputSequence	USINT	•			
Index * 1 + 0	Index * 1 + 0	RxByte1 to RxByte7	USINT	•			
0	0	OutputSequence	USINT			•	
Index * 1 + 0	Index * 1 + 0	TxByte1 to TxByte7	USINT			•	
2053	-	FlatstreamMode	USINT				•
2055	-	Forward	USINT				•
2057	-	ForwardDelay	UINT				•

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

5.3.1 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).

5.3.2 CAN I/O bus controller

The module occupies 3 analog logical slots with CAN I/O.

5.4 General information

The M-Bus standard is a serial bus system that handles half-duplex or asynchronous communication. The high level of variability provided by this protocol enables a wide range of information to be handled via the same interface. In basic M-Bus networks, the master communicates with up to 250 slaves via the "primary address". In later stages of development, the secondary address (4 bytes) was then also specified. This made it possible to significantly increase the number of slaves in a network.

Important information about the module

- Generally: Primary address used (1 to 250)
- Work with the secondary address is supported only by Flatstream.
- Bus can supply 64 slaves with power

5.5 Module configuration

The flexible design of the M-Bus protocol can quickly add up to a lot of configuration work. That's why B&R offers two different user interfaces for the module: "Standard" and "FlatStream". The user-friendly B&R Standard interface allows users to view up to eight values requested cyclically from the M-Bus network. In FlatStream mode, the module acts as a bridge between the PLC and the M-Bus slave, which makes all M-Bus functions available.

Information:

The B&R Standard interface is statically configured and based on cyclic registers. Because X2X Link can only transfer a certain number of values cyclically, the user must make his selection accordingly.

5.5.1 Settings for operation

Name:
CfO_FunctionModel

This register can be used to enable either the Standard or FlatStream interface, which makes the module much more efficient.

Bits 8 to 15 are only evaluated if bit 0 (B&R default interface) is enabled.

Data type	Values	Bus controller default setting
USINT	See the bit structure.	1825

Bit structure:

Bit	Description	Value	Information
0	B&R standard interface	0	Disabled
		1	Enabled (bus controller default setting)
1	Flatstream	0	Disabled (bus controller default setting)
		1	Enabled
3 - 7	Reserved	0	
8	Data1	0	Disabled
		1	Enabled (bus controller default setting)
...		...	
10	Data 3	0	Disabled
		1	Enabled (bus controller default setting)
11	Data 4	0	Disabled (bus controller default setting)
		1	Enabled
...		...	
15	Data8	0	Disabled (bus controller default setting)
		1	Enabled

5.6 M-Bus - Configuration

Separate configuration registers are provided for each value to read. These must be configured correctly in order to call up a counter value from the M-Bus network. The user must know the following values from the slave:

- Transfer rate configured on the slave
- Primary address configured on the slave (value: 1 to 250, otherwise only point-to-point connection is possible)
- Data type / data length of value
- How the slave's memory is structured

Information:

The following section "M-Bus - Configuration" is based solely on the B&R standard interface.

5.6.1 Data length

Name:
CfO_LengthData1 to CfO_LengthData8

The Standard interface is able to request data from the M-Bus slave with different lengths. When using Automation Studio the value of the "Length" register is a result of the data type defined for the X2X Link. All common data types with up to 4 bytes in length are supported.

Data type	Value	Bus controller default setting
USINT	See the bit structure.	8

Bit structure:

Bit	Name	Value	Information
0 - 5	Data length code	00 0000	USINT
		00 0001	SINT
		00 0010	UINT
		00 0100	INT
		00 1000	UDINT (bus controller default setting)
		01 0000	DINT
		10 0000	REAL
6 - 7	Reserved	0	

5.6.2 Transfer rate

Name:
CfO_BaudData1 to CfO_BaudData8

This register can be used to define the transfer rate for retrieving the desired values.

Data type	Value	Bus controller default setting
USINT	See the bit structure.	4

Bit structure:

Bit	Name	Value	Information
0 - 3	Baud rate (Code)	0000	Reserved!
		0001	300 bit/s
		0010	600 bit/s
		0011	1200 bit/s
		0100	2400 bits/s (bus controller default setting)
		0101	4800 bit/s
		0110	9600 bit/s
		0111	19200 bit/s
		1000	38400 bit/s
4 - 7	Reserved	0	

5.6.3 Address

Name:
CfO_PAdrData1 to CfO_PAdrData8

This register can be used to define the address where the desired values will be requested from.

Data type	Value	Information
USINT	1 to 250	Bus controller default setting: 254

Special addresses:

Value	Information
251 to 253	Reserved (in accordance with M-Bus specification)
254	Broadcast address (response from all connected slaves - risk of collision)

5.6.4 Index

Name:
CfO_IndexData1 to CfO_IndexData8

This register is used to specify the ordinal number of the value (independent of the medium). This value results from the order of values in the slave. The value is then transferred to the data register.

Data type	Value	Information
USINT	1 to 255	Bus controller default setting: 1

5.6.5 Specifies the refresh time

Name:
CfO_ReqTimeData1 to CfO_ReqTimeData8

The slave information can be queried manually or time-based. A time-controlled query must include the value for the refresh time. The associated unit is defined in register "M-Bus mode" on page 11.

Data type	Value	Information
USINT	1 to 255	In [s, min]. Bus controller default setting: 0

5.6.6 M-Bus mode

Name:
CfO_MBusModeData1 to CfO_MBusModeData8

To speed up the module's boot procedure, various configuration details that define the module's behavior have been combined in this register.

Data type	Values	Bus controller default setting
UINT	See the bit structure.	2

Bit structure:

Bit	Description	Value	Information
0 - 2	Byte offset	0 - 7	Data types. Bus controller default setting: 2
3 - 4	Reserved	0	
5	InitFrame	0	No additional frame (bus controller default setting)
		1	Transmit additional frame
6	ApplicationResetFrame	0	No additional frame (bus controller default setting)
		1	Transmit additional frame
7	Replacement value strategy	0	Hold last valid value (bus controller default setting)
		1	Replace with static value
8	Time-based reading	0	Disabled (bus controller default setting)
		1	Enabled
9	Manually triggered reading	0	Disabled (bus controller default setting)
		1	Reading via register "MBusCommand" on page 13
10	Unit of periodic reading	0	[s] - Seconds (bus controller default setting)
		1	[min] - Minutes
11 - 15	Reserved	0	

Byte offset

The M-Bus specification allows for many individual data types. In order to also read these counter values with up to 64 bits, a slave value may have to be read using 2 data registers. The byte offset can be defined to select a desired section of the information.

5.6.7 Timeout offset

Name:
CfO_ToutOffData1 to CfO_ToutOffData8

The timeout for the M-Bus communication generally depends on the currently defined transfer rate. The user can also define an offset value in addition to the calculated standard timeout.

Timeout = standard timeout + (timeout offset * 10 ms)

Data type	Value	Information
USINT	0 to 255	Resolution 10 ms. Bus controller default setting: 0

5.6.8 Static replacement value

Name:
CfO_ReplData1 to CfO_ReplData8

This register defines the static replacement value if replacement value strategy "Replace with static value" was enabled in register "CfO_MBusModeData" on page 11. The data register takes on this value if an invalid input value is detected.

Data type	Value	Information
(U)SINT (U)INT (U)DINT REAL	According to data type	Bus controller default setting: 0

5.7 M-Bus - Communication

Three important control and status bytes are provided in the B&R interface for communication with the M-Bus slaves. Register "MBusCommand" on page 13 switches UART on/off, for example, in order to increase the system's energy efficiency.

Up to 8 cyclic input registers are registered depending on the configuration. Manually configured data must be requested via register "MBusCommand" on page 13. Registers "ValidDataByte" on page 14 and "Invalid-DataByte" on page 14 can be used to determine the quality of the value currently read.

Information:

The following section "M-Bus - Communication" is based solely on the B&R standard interface.

5.7.1 M-Bus commands

Name:
MBusCommand

This register can be used to apply different commands to the module. The module only responds to positive edges.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	Activate UART	0 → 1	Execute command
1	Read manually triggered values	0 → 1	Execute command
2	Acknowledge the "MBusState" register	0 → 1	Execute command
3 - 6	Reserved	0	
7	Deactivate UART	0 → 1	Execute command

Bit 0 and 7

The level converter is switched on by default when the module boots. This bit can be used to switch it on or off from the application to save electricity, for example.

5.7.2 M-Bus operation

Name:
MBusOperation

This register shows the user which task the module is currently processing. The LSB is always set when the UART is active. Manual commands are indicated by an increase of one in this byte while being processed.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	UART	0	Inactive
		1	Active
1	Read values	0	-
		1	Command being processed
2	Refresh/reset the "MBusState" register	0	-
		1	Command being processed[1)]
3 - 6	Reserved	0	
7	UART	0	Inactive
		1	Active

1) Bit 2 is only set for one X2X cycle. Requesting this bit is not recommended when operating the module behind a bus controller.

5.7.3 M-Bus state

Name:
MBusState

This register contains the current M-Bus network error state. All bits are managed in nonvolatile memory. This means they must be reset via register "MBusCommand" on page 13.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0	Collision detection	0	Error-free addressing
		1	Multiple addresses on the bus
1	Read error (at least once)	0	Configured information OK
		1	Could not read information
2	Checksum	0	Received checksum OK
		1	Error in input direction
3	M-Bus load	0	Power supply OK
		1	Load too high on M-Bus network
4	Communication aborted due to overflow	0	Everything OK
		1	The master is overloaded and cannot take on any additional requests. Corrective measure: Repeat the request.[1)]
5	Communication aborted due to level converter	0	Everything OK
		1	Level converter is OFF (aborted at runtime or not started).
6	Data exchange since startup	0	Valid data not yet received
		1	Valid data at least once
7	UART off MBUS ENABLE	0	M-Bus driver, level converter inactive
		1	Module ready for communication

1) Communication is reestablished automatically as soon as the pending communication jobs have been processed.

5.7.4 Valid data

Name:
ValidDataByte

ValidData1 to ValidData8

This register indicates (by bit) which of the max. 8 read values are valid.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	ValidData1	0	Value 1 invalid
		1	Value 1 valid
...	...	...	...
7	ValidData8	0	Value 8 invalid
		1	Value 8 valid

5.7.5 Invalid data

Name:
InvalidDataByte

InvalidData1 to InvalidData8

The validity of the read values can be checked redundantly. This register indicates (by bit) which of the max. 8 read values are invalid.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	InvalidData1	0	Value 1 valid
		1	Value 1 invalid
...	...	...	...
7	InvalidData8	0	Value 8 valid
		1	Value 8 invalid

5.7.6 Data

Name:
Data1 to Data8

Each cyclic data register contains the respective pre-configured value from the M-Bus network. The data type of the data register was designed to be variable and must be specified by the user during configuration.

Information:

Because X2X Link can only transfer a certain number of bytes cyclically, the user must make his selection accordingly.

Data type	Value
(U)SINT (U)INT (U)DINT REAL	According to data type

5.7.7 Change the serial number of an M-bus slave

Name:
ChangedSNByte

This register indicates (by bit) whether one of the M-Bus slave serial numbers on the bus has changed. Only the serial numbers of the slaves accessed via the B&R interface are checked. The respective bit is toggled if a change is detected.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	SN (Slave 1)	0 -> 1	Slave1: Serial number changed
		1 -> 0	
...	...	...	...
7	SN (Slave 8)	0 -> 1	Slave8: Serial number changed
		1 -> 0	

5.7.8 M-Bus slave serial numbers

Name:
SNData1 to SNData8

These registers contain the serial numbers of the M-Bus slaves that are queried via the B&R interface. They are implemented acyclically and can be read using library AsIOAcc.

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

5.8 Flatstream communication

5.8.1 Introduction

B&R offers an additional communication method for some modules. "Flatstream" was designed for X2X and POWERLINK networks and allows data transmission 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.

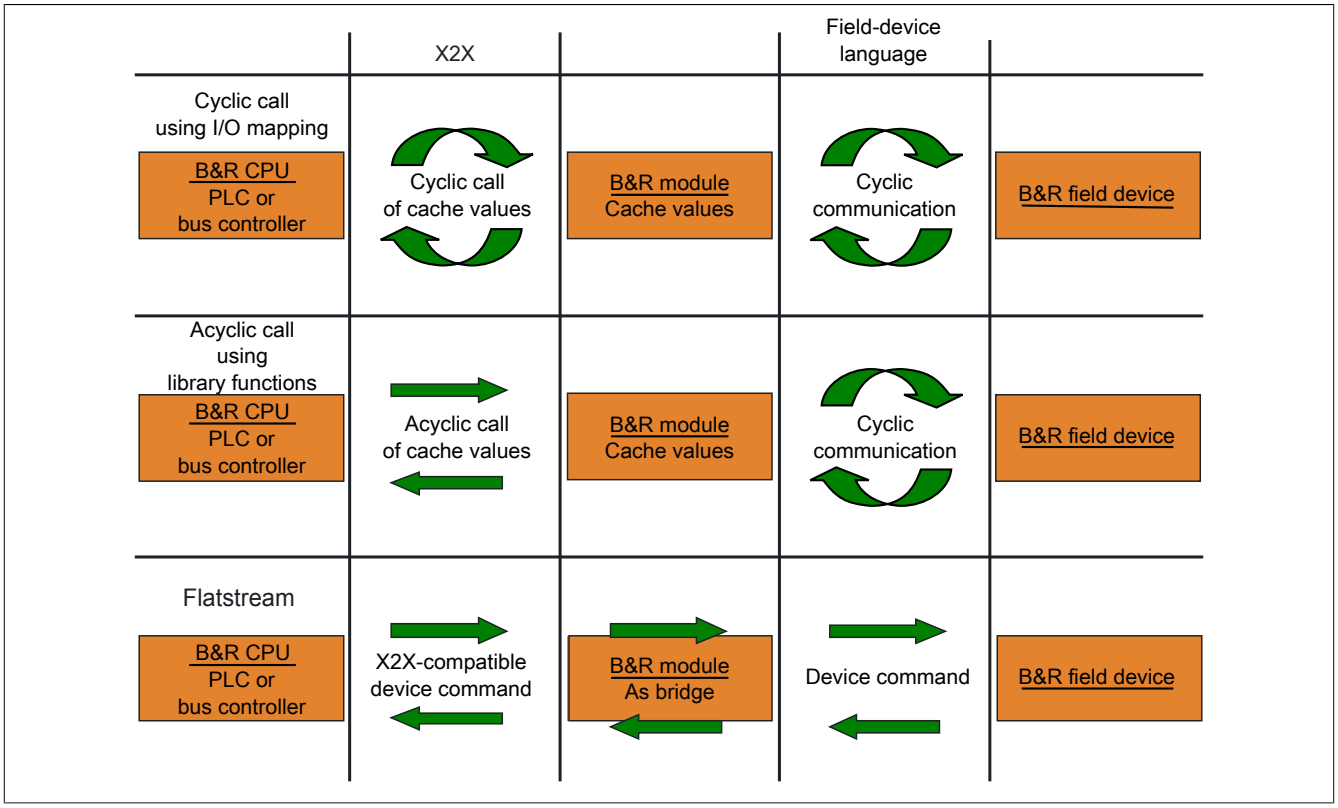


Figure 1: 3 types of communication

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

5.8.2 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 transmission was faulty and that all affected sequences must be repeated.

5.8.3 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.

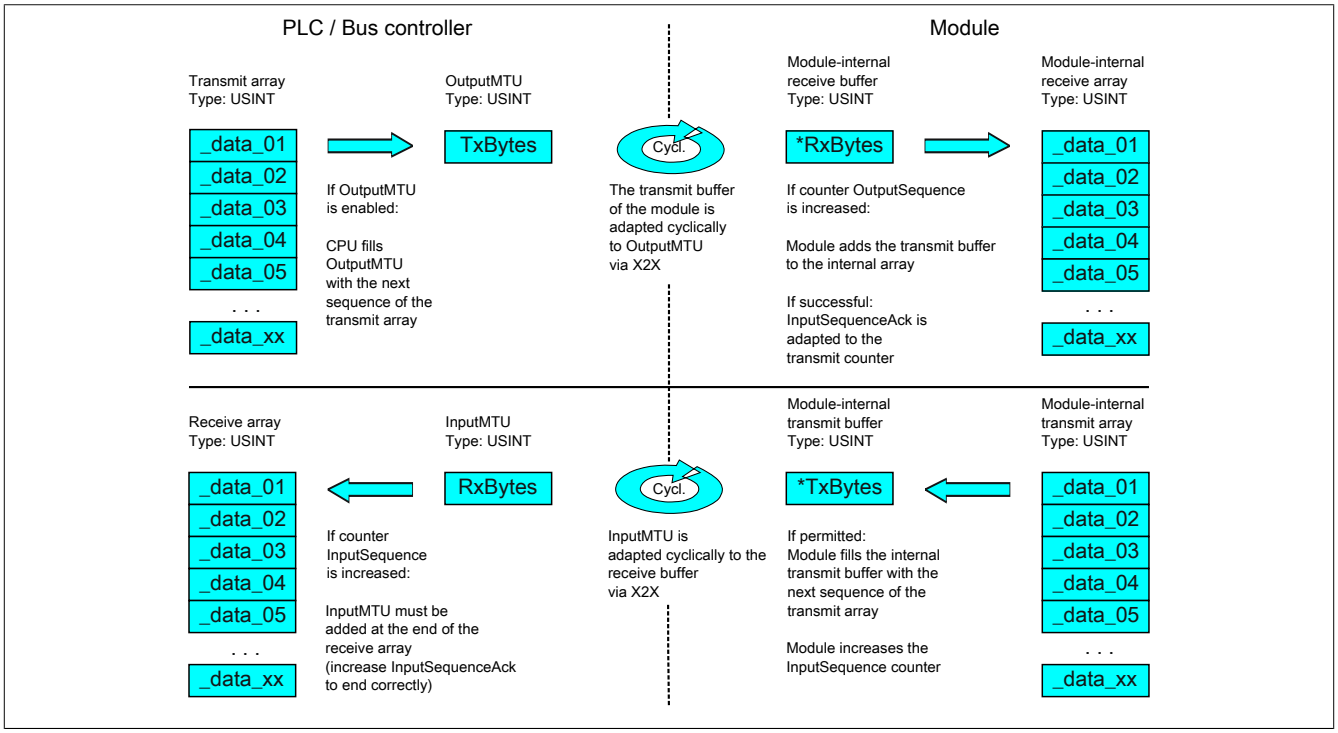


Figure 2: Flatstream communication

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.

5.8.4 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 CPU 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.

5.8.4.1 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.

5.8.4.1.1 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.

Information:

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

Data type	Values
USINT	See the module-specific register overview (theoretically: 3 to 27).

5.8.4.2 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.

5.8.4.2.1 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.

5.8.4.2.2 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.
In the user program, only the Tx and Rx bytes from the CPU 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 CPU.

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

Data type	Values
USINT	0 to 255

5.8.4.2.3 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

5.8.4.2.4 Communication status of the CPU

Name:
OutputSequence

Register "OutputSequence" contains information about the communication status of the CPU. It is written by the CPU 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 CPU. The CPU uses OutputSequenceCounter to direct the module to accept a sequence (the output direction must be synchronized when this happens).

OutputSyncBit

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

InputSequenceAck

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

InputSyncAck

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

5.8.4.2.5 Communication status of the module

Name:
InputSequence

Register "InputSequence" contains information about the communication status of the module. It is written by the module and should only be read by the CPU.

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 CPU 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 CPU. This indicates that the module is ready to receive data.

5.8.4.2.6 Relationship between OutputSequence and InputSequence

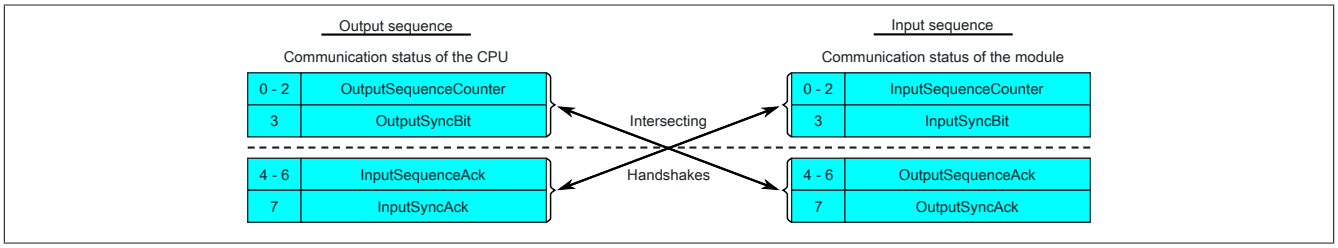


Figure 3: Relationship between OutputSequence and InputSequence

Registers "OutputSequence" and "InputSequence" are logically composed of 2 half-bytes. The low part signals to the opposite station whether a channel should be opened or if 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.

5.8.4.3 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 (CPU 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 CPU to the module.

Algorithm

1) The CPU must write 000 to OutputSequenceCounter and reset OutputSyncBit. The CPU 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 CPU registers the expected values in OutputSequenceAck and OutputSyncAck, it is permitted to increment OutputSequenceCounter. The CPU 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 CPU registers the expected values in OutputSequenceAck and OutputSyncAck, it is permitted to increment OutputSequenceCounter. The CPU 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 CPU can transmit data to the module.

Synchronization in the input direction (CPU 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 CPU.

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 CPU is not permitted to accept the current contents of InputMTU since the channel is not yet synchronized. The CPU 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 CPU is not permitted to accept the current contents of InputMTU since the channel is not yet synchronized. The CPU 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 CPU 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 CPU.

5.8.4.4 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.

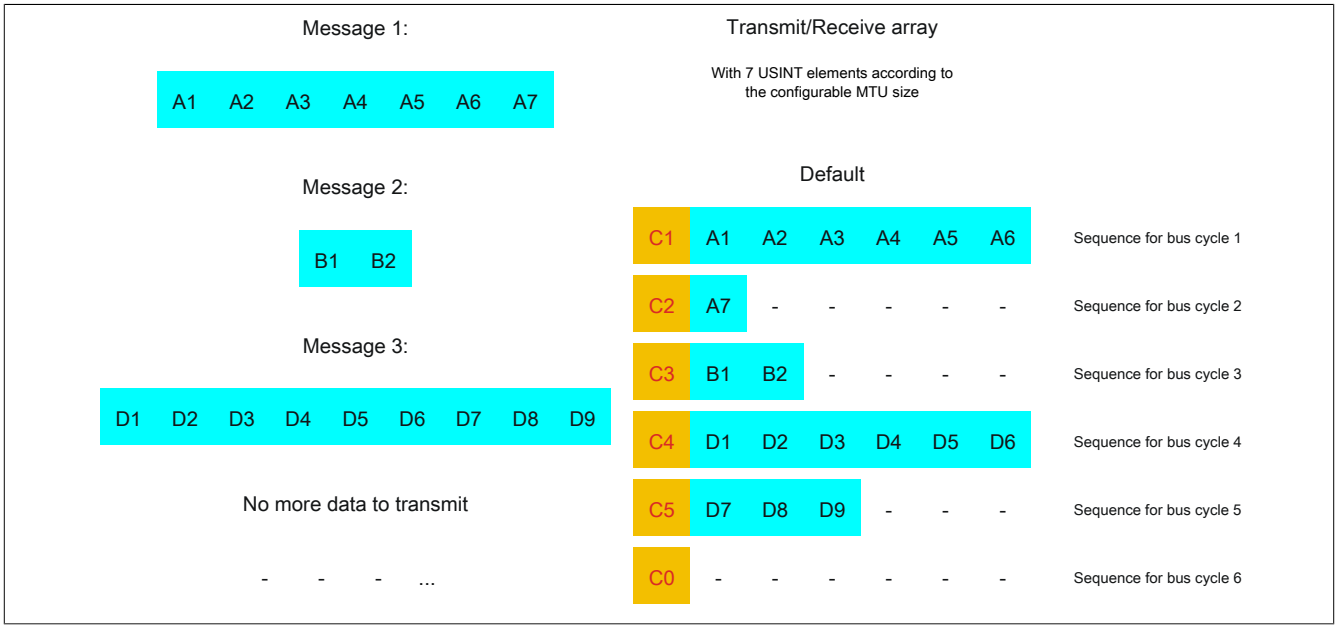


Figure 4: Transmit/Receive array (default)

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)

5.8.4.5 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.

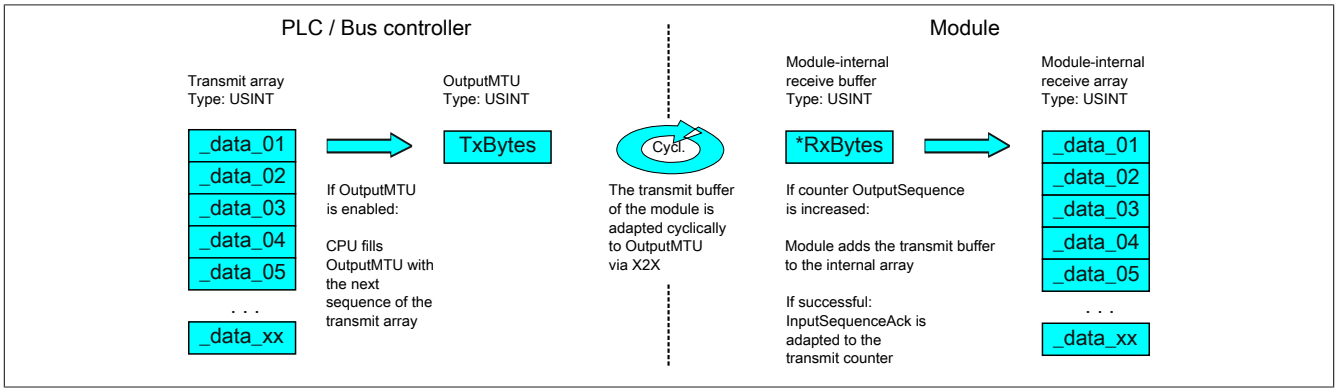


Figure 5: Flatstream communication (output)

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 CPU must check OutputSyncAck. → If OutputSyncAck = 0: Reset OutputSyncBit and resynchronize the channel. - The CPU 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 CPU must split up the message into valid segments and create the necessary control bytes. - The CPU must add the segments and control bytes to the transmit array.
2) Transmit: - The CPU 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 CPU 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 CPU 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

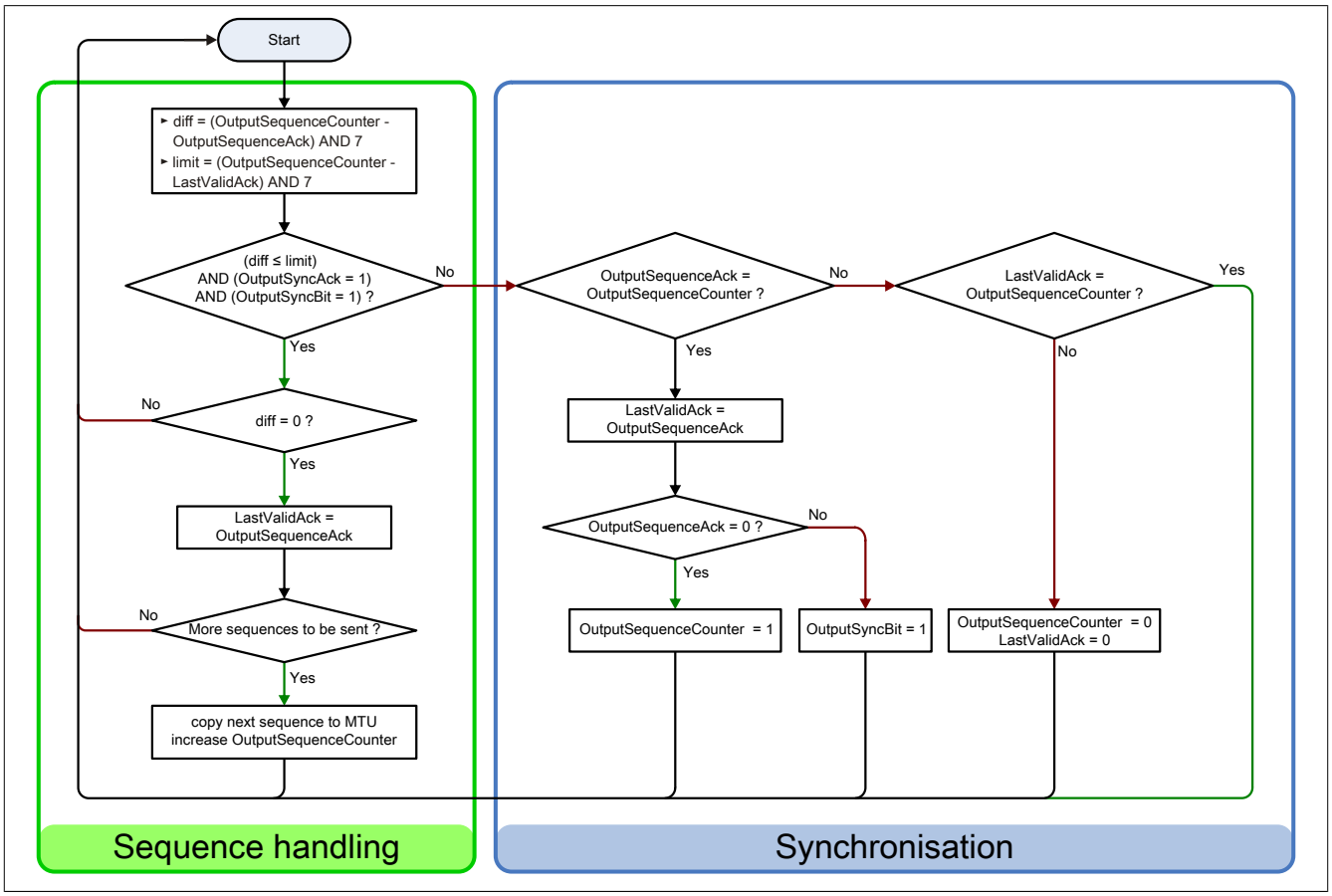


Figure 6: Flowchart for the output direction

5.8.4.6 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.

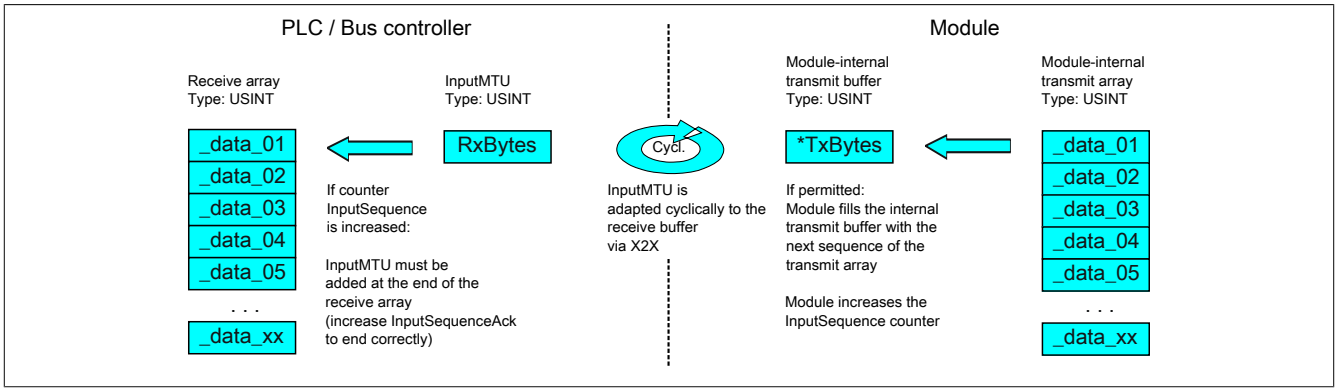


Figure 7: Flatstream communication (input)

Algorithm

0) Cyclic status query: - The CPU 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 CPU must apply data from InputMTU and append it to the end of the receive array. - The CPU 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

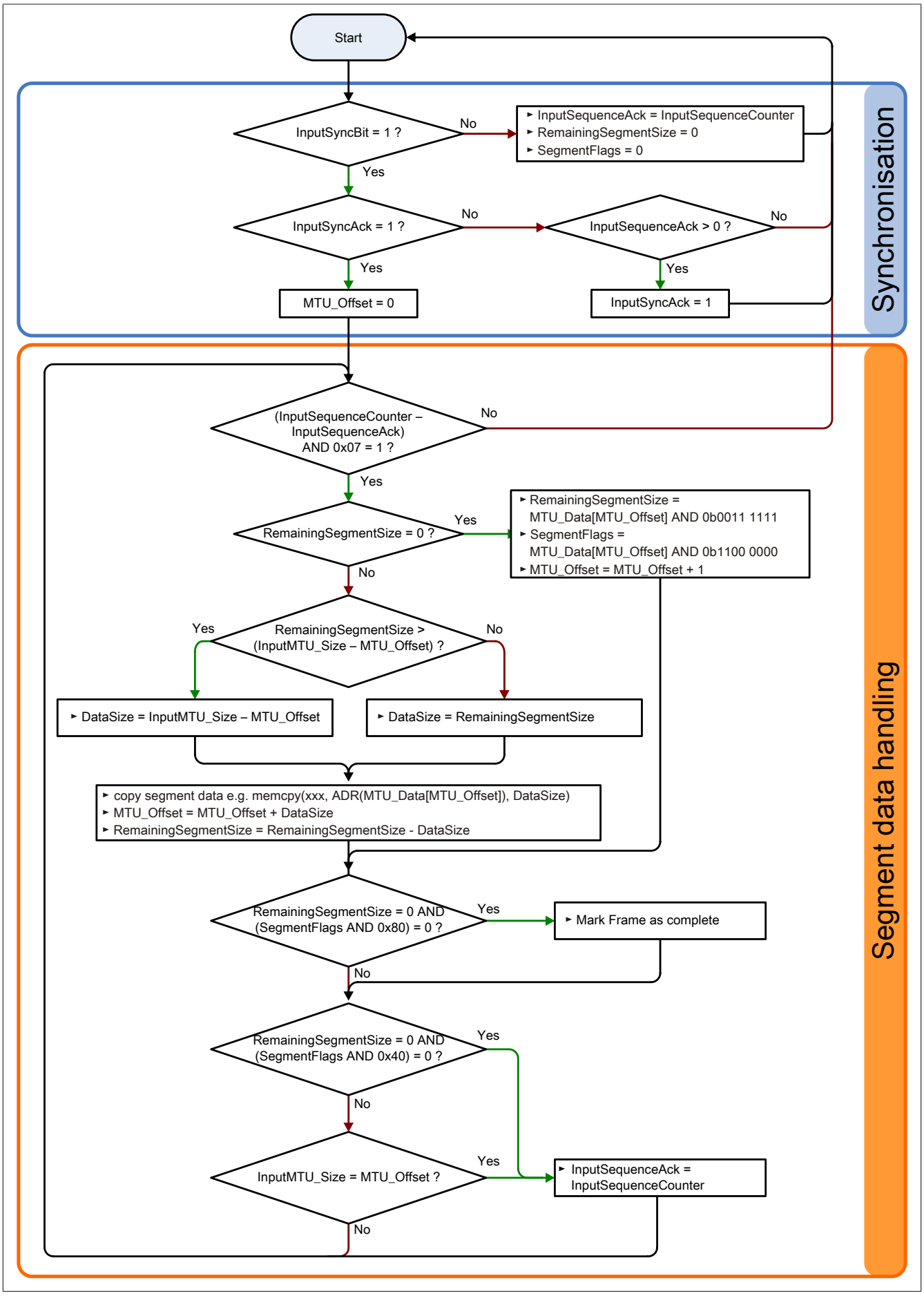


Figure 8: Flowchart for the input direction

5.8.4.7 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.

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

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.

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.

5.8.4.8 Flatstream mode

Name:
FlatstreamMode

In the input direction, the transmit array is generated automatically. This register offers 2 options to the user that allow an incoming data stream to have a more compact arrangement. 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.

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		

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.

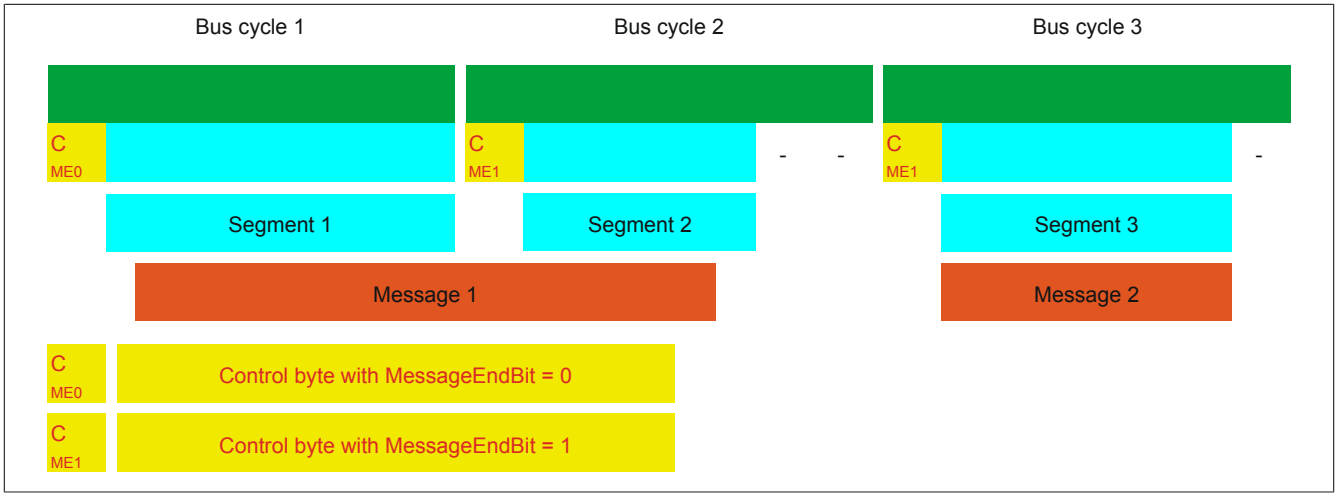


Figure 9: Message arrangement in the MTU (default)

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.

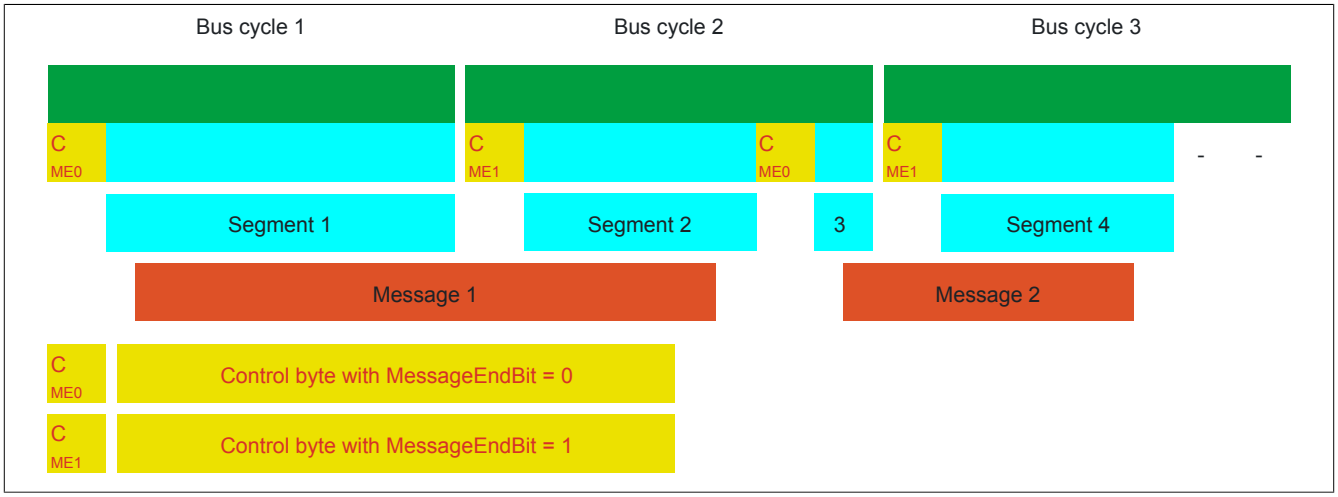


Figure 10: Arrangement of messages in the MTU (MultiSegmentMTUs)

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.

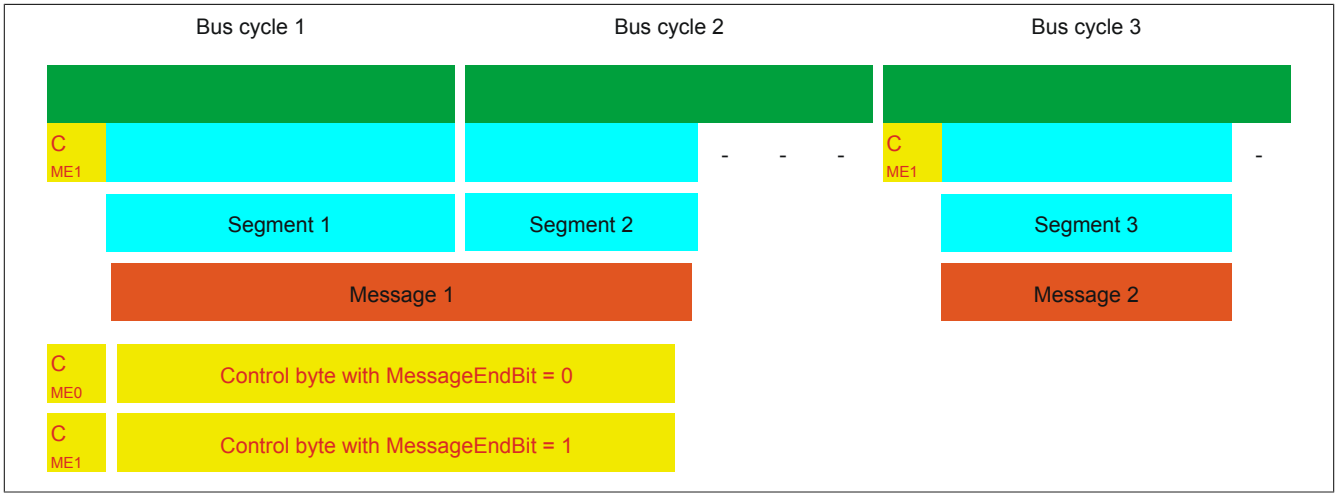


Figure 11: Arrangement of messages in the MTU (large segments)

Using both options

Using both options at the same time is also permitted.

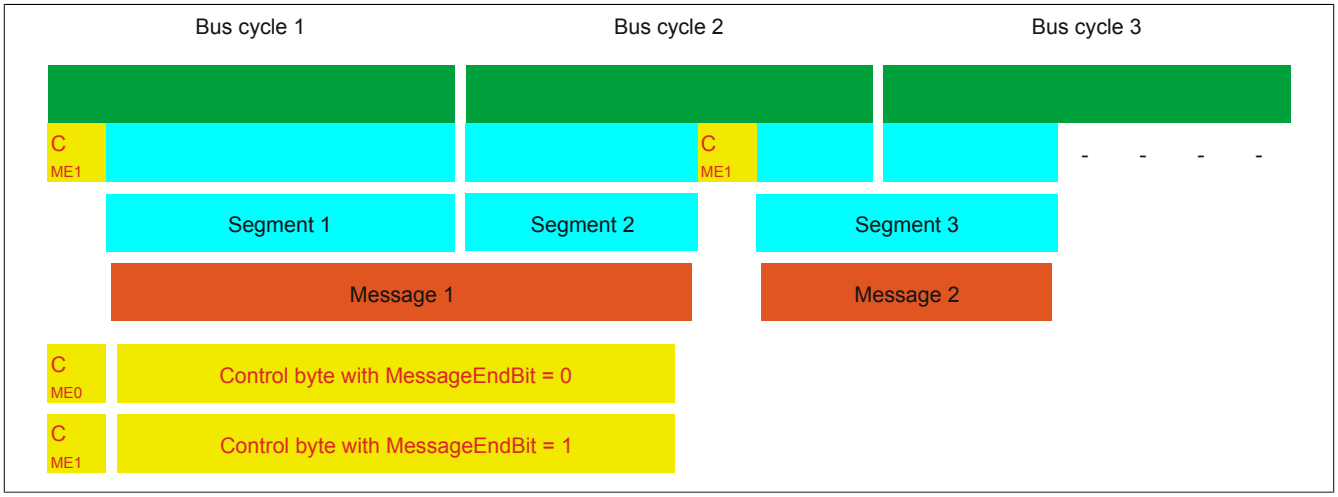


Figure 12: Arrangement of messages in the MTU (large segments and MultiSegmentMTUs)

5.8.4.9 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.

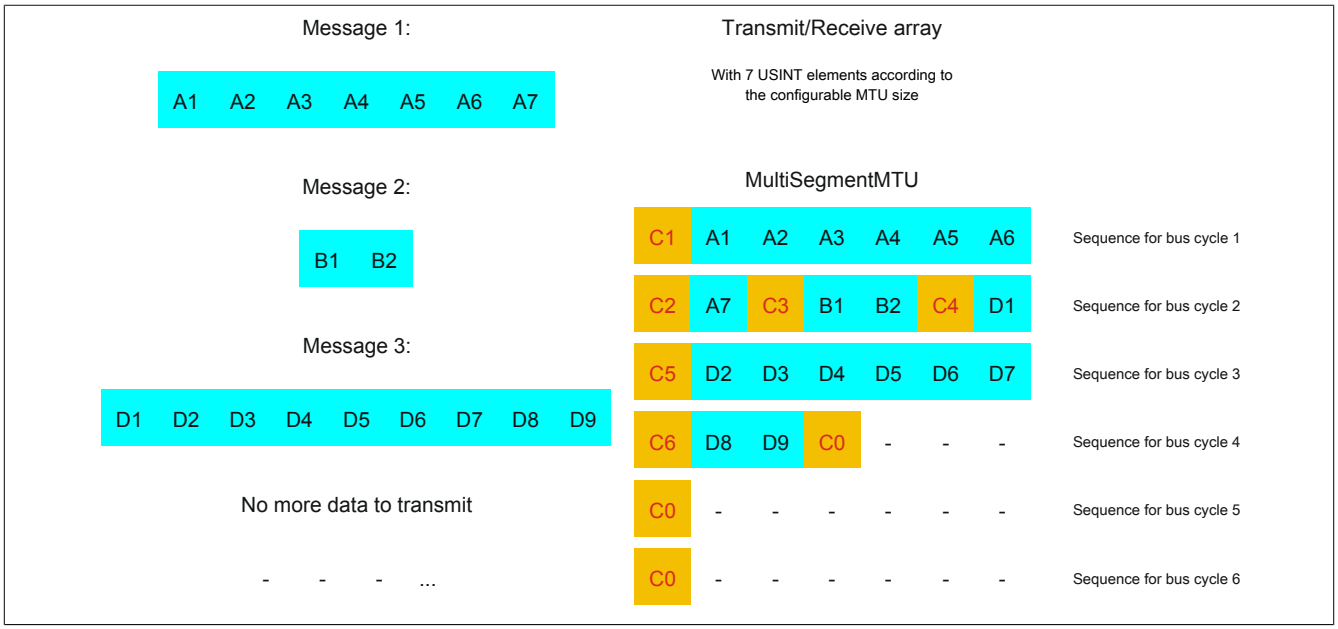


Figure 13: Transmit/receive array (MultiSegmentMTUs)

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.

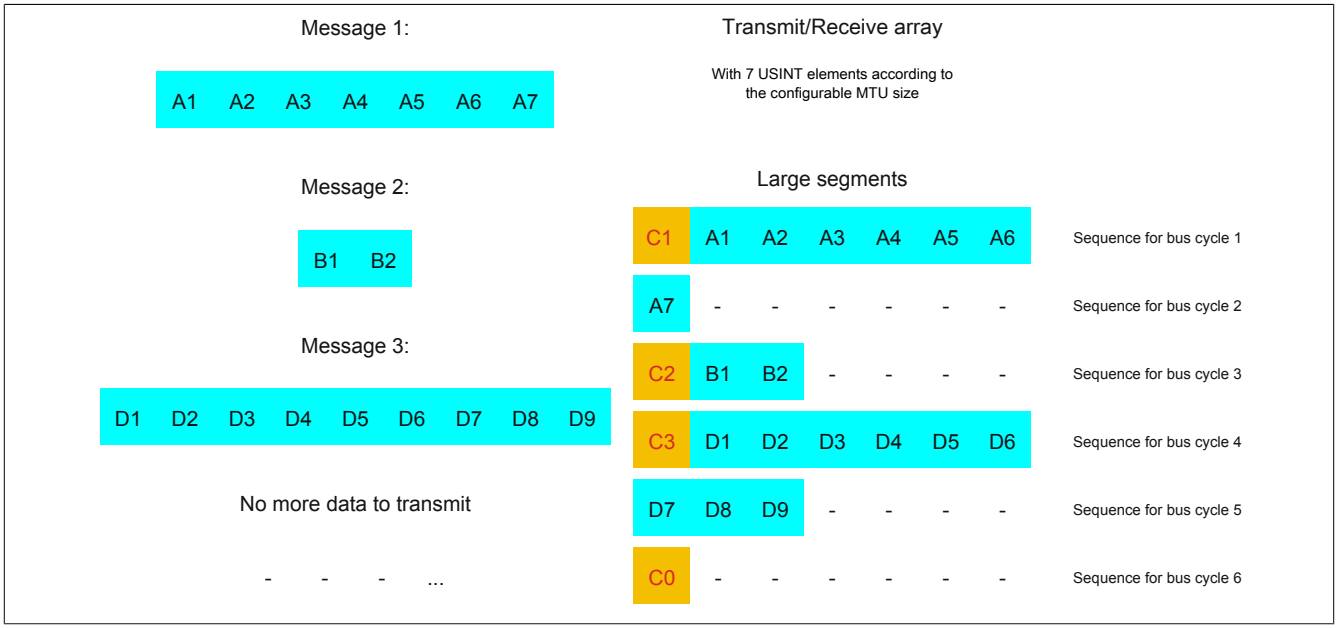


Figure 14: Transmit/receive array (large segments)

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.

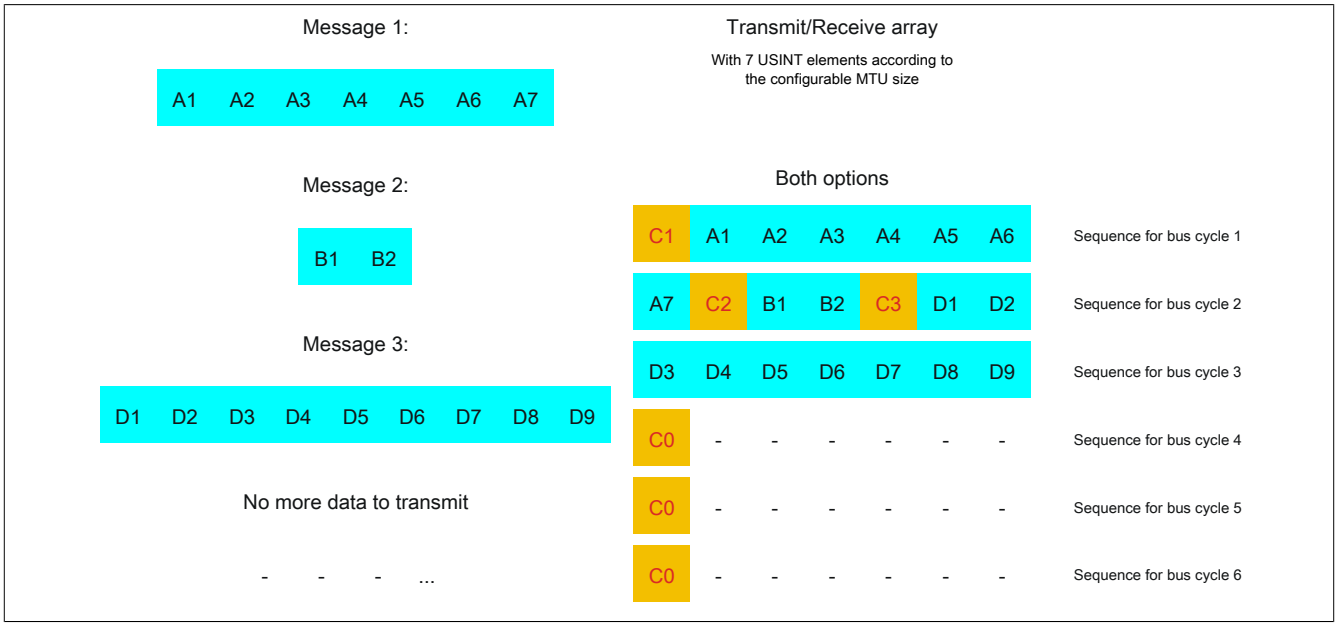


Figure 15: Transmit/receive array (large segments and MultiSegmentMTUs)

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

5.8.5 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.

5.8.5.1 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)

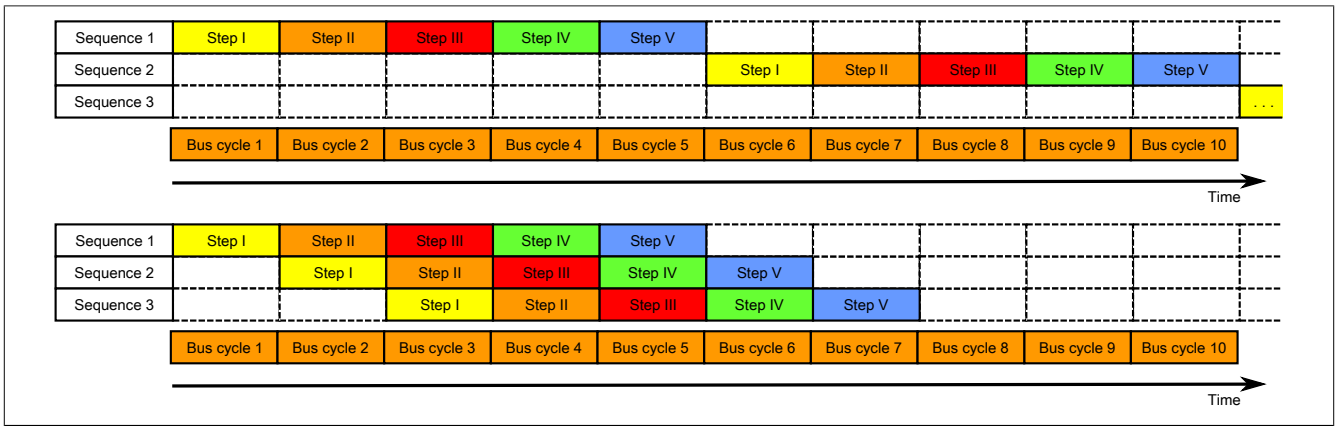


Figure 16: Comparison of transfer without/with Forward

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.

5.8.5.2 Configuration

The Forward function must only be enabled for the input direction. 2 additional configuration registers are available for doing so. 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.

5.8.5.2.1 Number of unacknowledged sequences

Name:
Forward

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

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

Data type	Values
USINT	1 to 7 Default: 1

5.8.5.2.2 Delay time

Name:
ForwardDelay

Register "ForwardDelay" is used to specify the delay time 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.

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

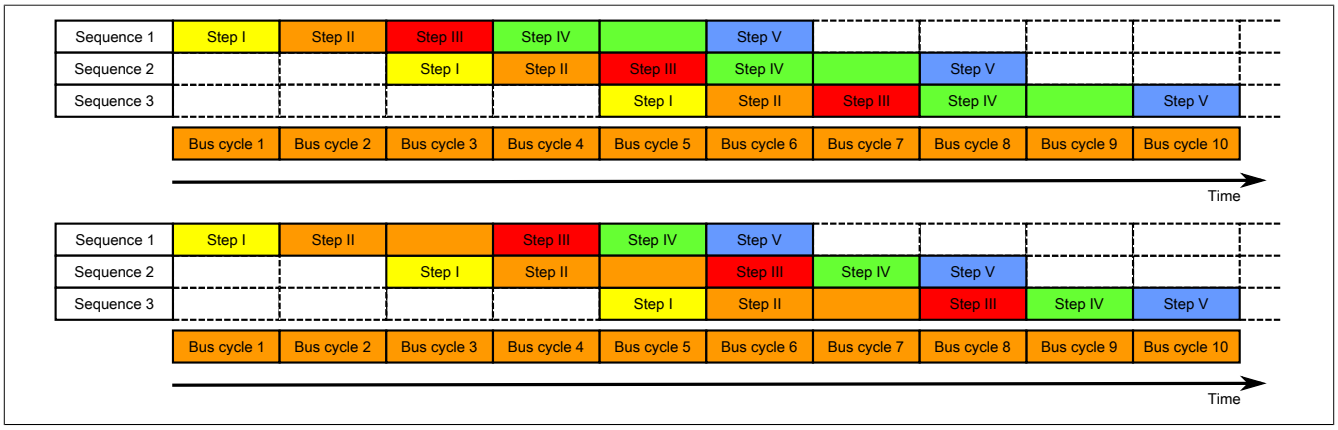


Figure 17: Effect of ForwardDelay when using Flatstream communication with the Forward function

In the program, it is important to make sure that the CPU 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 CPU has more time to process the incoming InputSequence or InputMTU.

5.8.5.3 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 CPU must check OutputSyncAck. → If OutputSyncAck = 0: Reset OutputSyncBit and resynchronize the channel. - The CPU 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 CPU must split up the message into valid segments and create the necessary control bytes. - The CPU must add the segments and control bytes to the transmit array.
2) Transmit: - The CPU must transfer the current part of the transmit array to OutputMTU. - The CPU must increase OutputSequenceCounter for the sequence to be accepted by the module. - The CPU 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 CPU 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 CPU 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 CPU must apply data from InputMTU and append it to the end of the receive array. - The CPU 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 CPU 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.

5.8.5.4 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).

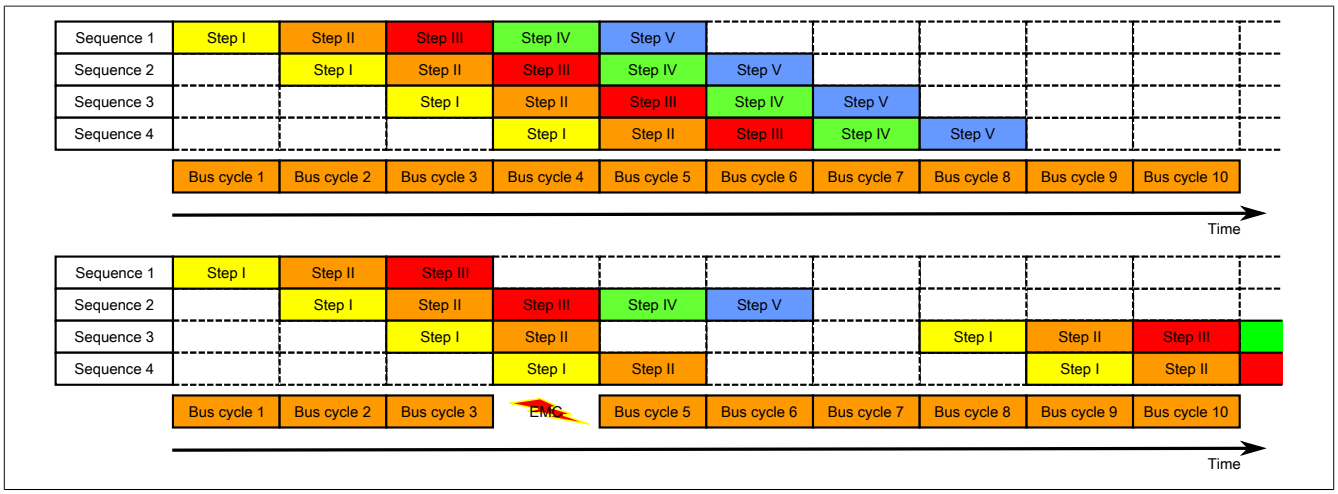


Figure 18: Effect of a lost bus cycle

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.

5.9 M-Bus 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 bus systems. Specific algorithms such as timeout and checksum monitoring are usually managed automatically. During normal operation, the user does not have direct access to these details.

Operation

The M-Bus specification recognizes four different frame types. From the application standpoint, only "long frames" are generated and transferred when using the M-Bus via FlatStreams. Due to the flexible design of the M-Bus protocol, the user must include the corresponding slave configuration with each request.

FlatStream structure		
In-/OutputSequence	Rx-/TxBytes	
(unchanged)	Control byte (unchanged)	M-Bus data (FlatStream)

5.9.1 FlatStream in output direction

FlatStream query

This standard protocol specifies that a data query via FlatStream consists of a main part and two index records. An index record is made up of an introduction containing various information and followed by a parameter block.

5.9.1.1 Introduction

The primary role of the main part is to assign a synchronization number and register the protocol type.

Note 1
When registering an undefined protocol type, the module works with the standard protocol.

Note 2
Because there is currently only one protocol type defined, the corresponding configuration bytes should be set to 1. This will allow the protocol to be expanded later without becoming incompatible with existing projects.

Bytes	Name	Value	Description
1	Frame number: For synchronization in the application	0 - 255	The frame number is repeated in the module's response. This allows the later response from the module to be distinctly attributed to the request.
2	Index Record Count "i"	2!	Number of subsequent index records
3	Protocol type	0	Native M-Bus (level converter mode) - see "Native M-Bus"
		1	Data query (raw data / parameters)
4	Reserved	1!	
...	Index record (configuration)		
...	Index record (data query)		

Native M-Bus

The "Native M-Bus" protocol type provides universal communication within the M-Bus network. It can be used to assemble and send M-Bus frames in the application.

A conventional data query is possible using a raw data or parameter query.

5.9.1.2 Index record 0

Configuration block

The interface parameters for defining the module's behavior in the M-Bus network must be chosen configuration part.

Information:

With the standard protocol, the index record must be resent with each request for configuration.

Introduction

Bytes	Name	Value	Description
1	Index record type	0!	Module interface configuration
2	Counter (config parameter)	5!	Number of subsequent M-Bus parameters
3	Length of parameter block - Low	19!	Length of index record description
4	Length of parameter block - High	0!	Length of index record description

Parameter block

Configuration parameter 0 - Addressing type

Bytes	Name	Value	Description
1	Parameter number	0!	
2	Length	1!	
3	Addressing type	1	Addressing via primary address
		2	Addressing via secondary address

Configuration parameter 1 - Address

Bytes	Name	Value	Description
4	Parameter number	1!	
5	Length	4!	
6	Address - LowLow	1 - 255	Primary address
7	Address - LowHigh	0 - 255	0!, if primary addressing
8	Address - HighLow	0 - 255	0!, if primary addressing
9	Address - HighHigh	0 - 255	0!, if primary addressing

Configuration parameter 2 - Transfer rate

Bytes	Name	Value	Description
10	Parameter number	2!	
11	Length	2!	
12	Transfer rate Low	0 - 255	Verified transfer rates 300 bit/s, 2400 bit/s, 9600 bit/s
13	Transfer rate High	0 - 255	

Configuration parameter 3 - Timeout offset

Bytes	Name	Value	Description
14	Parameter number	3!	
15	Length	1!	
16	TimeoutOffset	0 - 255	Additional time for timeout monitoring on the M-Bus (Resolution: 10 ms)

Configuration parameter 4 - Extra frames

Bytes	Name	Value	Description
17	Parameter number	4!	
18	Length	1!	
19	M-Bus options	Bit 0 … 1	Send Init frame
		Bit 1 … 1	Send application reset
		Bit 6 … 1	Set frame count bit[1)]
		Bit 7 … 1	Request media and version

1) Some M-Bus slaves use this bit to switch to a another data set.

5.9.1.3 Index record 1

Data query block

The M-Bus parameters to be retrieved from the memory of the M-Bus slaves are requested in the request part. The user can request certain parameters from the slave or the entire slave memory.

Introduction

Bytes	Name	Value	Description
1	Index record type	1!	Data request for M-Bus slave
2	Counter (data parameter) = (d + 1)	0	• Communication via native M-Bus • Read out M-Bus raw data
		1 - 20	Number of parameters to read out
3	Length of subsequent block - Low	0 - 255	• Length of M-Bus frame to be sent • Length of the parameter block • 0! with raw data query
4	Length of subsequent block - High	0 - 255	• Length of M-Bus frame to be sent • Length of the parameter block • 0! with raw data query
...	Depending on request: • Native M-Bus frame • Parameter block		*Not needed if raw data query = 0*

M-Bus frame

M-Bus frame to be sent

Bytes	Name	Value	Description
1	TxByte 1	0 - 255	Byte 1 in output direction
2	TxByte 2	0 - 255	Byte 2 in output direction
n	TxByte n	0 - 255	Byte n in output direction

Parameter block

Data parameter 0

Bytes	Name	Value	Description
1	Parameter number	0!	
2	Data index	1 - 48	Data index in the M-Bus frame

Data parameter 1

Bytes	Name	Value	Description
2	Parameter number	1!	
3	Data index	1 - 48	Data index in the M-Bus frame

Data parameter d

Bytes	Name	Value	Description
...	Parameter number	d!	
...	Data index	1 - 48	Data index in the M-Bus frame

5.9.2 FlatStream in input direction

FlatStream response

The standard protocol has three different responses according to the request.

5.9.2.1 Error response

The error response occurs when the module receives an invalid or incomplete request.

Byte	Name	Value	Description
1	Frame number: For synchronization in the application	0 - 255	This frame number is repeated in the module's response. In this way, the response of the module can be clearly assigned to the request.
2	Error code - LowLow	0 - 255	See the error code table.
3	Error code - LowHigh	0 - 255	See the error code table.
4	Error code - HighLow	0 - 255	See the error code table.
5	Error code - HighHigh	0 - 255	See the error code table.
6	Additional information - LowLow	0 - 255	Optional
7	Additional information - LowHigh	0 - 255	Optional
8	Additional information - HighLow	0 - 255	Optional
9	Additional information - HighHigh	0 - 255	Optional

Error codes

Error code and name	Error description
0x11111111	An M-Bus counter is not responding to a data request. This can have different causes: • The counter is not connected. • The counter is defective. • A counter with the selected addressing parameters does not exist on the bus.
0x22222222	This error code is transmitted if addressing via the secondary address and the selected counter does not respond.
0x33333333	If an invalid transfer rate is sent along with the stream, it is not evaluated. No M-Bus frame is sent; the Flatstream interface responds directly with this error code.
0x44444444	If a collision occurs on the bus while querying the data, the data request is ended and this error code is returned.
0x55555555	Communication aborted due to overflow (see bit 4 in section "M-Bus state" on page 14)
0x66666666	Before the data is evaluated by the M-Bus counter, the checksum of the M-Bus frame is checked. If this is not correct, the received data is not processed further; instead, the error code is transmitted to the CPU.
0x77777777	The stream (CPU → IOM) is not correct. It is possible that a parameter number is not correct. The stream is checked very carefully, so an incorrect stream is never used.
0x88888888	Overload during M-Bus communication
0x99999999	Communication aborted due to level converter (see bit 5 in section "M-Bus state" on page 14)
0xAAAAAAAA	Interpretation of slave data not possible. The M-Bus slave being used is not compatible with the parameter query. The M-Bus slave must be implemented using the native M-Bus protocol or a raw data query.

Additional information

Additional information	Error description
0x00000001	Number of index records less than 2
0x00000002	Invalid stream length
0x00000004	Invalid index numbers
0x00000008	Incorrect number of parameters per index record
0x00000010	Index length too small
0x00000020	Incorrect parameter number for index record 0
0x00000040	Incorrect parameter length for index record 0
0x00000080	Invalid addressing type
0x00000100	Invalid address
0x00000200	Invalid transfer rate
0x00000400	Invalid timeout offset
0x00000800	Invalid additional frame configuration

5.9.2.2 Response - native M-Bus

This response corresponds with a successfully transferred M-Bus frame created within the application.

Bytes	Name	Value	Description
1	Frame number: For synchronization in the application	0 - 255	The frame number is repeated in the module's response. This allows the response from the module to be distinctly attributed to the request.
2	Reserved	0	
...	Response		

Response

Bytes	Name	Value	Description
1	RxByte 1	0 - 255	Byte 1 in input direction
2	RxByte 2	0 - 255	Byte 2 in input direction
n	RxByte n	0 - 255	Byte n in input direction

5.9.2.3 Response - Raw data

The raw data response is sent if the M-Bus slave's entire memory is requested.

Bytes	Name	Value	Description
1	Frame number: For synchronization in the application	0 - 255	The frame number is repeated in the module's response. This allows the response from the module to be distinctly attributed to the request.
2	M-Bus status	0 - 255	Status info from M-Bus header
3	Raw data frame	0 - 255	Includes all bytes sent by the M-Bus slave.
...		0 - 255	

5.9.2.4 Response - Parameters

The parameter response is sent if one or more parameters from an M-Bus slave have been requested.

Bytes	Name	Value	Description
1	Frame number: For synchronization in the application	0 - 255	The frame number is repeated in the module's response. This allows the response from the module to be distinctly attributed to the request.
2	M-Bus status	0 - 255	Status info from M-Bus header
3	Parameter count "p"	0 - 255	Number of parameters received
4	M-Bus address	0 - 255	Primary address
5	Serial number - LowLow	0 - 255	Secondary address
6	Serial number - LowHigh	0 - 255	Secondary address
7	Serial number - HighLow	0 - 255	Secondary address
8	Serial number - HighHigh	0 - 255	Secondary address
9	VendorID – Low \ Version	0 - 255	See "MBus option (IndexRecord 0)" on page 45
10	VendorID – High \ Medium	0 - 255	See "MBus option (IndexRecord 0)" on page 45
11	Data structure (M-Bus)	1	Fixed data structure
		2	Variable data structure
...	Received parameter 1 through p		*Not needed if parameter count = 0*

Received parameter

Bytes	Name	Value	Description
1	Medium	0 - 255	Medium of subsequent counter value
2	Index	0 - 255	Index of subsequent counter value
3	Data length	1 - 8	Length of the counter value
		255	If the parameter number is invalid
4	DIF	0 - 255	0!, if fixed data structure
5	VIF	0 - 255	0!, if fixed data structure
6	Counter value	0 - 255	LowLowLowLowLowLowLowLow
...		...	...
13		0 - 255	HighHighHighHighHighHighHigh

5.10 Minimum cycle time

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

Minimum cycle time
200 µs

5.11 Minimum I/O update time

The minimum I/O update time 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
1 s