X20CS1012

1 Allgemeines

Der M-Bus Master ist als einfachbreites Modul ausgeführt und kann beliebig innerhalb eines X20 I/O-Systems gesteckt werden. Er ist damit dezentral bei verteilten Topologien einsetzbar. Vom M-Bus Master werden die Übertragungsraten 300, 2400 und 9600 Bit/s unterstützt, wobei bis zu 64 Slaves, die über den M-Bus versorgt werden, angeschlossen werden können.

Der M-Bus (Meter-Bus) ist ein relativ einfacher Feldbus zur Verbrauchsdatenerfassung, wie etwa von Strom- oder Wärmezählern. Er basiert auf einer verpolungssicheren Zweidrahtleitung und arbeitet nach dem Master-Slave Prinzip.

- Versorgung für bis zu 64 Slaves am M-Bus
- Kommunikationsschnittstelle dezentral einsetzbar

2 Bestelldaten

Bestellnummer	Kurzbeschreibung	Abbildung
	Kommunikation im X20 Elektronikmodul	
X20CS1012	X20 Schnittstellenmodul, 1 M-Bus Master Schnittstelle, integrierte Slave-Versorgung	
	Erforderliches Zubehör	
	Busmodule	
X20BM11	X20 Busmodul, 24 VDC codiert, interne I/O-Versorgung durchverbunden	
X20BM15	X20 Busmodul, mit Knotennummernschalter, 24 VDC codiert, interne I/O-Versorgung durchverbunden	
	Feldklemmen	
X20TB12	X20 Feldklemme, 12-polig, 24 VDC codiert	

Tabelle 1: X20CS1012 - Bestelldaten

3 Technische Daten

Bestellnummer	X20CS1012
Kurzbeschreibung	
Kommunikationsmodul	1 M-Bus Master zur Ansteuerung von bis zu 64 Slaves
Allgemeines	
B&R ID-Code	0xCABF
Statusanzeigen	Datenübertragung, M-Bus Versorgung, Betriebszustand, Modulstatus
Diagnose	
Modul Run/Error	Ja, per Status-LED und SW-Status
Datenübertragung	Ja, per Status-LED
M-Bus Versorgung	Ja, per Status-LED und SW-Status
Leistungsaufnahme	
Bus	0,2 W
I/O-intern	0,35 W + (Anzahl der Slaves * 0,08 W)
Modulverlustleistung	0,55 W + (Anzahl der Slaves * 0,006 W)
Zusätzliche Verlustleistung durch Aktoren (ohmsch) [W]	-
Zulassungen	
CE	Ja
EAC	Ja
UL	cULus E115267 Industrial Control Equipment
HazLoc	cCSAus 244665 Process Control Equipment for Hazardous Locations Class I, Division 2, Groups ABCD, T5
ATEX	Zone 2, II 3G Ex nA nC IIA T5 Gc IP20, Ta (siehe X20 Anwenderhandbuch) FTZÚ 09 ATEX 0083X
Isolationsspannung zwischen M-Bus und X2X Link	500 VDC, 1 min
Schnittstellen	
Schnittstelle	
Typ	M-Bus Master
Ausführung	Kontaktierung über 12-polige Feldklemme X20TB12
max. Reichweite	Siehe Abschnitt "M-Bus"
Übertragungsrate	300, 2400 oder 9600 Bit/s
Anzahl Slaves	max. 64
Innenwiderstand des Masters	max. 6 Ω
Busspannung Mark bei 0 mA	I/O-Versorgungsspannung + (11,5 bis 13,5) V
Busspannung Absenkung bei Space	12 bis 13,5 V
Überstromabschaltung	250 mA ±10%
Bitschwelle	6 bis 9 mA
Kollisionsschwelle	24 bis 36 mA
Empfänger Nachregelzeit	max. 10 s [1)]
Busleitung	Geschirmt oder ungeschirmt
Elektrische Eigenschaften	
Potenzialtrennung	M-Bus zu Bus getrennt M-Bus zu I/O-Versorgung nicht getrennt
Einsatzbedingungen	
Einbaulage	
waagrecht	Ja
senkrecht	Ja
Aufstellungshöhe über NN (Meeresspiegel)	
0 bis 2000 m	Keine Einschränkung
>2000 m	Reduktion der Umgebungstemperatur um 0,5°C pro 100 m
Schutzart nach EN 60529	IP20
Umgebungsbedingungen	
Temperatur	
Betrieb	
waagrechte Einbaulage	-25 bis 60°C
senkrechte Einbaulage	-25 bis 50°C
Derating	-
Lagerung	-40 bis 85°C
Transport	-40 bis 85°C
Luftfeuchtigkeit	
Betrieb	5 bis 95%, nicht kondensierend
Lagerung	5 bis 95%, nicht kondensierend
Transport	5 bis 95%, nicht kondensierend
Mechanische Eigenschaften	
Anmerkung	Feldklemme 1x X20TB12 gesondert bestellen Busmodul 1x X20BM11 gesondert bestellen
Rastermaß	12,5 $^{+0,2}$ mm

Tabelle 2: X20CS1012 - Technische Daten

1) Nach jeder Laständerung am M-Bus (z. B. zu- oder abschalten von Slaves)

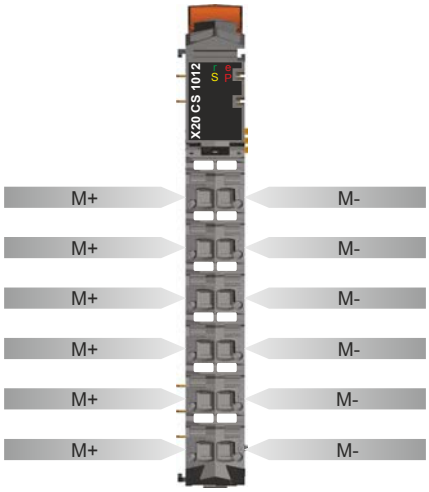
4 Status LEDs

Für die Beschreibung der verschiedenen Betriebsmodi siehe X20 System Anwenderhandbuch, Abschnitt "Zusätzliche Informationen - Diagnose-LEDs".

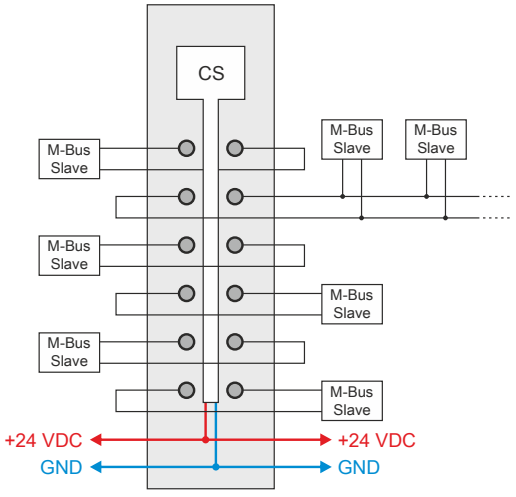
Abbildung	LED	Farbe	Status	Beschreibung
	r	Grün	Aus	Modul nicht versorgt
			Single Flash	Modus UNLINK
			Double Flash	Modus BOOT (während Firmware-Update)[1)]
			Blinkend	Modus PREOPERATIONAL
			Ein	Modus RUN
	e	Rot	Aus	Modul nicht versorgt oder alles in Ordnung
			Ein	Fehler- oder Resetzustand
	e + r	Rot ein / grüner Single Flash		Firmware ist ungültig
	S	Gelb	Aus	Keiner der Slaves sendet Daten
			Ein	Mindestens ein Slave sendet Daten über den M-Bus
	P	Rot	Aus	M-Bus Versorgung in Ordnung
			Ein	Auf dem M-Bus ist ein Kurzschluss oder eine Überlast aufgetreten

1) Je nach Konfiguration kann ein Firmware-Update bis zu mehreren Minuten benötigen.

5 Anschlussbelegung



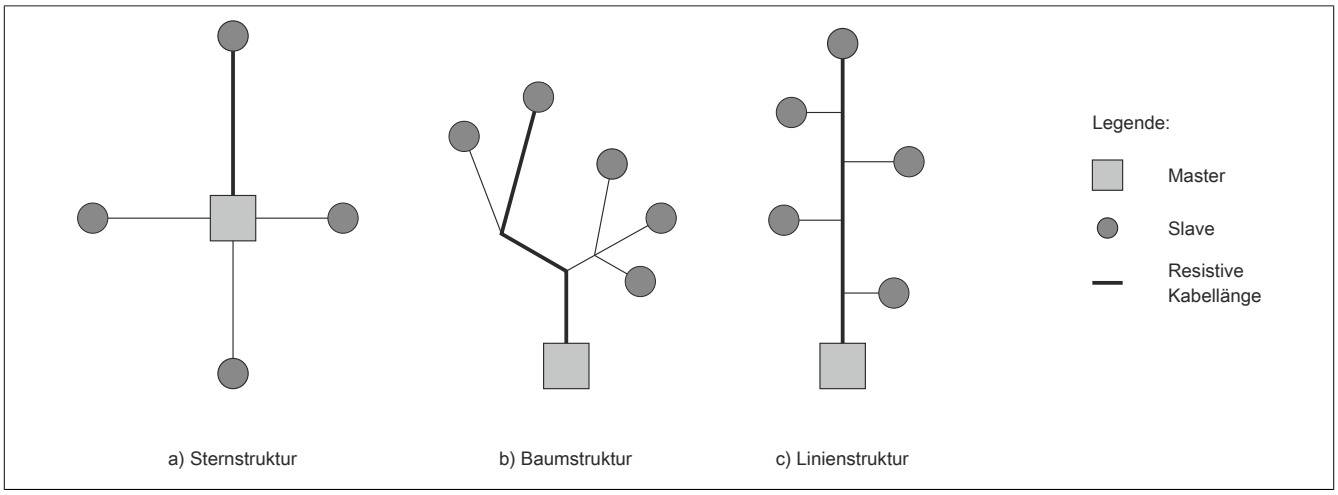
6 Anschlussbeispiel



7 M-Bus

7.1 Bustopologie

Die Bustopologie hat einen wesentlichen Einfluss auf die maximale Ausdehnung eines M-Bus Netzwerkes. Allgemein gilt, dass eine Sternstruktur einer Baumstruktur und diese einer Linienstruktur vorzuziehen ist. Ferner gilt, dass das gleichmäßige Anschließen der Slaves an den Bus bessere Werte liefert, als wenn die Slaves bei Festhalten sämtlicher anderer Parameter alle am Ende der Zweige angeschlossen werden.



7.2 Kabelquerschnitt

Das verwendete Kabel besitzt eine bestimmte Kapazität und einen bestimmten Widerstand, die einen Einfluss auf den Betrieb des Busses ausüben. Der resistive Einfluss des Kabels besteht darin, dass an der Leitung Spannung abfällt, die somit nicht mehr zur Versorgung des Busses zur Verfügung steht. Die Spannung an den Slaves, darf weder beim Senden des Masters zum Slave, noch in umgekehrter Richtung weniger als 12 V betragen, um eine ausreichende Spannungsversorgung zu gewährleisten. Entscheidend ist hierbei der längste Zweig eines Netzes, dessen Länge als resistive Kabellänge bezeichnet wird.

Die Kapazität des Kabels führt zu Signalverzerrungen bei der Datenübertragung, da die Anstiegsgeschwindigkeiten der steigenden und fallenden Flanke verzögert werden. Ersetzt man zum Beispiel einen 3 km langen Zweig eines Netzes durch zwei 1,5 km lange Zweige, so erhält man eine Signalverbesserung. Die Gesamtausdehnung des Netzes wird als kapazitive Kabellänge bezeichnet (Summe aller Segmentlängen).

Information:

Der maximal zulässige Leitungswiderstand (längste Schleife) beträgt 250 Ω.

Die maximal zulässige Leitungskapazität für den gesamten Bus beträgt 500 nF.

7.3 Sendestrom und Bitschwelle

Die Bitschwelle des Masters liegt typisch bei 7,5 mA. Somit ergibt sich bei einem Slave-Sendestrom von 15 mA die geringste bzw. bei 11 oder 20 mA die größte Signalverzerrung.

7.4 Übertragungsrate

Je geringer die Übertragungsrate desto weniger spielt die durch Kabelkapazität und Bitschwelle verursachte Signalverzerrung eine Rolle.

Information:

Ab einer Gesamtbuslänge >1 km sind die Slaves mit einer Baudrate <9600 Bit/s zu betreiben.

7.5 Berechnung der resistiven Buslänge

Um eine ausreichende Spannungsversorgung von 12 V am M-Bus zu gewährleisten, muss die resistive Kabellänge berechnet werden. Entscheidend ist hierbei das längste Segment zwischen Master und Slave.

Ohne Berücksichtigung eines erhöhten Busstroms durch Defekt eines Empfängers, wird die resistive Buslänge mit folgender Formel berechnet:

$$L_{res} = \frac{V_{I/O} - (n * 0{,}0015 + 0{,}02) * 6 - 12{,}6}{(n * 0{,}0015 + 0{,}02) * R_L} * 1000$$

L_{res} ... Resistive Buslänge [m]
n ... Anzahl der Slaves (alle am Ende der Leitung)
R_L ... Leitungswiderstand (Schleifenwiderstand [Ω/km])
$V_{I/O}$ ... I/O-Versorgungsspannung [V]

Beispiele für die resistive Buslänge:

Nr.	Beispiel	Maximale resistive Buslänge
1	• 64 Slaves (alle am Ende der Leitung) • 19,2 V I/O-Versorgungsspannung • 0,5 mm² Leitungsquerschnitt	675 m
2	• 64 Slaves (alle am Ende der Leitung) • 28,8 V I/O-Versorgungsspannung • 1,5 mm² Leitungsquerschnitt	5340 m

7.6 Berücksichtigung der kapazitiven Buslänge

Die Gesamtausdehnung des Netzes wird als kapazitive Buslänge bezeichnet (Summe aller Segmentlängen). Die kapazitive Buslänge hängt von zwei Faktoren ab:

- Kapazitätsbelag des Kabels
- Übertragungsrate

Kapazitätsbelag des Kabels

Je geringer der Kapazitätsbelag eines Kabels ist, desto höher ist die kapazitive Buslänge.

Übertragungsrate

Je geringer die Übertragungsrate eines M-Bus Systems ist, desto höher ist die kapazitive Buslänge.

Beispiel für ein Kabel mit einem Kapazitätsbelag von 50 nF/km:

Übertragungsrate	Kapazitive Buslänge
9600 Bit/s	1 km
2400 Bit/s	4 km
300 Bit/s	10 km

7.7 Businstallation

Zur Businstallation werden üblicherweise Kabel mit paarweise verdrillten Adern und einem Querschnitt von 0,5 mm² bis 1,5 mm² verwendet (laut Norm: J-Y(ST)Y nx2x0,8). Bei geschirmtem Kabel ist der Schirm nur einseitig am Modul zu erden. An den Slaves muss der Schirm für DC und niederfrequente Signale hochohmig sein.

7.8 Repeater

Durch den Einsatz von Repeatern kann die Ausdehnung des M-Bus Netzwerkes erhöht werden.

8 Registerbeschreibung

8.1 Allgemeine Datenpunkte

Neben den in der Registerbeschreibung beschriebenen Registern verfügt das Modul über zusätzliche allgemeine Datenpunkte. Diese sind nicht modulspezifisch, sondern enthalten allgemeine Informationen wie z. B. Seriennummer und Hardware-Variante.

Die allgemeinen Datenpunkte sind im X20 System Anwenderhandbuch, Abschnitt "Zusätzliche Informationen - Allgemeine Datenpunkte" beschrieben.

8.2 Funktionsmodell 0 - Standard

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Modulkonfiguration						
774	CfO_FunctionModel	UINT				•
M-Bus - Konfiguration						
Index * 16 + 767	CfO_LengthData1 bis CfO_LengthData8	USINT				•
Index * 16 + 775	CfO_BaudData1 bis CfO_BaudData8	USINT				•
Index * 16 + 761	CfO_PAdrData1 bis CfO_PAdrData8	USINT				•
Index * 16 + 765	CfO_IndexData1 bis CfO_IndexData8	USINT				•
Index * 16 + 773	CfO_ReqTimeData1 bis CfO_ReqTimeData8	USINT				•
Index * 16 + 770	CfO_MBusModeData1 bis CfO_MBusModeData8	UINT				•
Index * 16 + 763	CfO_ToutOffData1 bis CfO_ToutOffData8	USINT				•
Index * 8 + 1009	CfO_ReplData1 bis CfO_ReplData8	(U)SINT				•
Index * 8 + 1010	CfO_ReplData1 bis CfO_ReplData8	(U)INT				•
Index * 8 + 1012	CfO_ReplData1 bis CfO_ReplData8	(U)DINT REAL				•
M-Bus - Kommunikation						
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 bis Data8	(U)SINT	•			
Index * 8 + 266	Data1 bis Data8	(U)INT	•			
Index * 8 + 268	Data1 bis Data8	(U)DINT REAL	•			
337	ChangedSNByte	USINT	•			
Index * 8 + 900	SNData1 bis SNData8	UDINT		•		
Flatstream						
2051	InputMTU	USINT				•
2049	OutputMTU	USINT				•
2113	InputSequence	USINT	•			
Index * 2 + 2113	RxByte1 bis RxByte15	USINT	•			
2177	OutputSequence	USINT			•	
Index * 2 + 2177	TxByte1 bis TxByte15	USINT			•	
2053	FlatstreamMode	USINT				•
2055	Forward	USINT				•
2057	ForwardDelay	UINT				•

8.3 Funktionsmodell 254 - Bus Controller

Register	Offset [1]	Name	Datentyp	Lesen		Schreiben	
				Zyklisch	Azyklisch	Zyklisch	Azyklisch
Modulkonfiguration							
774	-	CfO_FunctionModel	UINT				•
M-Bus - Konfiguration							
Index * 16 + 767	-	CfO_LengthData1 bis CfO_LengthData8	USINT				•
Index * 16 + 775	-	CfO_BaudData1 bis CfO_BaudData8	USINT				•
Index * 16 + 761	-	CfO_PAdrData1 bis CfO_PAdrData8	USINT				•
Index * 16 + 765	-	CfO_IndexData1 bis CfO_IndexData8	USINT				•
Index * 16 + 773	-	CfO_ReqTimeData1 bis CfO_ReqTimeData8	USINT				•
Index * 16 + 770	-	CfO_MBusModeData1 bis CfO_MBusMode-Data8	UINT				•
Index * 16 + 763	-	CfO_ToutOffData1 bis CfO_ToutOffData8	USINT				•
Index * 8 + 1009	-	CfO_ReplData1 bis CfO_ReplData8	(U)SINT				•
Index * 8 + 1010	-	CfO_ReplData1 bis CfO_ReplData8	(U)INT				•
Index * 8 + 1012	-	CfO_ReplData1 bis CfO_ReplData8	(U)DINT REAL				•
M-Bus - Kommunikation							
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 bis Data8	(U)SINT	•			
Index * 4 + 6	Index * 4 + 8	Data1 bis Data8	(U)INT	•			
Index * 4 + 8	Index * 4 + 8	Data1 bis Data8	(U)DINT REAL	•			
337	-	ChangedSNByte	USINT		•		
Index * 8 + 900	-	SNData1 bis SNData8	UDINT		•		
Flatstream							
2051	-	InputMTU	USINT				•
2049	-	OutputMTU	USINT				•
0	0	InputSequence	USINT	•			
Index * 1 + 0	Index * 1 + 0	RxByte1 bis RxByte7	USINT	•			
0	0	OutputSequence	USINT			•	
Index * 1 + 0	Index * 1 + 0	TxByte1 bis TxByte7	USINT			•	
2053	-	FlatstreamMode	USINT				•
2055	-	Forward	USINT				•
2057	-	ForwardDelay	UINT				•

1) Der Offset gibt an, wo das Register im CAN-Objekt angeordnet ist.

8.3.1 Verwendung des Moduls am Bus Controller

Das Funktionsmodell 254 "Bus Controller" wird defaultmäßig nur von nicht konfigurierbaren Bus Controllern verwendet. Alle anderen Bus Controller können, abhängig vom verwendeten Feldbus, andere Register und Funktionen verwenden.

Für Detailinformationen siehe X20 Anwenderhandbuch (ab Version 3.50), Abschnitt "Zusätzliche Informationen - Verwendung von I/O-Modulen am Bus Controller".

8.3.2 CAN-I/O Bus Controller

Das Modul belegt an CAN-I/O 3 analoge logische Steckplätze.

8.4 Allgemeines

Beim M-Bus Standard handelt es sich um ein serielles Bussystem, das halbduplex bzw. asynchron kommuniziert. Die hohe Variabilität des Protokolls ermöglicht den Abruf von sehr unterschiedlichen Informationen über dasselbe Interface. In einfachen M-Bus Netzwerken kommuniziert der Master mit bis zu 250 Slaves über die „primäre Adresse“. In späteren Entwicklungsschritten wurde zusätzlich die Sekundäradresse (4 Byte) spezifiziert. Auf diese Weise wurde es möglich die Anzahl der Slaves in einem Netzwerk stark zu steigern.

Wichtige Eckdaten des Moduls

- In der Regel: Verwendung der primären Adresse (1 bis 250)
- Arbeit mit sekundärer Adresse wird nur per Flatsteam unterstützt
- Energieversorgung von 64 Slaves über Bus möglich

8.5 Modulkonfiguration

Die flexible Gestaltung des M-Bus Protokolls führt schnell zu hohem Konfigurationsaufwand. Aus diesem Grund bietet B&R für das Modul zwei unterschiedliche Bedienoberflächen an: "Standard" und "Flatstream". Mit dem benutzerfreundlichen B&R-Standard-Interface können bis zu acht Informationen des M-Bus Netzwerkes zyklisch abgerufen werden. Im Flatstream-Modus arbeitet das Modul als Bridge zwischen SPS und M-Bus Slave. Auf diese Weise stehen alle M-Bus Funktionen zur Verfügung.

Information:

Das Standard-Interface von B&R wird statisch konfiguriert und baut auf zyklischen Registern auf. Da der X2X Link nur eine begrenzte Anzahl an Informationen zyklisch übertragen kann, muss der Anwender eine angepasste Auswahl treffen.

8.5.1 Einstellung der Bedienung

Name:
CfO_FunctionModel

Über dieses Register wird die Standard- bzw. die Flatstreambedienung aktiviert. Auf diese Weise kann die Effizienz des Moduls gesteigert werden.

Die Bits 8 bis 15 werden nur ausgewertet, wenn Bit 0 (B&R Standard-Schnittstelle) aktiviert ist.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	1825

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	B&R Standard-Schnittstelle	0	Deaktiviert
		1	Aktiviert (Bus Controller Default)
1	Flatstream	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
3 - 7	Reserviert	0	
8	Data1	0	Deaktiviert
		1	Aktiviert (Bus Controller Default)
...		...	
10	Data 3	0	Deaktiviert
		1	Aktiviert (Bus Controller Default)
11	Data 4	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
...		...	
15	Data8	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert

8.6 M-Bus - Konfiguration

Für jede einzulesende Information wurden separate Konfigurationsregister aufgelegt. Um einen Zählerwert aus dem M-Bus Netzwerk abzurufen, müssen sie korrekt konfiguriert sein. Grundsätzlich muss der Anwender folgende Informationen seines Slaves kennen:

- Im Slave konfigurierte Übertragungsrate
- Im Slave konfigurierte primäre Adresse (Wert: 1 bis 250, sonst nur Punkt-zu-Punkt-Verbindung möglich)
- Datentyp/Länge der Information
- Strukturierung des Slave-Informationsspeichers

Information:

Der folgende Abschnitt "M-Bus - Konfiguration" bezieht sich ausschließlich auf die B&R Standard-schnittstelle.

8.6.1 Datenlänge

Name:
CfO_LengthData1 bis CfO_LengthData8

Das Standard-Interface ist in der Lage, Daten unterschiedlicher Länge vom M-Bus Slave anzufordern. Bei Verwendung des Automation Studios ergibt sich der Wert des Registers "Length" aus dem eingestellten Datentyp des X2X Link. Es werden alle gängigen Datentypen mit bis zu 4 Byte Länge unterstützt.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	8

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0 - 5	Datalength Code	00 0000	USINT
		00 0001	SINT
		00 0010	UINT
		00 0100	INT
		00 1000	UDINT (Bus Controller Default)
		01 0000	DINT
		10 0000	REAL
6 - 7	Reserviert	0	

8.6.2 Übertragungsrate

Name:
CfO_BaudData1 bis CfO_BaudData8

Über dieses Register wird die Übertragungsrate eingestellt, mit der die gewünschte Information abgerufen werden soll.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	4

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0 - 3	Baudrate (Code)	0000	Reserviert!
		0001	300 Bit/s
		0010	600 Bit/s
		0011	1200 Bit/s
		0100	2400 Bit/s (Bus Controller Default)
		0101	4800 Bit/s
		0110	9600 Bit/s
		0111	19200 Bit/s
		1000	38400 Bit/s
4 - 7	Reserviert	0	

8.6.3 Adresse

Name:
CfO_PAdrData1 bis CfO_PAdrData8

Über dieses Register wird die Adresse eingestellt, von der die gewünschte Information abgerufen werden soll.

Datentyp	Werte	Information
USINT	1 bis 250	Bus Controller Default: 254

Spezielle Adressen:

Wert	Information
251 bis 253	Reserviert (gemäß M-Bus Spezifikation)
254	Broadcastadresse (Antwort aller verbundenen Slaves - Kollisionsgefahr)

8.6.4 Index

Name:
CfO_IndexData1 bis CfO_IndexData8

Über dieses Register wird die Ordnungszahl der Information (unabhängig vom Medium) angegeben. Der Wert ergibt sich aus der Reihenfolge der Informationen im Slave. Im Anschluss wird die Information an das Datenregister übermittelt.

Datentyp	Werte	Information
USINT	1 bis 255	Bus Controller Default: 1

8.6.5 Angabe der Refreshzeit

Name:
CfO_ReqTimeData1 bis CfO_ReqTimeData8

Die Abfrage der Slave-Informationen kann manuell ausgelöst oder zeitbasiert erfolgen. Für die zeitgesteuerte Abfrage muss in diesem der Wert der Refreshzeit angegeben werden. Im Register "M-Bus Modus" auf Seite 10 wird die dazugehörende Einheit festgelegt.

Datentyp	Werte	Information
USINT	1 bis 255	in [s, min]; Bus Controller Default: 0

8.6.6 M-Bus Modus

Name:
CfO_MBusModeData1 bis CfO_MBusModeData8

Um den Bootvorgang des Moduls zu beschleunigen wurden in diesem Register verschiedene Konfigurationsdetails vereint, die das Verhalten des Moduls definieren.

Datentyp	Werte	Bus Controller Default
UINT	Siehe Bitstruktur	2

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0 - 2	Byte Offset	0 - 7	Datentypen; Bus Controller Default: 2
3 - 4	Reserviert	0	
5	InitFrame	0	Kein Zusatzframe (Bus Controller Default)
		1	Zusatzframe senden
6	ApplicationResetFrame	0	Kein Zusatzframe (Bus Controller Default)
		1	Zusatzframe senden
7	Ersatzwertstrategie	0	Letzten gültigen Wert halten (Bus Controller Default)
		1	Durch statischen Wert ersetzen
8	Zeitbasiertes Auslesen	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
9	Manuell ausgelöstes Lesen	0	Deaktiviert (Bus Controller Default)
		1	Auslesen über das Register "MBusCommand" auf Seite 12
10	Unit of periodic read	0	[s] - Sekunde (Bus Controller Default)
		1	[min] - Minute
11 - 15	Reserviert	0	

Byte Offset

Die M-Bus Spezifikation lässt viele individuelle Datentypen zu. Um auch diese Counterwerte mit bis zu 64 Bit einlesen zu können, muss eine Slave-Information gegebenenfalls auf 2 Datenregister aufgeteilt eingelesen werden. Um den gewünschten Abschnitt der Information anzuwählen, kann der Byte Offset definiert werden.

8.6.7 Timeout-Offset

Name:
CfO_ToutOffData1 bis CfO_ToutOffData8

Die Timeout-Zeit für die M-Bus Kommunikation hängt grundsätzlich von der momentan eingestellten Übertragungsrate ab. Der Anwender hat die Möglichkeit, zusätzlich zum errechneten Standard-Timeout einen zusätzlichen Offset-Wert zu definieren.

Es gilt: Timeout = Standard-Timeout + (Timeout-Offset * 10 ms)

Datentyp	Werte	Information
USINT	0 bis 255	Auflösung 10 ms; Bus Controller Default: 0

8.6.8 Statischer Ersatzwert

Name:
CfO_ReplData1 bis CfO_ReplData8

Über dieses Register wird der statische Ersatzwert definiert, wenn im Register "CfO_MBusModeData" auf Seite 10 die Ersatzwertstrategie "Durch statischen Wert ersetzen" aktiviert wurde. Das Datenregister nimmt diesen Wert an, falls ein ungültiger Eingangswert erkannt wird.

Datentyp	Werte	Information
(U)SINT (U)INT (U)DINT REAL	Gemäß Datentyp	Bus Controller Default: 0

8.7 M-Bus - Kommunikation

Zur Kommunikation mit den M-Bus Slaves werden beim B&R-Interface drei wichtige Steuer- bzw. Statusbytes aufgelegt. Über das Register "MBusCommand" auf Seite 12 kann z. B. der UART ein-/ausgeschaltet werden, um die Energieeffizienz des Systems zu steigern.

Je nach Konfiguration sind bis zu 8 zyklische Eingangsregister angemeldet. Manuell konfigurierte Daten müssen über das Register "MBusCommand" auf Seite 12 angefordert werden. Über die Register "ValidDataByte" auf Seite 13 und "InvalidDataByte" auf Seite 13 wird eine Aussage über die Qualität der aktuell eingelesenen Information möglich.

> **Information:**
>
> **Der folgende Abschnitt "M-Bus - Kommunikation" bezieht sich ausschließlich auf die B&R Standard-schnittstelle.**

8.7.1 M-Bus Anweisungen

Name:
MBusCommand

Mit diesem Register können dem Modul verschiedene Anweisungen gegeben werden. Dabei reagiert das Modul ausschließlich auf positive Flanken.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	UART einschalten	0 → 1	Anweisung ausführen
1	Lesen manuell getriggerter Informationen	0 → 1	Anweisung ausführen
2	Quittieren des Registers "MBusState"	0 → 1	Anweisung ausführen
3 - 6	Reserviert	0	
7	UART ausschalten	0 → 1	Anweisung ausführen

Bit 0 und 7

Der Pegelwandler ist nach dem Hochfahren des Moduls defaultmäßig eingeschaltet. Mit Hilfe dieser Bits kann er applikationsmäßig ein- und ausgeschaltet werden, um z. B. Strom zu sparen.

8.7.2 M-Bus Betrieb

Name:
MBusOperation

Dieses Register zeigt dem Anwender, welche Aufgabe das Modul gerade abarbeitet. Dabei ist das LSB immer gesetzt, solange der UART aktiv ist. Manuelle Anweisungen werden zum Zeitpunkt der Ausführung in diesem Byte um eins erhöht gespiegelt.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	UART	0	Nicht aktiv
		1	Aktiv
1	Informationen lesen	0	-
		1	Anweisung wird ausgeführt
2	Refresh/reset des Registers "MBusState"	0	-
		1	Anweisung wird ausgeführt[1)]
3 - 6	Reserviert	0	
7	UART	0	Nicht aktiv
		1	Aktiv

1) Bit 2 wird nur für einen X2X Zyklus gesetzt. Für den Betrieb des Moduls hinter einem Bus Controller wird die Abfrage dieses Bits nicht empfohlen.

8.7.3 M-Bus Status

Name:
MBusState

Dieses Register enthält den aktuellen M-Bus Netzwerkfehlerstatus. Alle Bits arbeiten remanent. Das heißt, sie müssen über das Register "MBusCommand" auf Seite 12 zurückgesetzt werden.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	Kollisionserkennung	0	Adressierung fehlerfrei
		1	Adresse mehrfach am Bus
1	Lesefehler (mindestes einmal)	0	Konfigurierte Informationen okay
		1	Information konnte nicht gelesen werden
2	Checksumme	0	Empfangene Checksumme okay
		1	Fehler in Eingangsrichtung
3	M-Bus Last	0	Energieversorgung okay
		1	Last des M-Bus Netzwerkes zu hoch
4	Kommunikationabbruch wegen Überlauf	0	Alles OK
		1	Master ist ausgelastet und kann keine weiteren Anfragen annehmen Maßnahme: Anfrage wiederholen[1)]
5	Kommunikationsabbruch wegen Pegelwandler	0	Alles OK
		1	Pegelwandler ist AUS (zur Laufzeit abgebrochen bzw. nicht gestartet)
6	Datenaustausch seit Hochlauf	0	Bisher keine gültigen Daten empfangen
		1	Mindestens einmal gültige Daten
7	UART off MBUS ENABLE	0	M-Bus Treiber, Pegelwandler inaktiv
		1	Modul bereit zur Kommunikation

1) Kommunikation wird automatisch wieder aufgenommen, sobald die ausstehenden Kommunikationsjobs abgearbeitet wurden.

8.7.4 Gültige Daten

Name:
ValidDataByte

ValidData1 bis ValidData8

Dieses Register zeigt bitweise welche der max. 8 eingelesenen Informationen gültig sind.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	ValidData1	0	Information 1 ungültig
		1	Information 1 gültig
…	…	…	…
7	ValidData8	0	Information 8 ungültig
		1	Information 8 gültig

8.7.5 Ungültige Daten

Name:
InvalidDataByte

InvalidData1 bis InvalidData8

Die Gültigkeit der eingelesenen Informationen kann redundant geprüft werden. Mit diesem Register wird bitweise angezeigt, welche der max. 8 eingelesenen Informationen ungültig sind.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	InvalidData1	0	Information 1 gültig
		1	Information 1 ungültig
…	…	…	…
7	InvalidData8	0	Information 8 gültig
		1	Information 8 ungültig

8.7.6 Daten

Name:
Data1 bis Data8

Die zyklischen Datenregister enthalten die jeweils vorkonfigurierte Information aus dem M-Bus Netzwerk. Der Datentyp der Datenregister wurde variabel gestaltet und muss vom Anwender bei der Konfiguration festgelegt werden.

Information:

Da der X2X Link nur eine begrenzte Anzahl an Bytes zyklisch übertragen kann, muss der Anwender eine angepasste Auswahl treffen.

Datentyp	Werte
(U)SINT (U)INT (U)DINT REAL	Gemäß Datentyp

8.7.7 Änderung der Seriennummer eines M-Bus Slaves

Name:
ChangedSNByte

Dieses Register gibt bitweise Auskunft darüber, ob sich eine der M-Bus Slave Seriennummern am Bus geändert hat. Erfasst werden dabei nur die Seriennummern der Slaves, die über die B&R Schnittstelle abgerufen werden. Wird eine Änderung detektiert, alterniert das jeweilige Bit.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	SN (Slave 1)	0 -> 1	Slave1: Änderung der Seriennummer
		1 -> 0	
…	…	…	…
7	SN (Slave 8)	0 -> 1	Slave8: Änderung der Seriennummer
		1 -> 0	

8.7.8 Seriennummern der M-Bus Slaves

Name:
SNData1 bis SNData8

Diese Register enthalten die Seriennummern der M-Bus Slaves, die über die B&R Schnittstelle abgefragt werden. Sie sind azyklisch implementiert und können per AsIOAcc-Bibliothek eingelesen werden.

Datentyp	Werte
UDINT	0 bis 4.294.967.295

8.8 Die Flatstream-Kommunikation

8.8.1 Einleitung

Für einige Module stellt B&R ein zusätzliches Kommunikationsverfahren bereit. Der "Flatstream" wurde für X2X und POWERLINK Netzwerke konzipiert und ermöglicht einen individuell angepassten Datentransfer. Obwohl das Verfahren nicht unmittelbar echtzeitfähig ist, kann die Übertragung effizienter gestaltet werden als bei der zyklischen Standardabfrage.

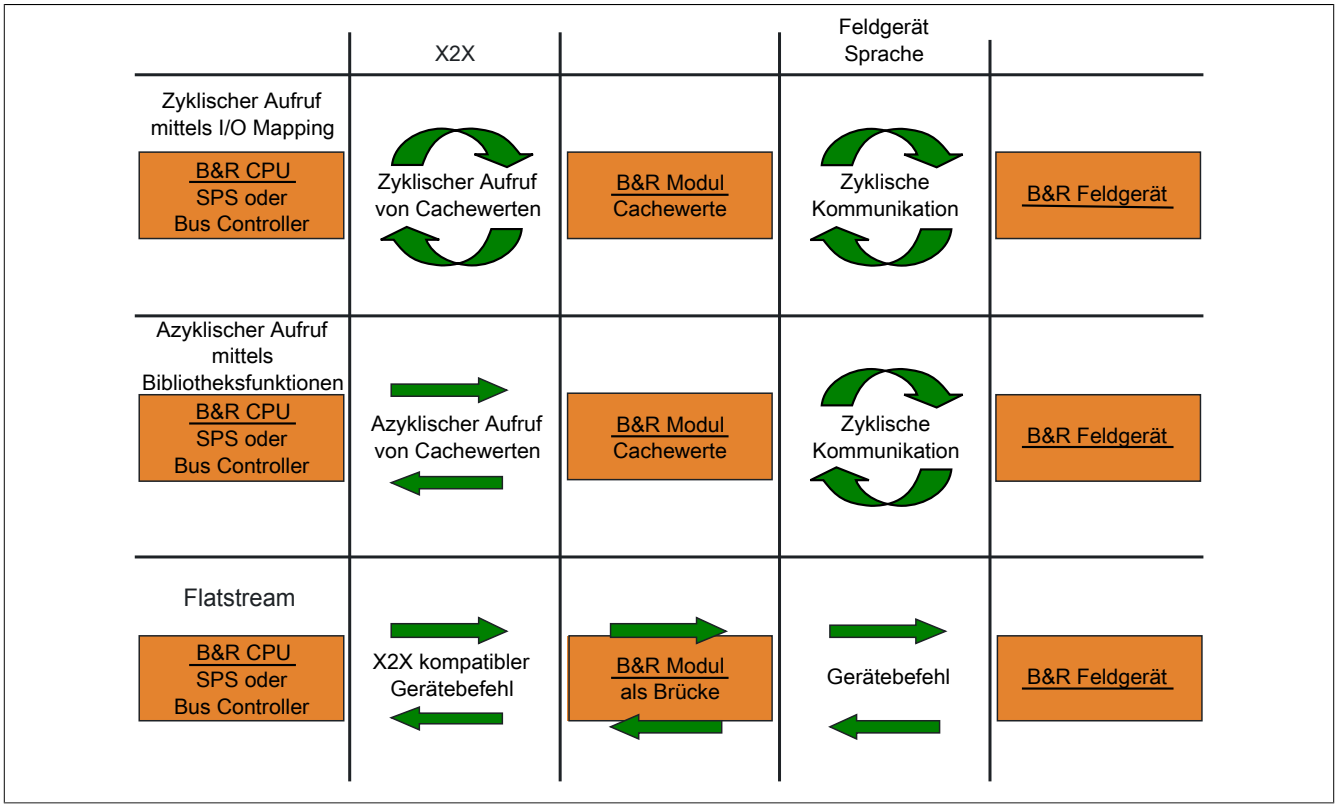


Abbildung 1: 3 Arten der Kommunikation

Durch den Flatstream wird die zyklische bzw. azyklische Abfrage ergänzt. Bei der Flatstream-Kommunikation fungiert das Modul als Bridge. Die Anfragen der CPU werden über das Modul direkt zum Feldgerät geleitet.

8.8.2 Nachricht, Segment, Sequenz, MTU

Die physikalischen Eigenschaften des Bussystems begrenzen die Datenmenge, die während eines Buszyklus übermittelt werden kann. Bei der Flatstream-Kommunikation werden alle Nachrichten als fortlaufender Datenstrom (engl. stream) betrachtet. Lange Datenströme müssen in mehrere Teile zerlegt und nacheinander versendet werden. Um zu verstehen wie der Empfänger die ursprüngliche Information wieder zusammensetzt, werden die Begriffe Nachricht, Segment, Sequenz und MTU unterschieden.

Nachricht

Eine Nachricht ist eine Mitteilung, die zwischen 2 Kommunikationspartnern ausgetauscht werden soll. Die Länge einer solchen Mitteilung wird durch das Flatstream-Verfahren nicht begrenzt. Es müssen allerdings modulspezifische Beschränkungen beachtet werden.

Segment (logische Gliederung einer Nachricht)

Ein Segment ist endlich groß und kann als Abschnitt der Nachricht verstanden werden. Die Anzahl der Segmente pro Nachricht ist beliebig. Damit der Empfänger die übertragenen Segmente wieder korrekt zusammensetzen kann, geht jedem Segment ein Byte mit Zusatzinformationen voraus. Das sogenannte Controlbyte enthält z. B. Informationen über die Länge eines Segments und ob das kommende Segment die Mitteilung vervollständigt. Auf diesem Weg wird der Empfänger in die Lage versetzt, den ankommenden Datenstrom korrekt zu interpretieren.

Sequenz (physikalisch notwendige Gliederung eines Segments)

Die maximale Größe einer Sequenz entspricht der Anzahl der aktivierten Rx- bzw. Tx-Bytes (später: "MTU"). Die sendende Station teilt das Sendearray in zulässige Sequenzen, die nacheinander in die MTU geschrieben, zum Empfänger übertragen und dort wieder aneinandergereiht werden. Der Empfänger legt die ankommenden Sequenzen in einem Empfangsarray ab und erhält somit ein Abbild des Datenstroms.
Bei der Flatstream-Kommunikation werden die abgesetzten Sequenzen gezählt. Erfolgreich übertragene Sequenzen müssen vom Empfänger bestätigt werden, um die Übertragung abzusichern.

MTU (Maximum Transmission Unit) - Physikalischer Transport

Die MTU des Flatstreams beschreibt die aktivierten USINT-Register für den Flatstream. Die Register können mindestens eine Sequenz aufnehmen und zum Empfänger übertragen. Für beide Kommunikationsrichtungen wird eine separate MTU vereinbart. Die OutputMTU definiert die Anzahl der Flatstream-Tx-Bytes und die InputMTU beschreibt die Anzahl der Flatstream-Rx-Bytes. Die MTUs werden zyklisch über den X2X Link transportiert, sodass die Auslastung mit jedem zusätzlich aktivierten USINT-Register steigt.

Eigenschaften

Flatstream-Nachrichten werden nicht zyklisch und nicht unmittelbar in Echtzeit übertragen. Zur Übertragung einer bestimmten Mitteilung werden individuell viele Buszyklen benötigt. Die Rx-/Tx-Register werden zwar zyklisch zwischen Sender und Empfänger ausgetauscht, aber erst weiterverarbeitet, wenn die Übernahme durch die Register "InputSequence" bzw. "OutputSequence" explizit angewiesen wird.

Verhalten im Fehlerfall (Kurzfassung)

Das Protokoll von X2X bzw. POWERLINK Netzwerken sieht vor, dass bei einer Störung die letzten gültigen Werte erhalten bleiben. Bei der herkömmlichen Kommunikation (zyklische/azyklische Abfrage) kann ein solcher Fehler in der Regel ignoriert werden.
Damit auch via Flatstream problemlos kommuniziert werden kann, müssen alle abgesetzten Sequenzen vom Empfänger bestätigt werden. Ohne die Nutzung des Forward verzögert sich die weitere Kommunikation um die Dauer der Störung.
Falls der Forward genutzt wird, erhält die Empfängerstation einen doppelt inkrementierten Sendezähler. Der Empfänger stoppt, das heißt, er schickt keine Bestätigungen mehr zurück. Anhand des SequenceAck erkennt die Sendestation, dass die Übertragung fehlerhaft war und alle betroffenen Sequenzen wiederholt werden müssen.

8.8.3 Prinzip des Flatstreams

Voraussetzung

Bevor der Flatstream genutzt werden kann, muss die jeweilige Kommunikationsrichtung synchronisiert sein, das heißt, beide Kommunikationspartner fragen zyklisch den SequenceCounter der Gegenstelle ab. Damit prüfen sie, ob neue Daten vorliegen, die übernommen werden müssen.

Kommunikation

Wenn ein Kommunikationspartner eine Nachricht an seine Gegenstelle senden will, sollte er zunächst ein Sendearray anlegen, das den Konventionen des Flatstreams entspricht. Auf diese Weise kann der Flatstream sehr effizient gestaltet werden, ohne wichtige Ressourcen zu blockieren.

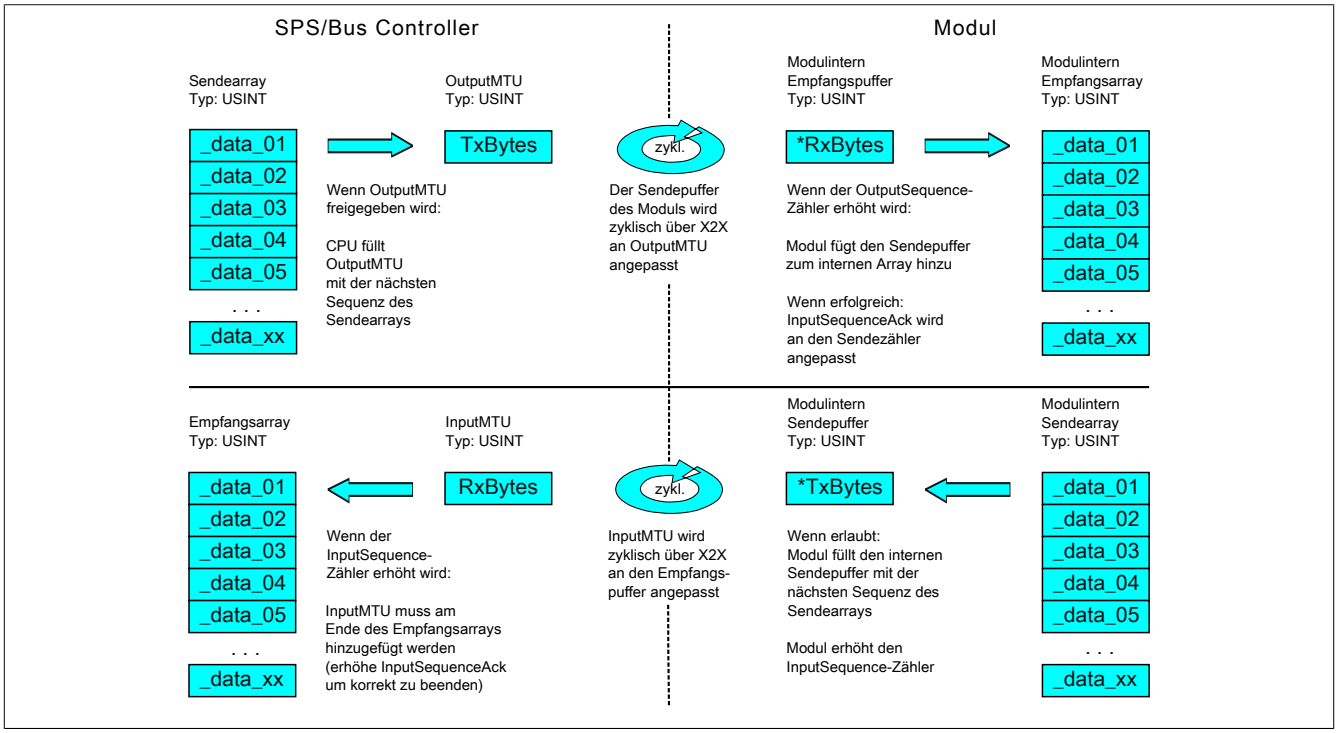


Abbildung 2: Kommunikation per Flatstream

Vorgehensweise

Als erstes wird die Nachricht in zulässige Segmente mit max. 63 Bytes aufgeteilt und die entsprechenden Controlbytes gebildet. Die Daten werden zu einem Datenstrom zusammengefügt, das heißt, je ein Controlbyte und das dazugehörige Segment im Wechsel. Dieser Datenstrom kann in das Sendearray geschrieben werden. Jedes Arrayelement ist dabei max. so groß, wie die freigegebene MTU, sodass ein Element einer Sequenz entspricht.
Wenn das Array vollständig angelegt ist, prüft der Sender, ob die MTU neu befüllt werden darf. Danach kopiert er das erste Element des Arrays bzw. die erste Sequenz auf die Tx-Byte-Register. Die MTU wird zyklisch über den X2X Link zur Empfängerstation transportiert und auf den korrespondierenden Rx-Byte-Registern abgelegt. Als Signal, dass die Daten vom Empfänger übernommen werden sollen, erhöht der Sender seinen SequenceCounter. Wenn die Kommunikationsrichtung synchronisiert ist, erkennt die Gegenstelle den inkrementierten Sequence-Counter. Die aktuelle Sequenz wird an das Empfangsarray angefügt und per SequenceAck bestätigt. Mit dieser Bestätigung wird dem Sender signalisiert, dass die MTU wieder neu befüllt werden kann.

Bei erfolgreicher Übertragung entsprechen die Daten im Empfangsarray exakt denen im Sendearray. Während der Übertragung muss die Empfangsstation die ankommenden Controlbytes erkennen und auswerten. Für jede Nachricht sollte ein separates Empfangsarray angelegt werden. Auf diese Weise kann der Empfänger vollständig übertragene Nachrichten sofort weiterverarbeiten.

8.8.4 Die Register für den Flatstream-Modus

Zur Konfiguration des Flatstreams sind 5 Register vorgesehen. Mit der Standardkonfiguration können geringe Datenmengen relativ einfach übermittelt werden.

Information:

Die CPU kommuniziert über die Register "OutputSequence" und "InputSequence" sowie den aktivierten Tx- bzw. RxBytes direkt mit dem Feldgerät. Deshalb benötigt der Anwender ausreichend Kenntnisse über das Kommunikationsprotokoll des Feldgerätes.

8.8.4.1 Konfiguration des Flatstreams

Um den Flatstream zu nutzen, muss der Programmablauf erweitert werden. Die Zykluszeit der Flatstream-Routinen muss auf ein Vielfaches des Buszyklus festgelegt werden. Die zusätzlichen Programmroutinen sollten in Cyclic #1 implementiert werden, um die Datenkonsistenz zu gewährleisten.

Bei der Minimalkonfiguration müssen die Register "InputMTU" und "OutputMTU" eingestellt werden. Alle anderen Register werden beim Start mit Standardwerten belegt und können sofort genutzt werden. Sie stellen zusätzliche Optionen bereit, um Daten kompakter zu übertragen bzw. den allgemeinen Ablauf hoch effizient zu gestalten.

Mit den Forward-Registern wird der Ablauf des Flatstream-Protokolls erweitert. Diese Funktion eignet sich, um die Datenrate des Flatstreams stark zu erhöhen, bedeutet aber erheblichen Mehraufwand bei der Erstellung des Programmablaufs.

8.8.4.1.1 Anzahl der aktivierten Tx- bzw. Rx-Bytes

Name:
OutputMTU
InputMTU

Diese Register definieren die Anzahl der aktivierten Tx- bzw. Rx-Bytes und somit auch die maximale Größe einer Sequenz. Der Anwender muss beachten, dass mehr freigegebene Bytes auch eine stärkere Belastung für das Bussystem bedeuten.

Information:

In der weiteren Beschreibung stehen die Bezeichnungen "OutputMTU" und "InputMTU" nicht für die hier erläuterten Register, sondern als Synonym für die momentan aktivierten Tx- bzw. Rx-Bytes.

Datentyp	Werte
USINT	Siehe modulspezifische Registerübersicht (theoretisch: 3 bis 27)

8.8.4.2 Bedienung des Flatstreams

Bei der Verwendung des Flatstreams ist die Kommunikationsrichtung von großer Bedeutung. Für das Senden von Daten an ein Modul (Output-Richtung) werden die Tx-Bytes genutzt. Für den Empfang von Daten eines Moduls (Input-Richtung) sind die Rx-Bytes vorgesehen.
Mit den Registern "OutputSequence" und "InputSequence" wird die Kommunikation gesteuert bzw. abgesichert, das heißt, der Sender gibt damit die Anweisung, Daten zu übernehmen und der Empfänger bestätigt eine erfolgreich übertragene Sequenz.

8.8.4.2.1 Format der Ein- und Ausgangsbytes

Name:

"Format des Flatstream" im Automation Studio

Bei einigen Modulen kann mit Hilfe dieser Funktion eingestellt werden, wie die Ein- und Ausgangsbytes des Flatstream (Tx- bzw. Rx-Bytes) übergeben werden.

- **gepackt:** Daten werden als ein Array übergeben
- **byteweise:** Daten werden als einzelne Bytes übergeben

8.8.4.2.2 Transport der Nutzdaten und der Controlbytes

Name:
TxByte1 bis TxByteN
RxByte1 bis RxByteN

(Die Größe der Zahl N ist je nach verwendetem Bus Controller Modell unerschiedlich.)

Die Tx- bzw. Rx-Bytes sind zyklische Register, die zum Transport der Nutzdaten und der notwendigen Controlbytes dienen. Die Anzahl aktiver Tx- bzw. Rx-Bytes ergibt sich aus der Konfiguration der Register "OutputMTU" bzw. "InputMTU".
Im Programmablauf des Anwenders können nur die Tx- bzw. Rx-Bytes der CPU genutzt werden. Innerhalb des Moduls gibt es die entsprechenden Gegenstücke, welche für den Anwender nicht zugänglich sind. Aus diesem Grund wurden die Bezeichnungen aus Sicht der CPU gewählt.

- "T" - "transmit" → CPU *sendet* Daten an das Modul
- "R" - "receive" → CPU *empfängt* Daten vom Modul

Datentyp	Werte
USINT	0 bis 255

8.8.4.2.3 Controlbytes

Neben den Nutzdaten übertragen die Tx- bzw. Rx-Bytes auch die sogenannten Controlbytes. Sie enthalten zusätzliche Informationen über den Datenstrom, damit der Empfänger die übertragenen Segmente wieder korrekt zur ursprünglichen Nachricht zusammensetzen kann.

Bitstruktur eines Controlbytes

Bit	Bezeichnung	Wert	Information
0 - 5	SegmentLength	0 - 63	Bytegröße des folgenden Segments (Standard: max. MTU-Größe - 1)
6	nextCBPos	0	Nächstes Controlbyte zu Beginn der nächsten MTU
		1	Nächstes Controlbyte direkt nach Ende des Segments
7	MessageEndBit	0	Nachricht wird nach dem folgenden Segment fortgesetzt
		1	Nachricht wird durch das folgende Segment beendet

SegmentLength

Die Segmentlänge kündigt dem Empfänger an, wie lang das kommende Segment ist. Wenn die eingestellte Segmentlänge für eine Nachricht nicht ausreicht, muss die Mitteilung auf mehrere Segmente verteilt werden. In diesen Fällen kann das tatsächliche Ende der Nachricht über Bit 7 (Controlbyte) erkannt werden.

Information:

Bei der Bestimmung der Segmentlänge wird das Controlbyte nicht mitgerechnet. Die Segmentlänge ergibt sich rein aus den Bytes der Nutzdaten.

nextCBPos

Mit diesem Bit wird angezeigt, an welcher Position das nächste Controlbyte zu erwarten ist. Diese Information ist vor allem bei Anwendung der Option "MultiSegmentMTU" wichtig.
Bei der Flatstream-Kommunikation mit MultiSegmentMTUs ist das nächste Controlbyte nicht mehr auf dem ersten Rx-Byte der darauffolgenden MTU zu erwarten, sondern wird direkt im Anschluss an das Segment übertragen.

MessageEndBit

Das "MessageEndBit" wird gesetzt, wenn das folgende Segment eine Nachricht abschließt. Die Mitteilung ist vollständig übertragen und kann weiterverarbeitet werden.

Information:

In Output-Richtung muss dieses Bit auch dann gesetzt werden, wenn ein einzelnes Segment ausreicht, um die vollständige Nachricht aufzunehmen. Das Modul verarbeitet eine Mitteilung intern nur, wenn diese Kennzeichnung vorgenommen wurde.
Die Größe einer übertragenen Mitteilung lässt sich berechnen, wenn alle Segmentlängen der Nachricht addiert werden.

Flatstream-Formel zur Berechnung der Nachrichtenlänge:

Nachricht [Byte] = Segmentlängen (aller CBs ohne ME) + Segmentlänge (des ersten CB mit ME)	CB	Controlbyte
	ME	MessageEndBit

8.8.4.2.4 Kommunikationsstatus der CPU

Name:
OutputSequence

Das Register "OutputSequence" enthält Informationen über den Kommunikationsstatus der CPU. Es wird von der CPU geschrieben und vom Modul gelesen.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0 - 2	OutputSequenceCounter	0 - 7	Zähler der in Output abgesetzten Sequenzen
3	OutputSyncBit	0	Output-Richtung deaktiviert (disable)
		1	Output-Richtung aktiviert (enable)
4 - 6	InputSequenceAck	0 - 7	Spiegel des InputSequenceCounters
7	InputSyncAck	0	Input-Richtung nicht bereit (disable)
		1	Input-Richtung bereit (enable)

OutputSequenceCounter

Der OutputSequenceCounter ist ein umlaufender Zähler der Sequenzen, die von der CPU abgeschickt wurden. Über den OutputSequenceCounter weist die CPU das Modul an, eine Sequenz zu übernehmen (zu diesem Zeitpunkt muss die Output-Richtung synchronisiert sein).

OutputSyncBit

Mit dem OutputSyncBit versucht die CPU den Output-Kanal zu synchronisieren.

InputSequenceAck

Der InputSequenceAck dient zur Bestätigung. Der Wert des InputSequenceCounters wird darin gespiegelt, wenn die CPU eine Sequenz erfolgreich empfangen hat.

InputSyncAck

Das Bit InputSyncAck bestätigt dem Modul die Synchronität des Input-Kanals. Die CPU zeigt damit an, dass sie bereit ist, Daten zu empfangen.

8.8.4.2.5 Kommunikationsstatus des Moduls

Name:
InputSequence

Das Register "InputSequence" enthält Informationen über den Kommunikationsstatus des Moduls. Es wird vom Modul geschrieben und sollte von der CPU nur gelesen werden.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0 - 2	InputSequenceCounter	0 - 7	Zähler der in Input abgesetzten Sequenzen
3	InputSyncBit	0	Nicht bereit (disable)
		1	Bereit (enable)
4 - 6	OutputSequenceAck	0 - 7	Spiegel des OutputSequenceCounters
7	OutputSyncAck	0	Nicht bereit (disable)
		1	Bereit (enable)

InputSequenceCounter

Der InputSequenceCounter ist ein umlaufender Zähler der Sequenzen, die vom Modul abgeschickt wurden. Über den InputSequenceCounter weist das Modul die CPU an, eine Sequenz zu übernehmen (zu diesem Zeitpunkt muss die Input-Richtung synchronisiert sein).

InputSyncBit

Mit dem InputSyncBit versucht das Modul den Input-Kanal zu synchronisieren.

OutputSequenceAck

Der OutputSequenceAck dient zur Bestätigung. Der Wert des OutputSequenceCounters wird darin gespiegelt, wenn das Modul eine Sequenz erfolgreich empfangen hat.

OutputSyncAck

Das Bit OutputSyncAck bestätigt der CPU die Synchronität des Output-Kanals. Das Modul zeigt damit an, dass es bereit ist, Daten zu empfangen.

8.8.4.2.6 Beziehung zwischen Output- und InputSequence

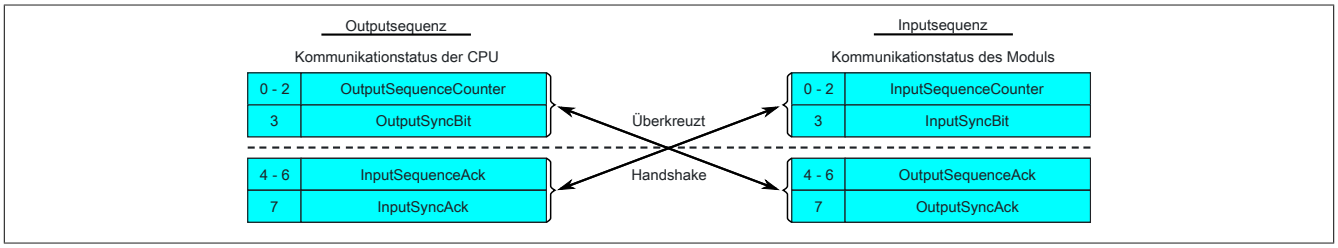


Abbildung 3: Zusammenhang zwischen Output- und InputSequence

Die Register "OutputSequence" und "InputSequence" sind logisch aus 2 Halb-Bytes aufgebaut. Über den Low-Teil wird der Gegenstelle signalisiert, ob ein Kanal geöffnet werden soll bzw. ob Daten übernommen werden können. Der High-Teil dient zur Bestätigung, wenn die geforderte Aktion erfolgreich ausgeführt wurde.

SyncBit und SyncAck

Wenn das SyncBit und das SyncAck einer Kommunikationsrichtung gesetzt sind, gilt der Kanal als "synchronisiert", das heißt, es können Nachrichten in diese Richtung versendet werden. Das Statusbit der Gegenstelle muss zyklisch überprüft werden. Falls das SyncAck zurückgesetzt wurde, muss das eigene SyncBit angepasst werden. Bevor neue Daten übertragen werden können, muss der Kanal resynchronisiert werden.

SequenceCounter und SequenceAck

Die Kommunikationspartner prüfen zyklisch, ob sich das Low-Nibble der Gegenstelle ändert. Wenn ein Kommunikationspartner eine neue Sequenz vollständig auf die MTU geschrieben hat, erhöht er seinen SequenceCounter. Daraufhin übernimmt der Empfänger die aktuelle Sequenz und bestätigt den erfolgreichen Empfang per SequenceAck. Auf diese Weise wird ein Handshake-Verfahren initiiert.

Information:

Bei einer Unterbrechung der Kommunikation werden Segmente von unvollständig übermittelten Mitteilungen verworfen. Alle fertig übertragenen Nachrichten werden bearbeitet.

8.8.4.3 Synchronisieren

Beim Synchronisieren wird ein Kommunikationskanal geöffnet. Es muss sichergestellt sein, dass ein Modul vorhanden und der aktuelle Wert des SequenceCounters beim Empfänger der Nachricht hinterlegt ist.
Der Flatstream bietet die Möglichkeit Vollduplex zu kommunizieren. Beide Kanäle/Kommunikationsrichtungen können separat betrachtet werden. Sie müssen unabhängig voneinander synchronisiert werden, sodass theoretisch auch simplex kommuniziert werden könnte.

Synchronisation der Output-Richtung (CPU als Sender)

Die korrespondierenden Synchronisationsbits (OutputSyncBit und OutputSyncAck) sind zurückgesetzt. Aus diesem Grund können momentan keine Nachrichten von der CPU an das Modul per Flatstream übertragen werden.

Algorithmus

1) CPU muss 000 in OutputSequenceCounter schreiben und OutputSyncBit zurücksetzen. CPU muss High-Nibble des Registers "InputSequence" zyklisch abfragen (Prüfung ob 000 in OutputSequenceAck und 0 in OutputSyncAck).
Modul übernimmt den aktuellen Inhalt der InputMTU nicht, weil der Kanal noch nicht synchronisiert ist. *Modul gleicht OutputSequenceAck und OutputSyncAck an die Werte des OutputSequenceCounters bzw. des OutputSyncBits an.*
2) Wenn die CPU die erwarteten Werte in OutputSequenceAck und OutputSyncAck registriert, darf sie den OutputSequenceCounter inkrementieren. Die CPU fragt das High-Nibble des Registers "OutputSequence" weiter zyklisch ab (Prüfung ob 001 in OutputSequenceAck und 0 in InputSyncAck).
Modul übernimmt den aktuellen Inhalt der InputMTU nicht, weil der Kanal noch nicht synchronisiert ist. *Modul gleicht OutputSequenceAck und OutputSyncAck an die Werte des OutputSequenceCounters bzw. des OutputSyncBits an.*
3) Wenn die CPU die erwarteten Werte in OutputSequenceAck und OutputSyncAck registriert, darf sie das OutputSyncBit setzen. Die CPU fragt das High-Nibble des Registers "OutputSequence" weiter zyklisch ab (Prüfung ob 001 in OutputSequenceAck und 1 in InputSyncAck). **Hinweis:** Theoretisch könnten ab diesem Moment Daten übertragen werden. Es wird allerdings empfohlen, erst dann Daten zu übertragen, wenn die Output-Richtung vollständig synchronisiert ist.
Modul setzt OutputSyncAck.
Output-Richtung synchronisiert, CPU kann Daten an Modul senden.

Synchronisation der Input-Richtung (CPU als Empfänger)

Die korrespondierenden Synchronisationsbits (InputSyncBit und InputSyncAck) sind zurückgesetzt. Aus diesem Grund können momentan keine Nachrichten vom Modul an die CPU per Flatstream übertragen werden.

Algorithmus

Modul schreibt 000 in InputSequenceCounter und setzt InputSyncBit zurück. *Modul überwacht High-Nibble des Registers "OutputSequence" - erwartet 000 in InputSequenceAck bzw. 0 in InputSyncAck.*
1) CPU darf den aktuellen Inhalt der InputMTU nicht übernehmen, weil der Kanal noch nicht synchronisiert ist. CPU muss InputSequenceAck und InputSyncAck an die Werte des InputSequenceCounters bzw. des InputSyncBits angleichen.
Wenn das Modul die erwarteten Werte in InputSequenceAck und in InputSyncAck registriert, inkrementiert es den InputSequenceCounter. *Modul überwacht High-Nibble des Registers "OutputSequence" - erwartet 001 in InputSequenceAck bzw. 0 in InputSyncAck.*
2) CPU darf den aktuellen Inhalt der InputMTU nicht übernehmen, weil der Kanal noch nicht synchronisiert ist. CPU muss InputSequenceAck und InputSyncAck an die Werte des InputSequenceCounters bzw. des InputSyncBits angleichen.
Wenn das Modul die erwarteten Werte in InputSequenceAck und in InputSyncAck registriert, setzt es das InputSyncBit. *Modul überwacht High-Nibble des Registers "OutputSequence" - erwartet 1 in InputSyncAck.*
3) CPU darf InputSyncAck setzen. **Hinweis:** Theoretisch könnten bereits in diesem Zyklus Daten übertragen werden. Es gilt: Wenn das InputSyncBit gesetzt ist und der InputSequenceCounter um 1 erhöht wurde, müssen die Informationen der aktivierten Rx-Bytes übernommen und bestätigt werden (siehe dazu auch Kommunikation in Input-Richtung).
Input-Richtung synchronisiert, Modul kann Daten an CPU senden.

8.8.4.4 Senden und Empfangen

Wenn ein Kanal synchronisiert ist, gilt die Gegenstelle als empfangsbereit und der Sender kann Nachrichten verschicken. Bevor der Sender Daten absetzen kann, legt er das sogenannte Sendearray an, um den Anforderungen des Flatstreams gerecht zu werden.

Die sendende Station muss für jedes erstellte Segment ein individuelles Controlbyte generieren. Ein solches Controlbyte enthält Informationen, wie der nächste Teil der übertragenen Daten zu verarbeiten ist. Die Position des nächsten Controlbytes im Datenstrom kann variieren. Aus diesem Grund muss zu jedem Zeitpunkt eindeutig definiert sein, wann ein neues Controlbyte übermittelt wird. Das erste Controlbyte befindet sich immer auf dem ersten Byte der ersten Sequenz. Alle weiteren Positionen werden rekursiv mitgeteilt.

Flatstream-Formel zur Berechnung der Position des nächsten Controlbytes:

Position (nächstes Controlbyte) = aktuelle Position + 1 + Segmentlänge

Beispiel

Es werden 3 unabhängige Nachrichten (7 Bytes, 2 Bytes, 9 Bytes) über eine 7-Byte breite MTU übermittelt. Die sonstige Konfiguration entspricht den Standardeinstellungen.

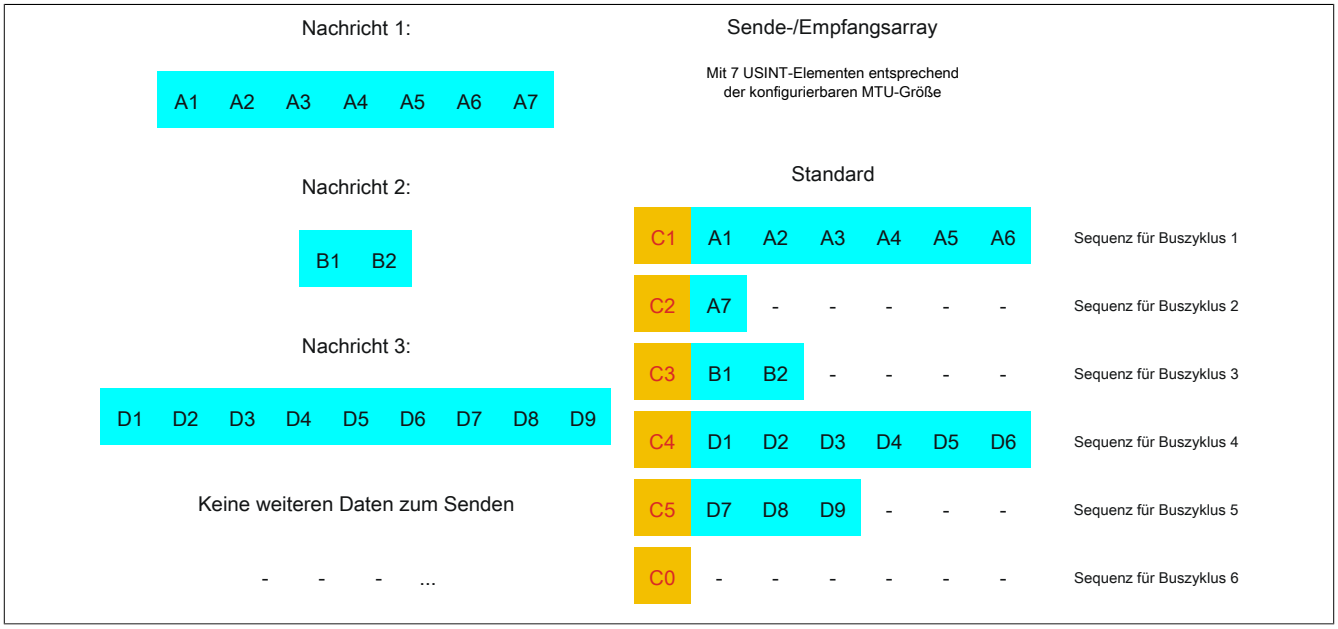


Abbildung 4: Sende-/Empfangsarray (Standard)

Zunächst müssen die Nachrichten in Segmente geteilt werden. Bei der Standardkonfiguration muss sichergestellt sein, dass jede Sequenz ein gesamtes Segment inklusive dem dazugehörigen Controlbyte aufnehmen kann. Die Sequenz ist auf die Größe der aktivierten MTU begrenzt, das heißt, ein Segment muss mindestens um 1 Byte kleiner sein als die aktivierte MTU.

MTU = 7 Bytes → max. Segmentlänge 6 Bytes

- Nachricht 1 (7 Bytes)
  - ⇨ erstes Segment = Controlbyte + 6 Datenbytes
  - ⇨ zweites Segment = Controlbyte + 1 Datenbyte
- Nachricht 2 (2 Bytes)
  - ⇨ erstes Segment = Controlbyte + 2 Datenbytes
- Nachricht 3 (9 Bytes)
  - ⇨ erstes Segment = Controlbyte + 6 Datenbytes
  - ⇨ zweites Segment = Controlbyte + 3 Datenbytes
- Keine weiteren Nachrichten
  - ⇨ C0-Controlbyte

Für jedes gebildete Segment muss ein spezifisches Controlbyte generiert werden. Außerdem wird das Controlbyte C0 generiert, um die Kommunikation auf Standby halten zu können.

C0 (Controlbyte0)			**C1 (Controlbyte1)**			**C2 (Controlbyte2)**		
- 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
Controlbyte	Σ	0	Controlbyte	Σ	6	Controlbyte	Σ	129

Tabelle 3: Flatstream-Ermittlung der Controlbytes für Beispiel mit Standardkonfiguration (Teil 1)

C3 (Controlbyte3)			**C4 (Controlbyte4)**			**C5 (Controlbyte5)**		
- 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
Controlbyte	Σ	130	Controlbyte	Σ	6	Controlbyte	Σ	131

Tabelle 4: Flatstream-Ermittlung der Controlbytes für Beispiel mit Standardkonfiguration (Teil 2)

8.8.4.5 Senden von Daten an ein Modul (Output)

Beim Senden muss das Sendearray im Programmablauf generiert werden. Danach wird es Sequenz für Sequenz über den Flatstream übertragen und vom Modul empfangen.

Information:

Obwohl alle B&R Module mit Flatstream-Kommunikation stets die kompakteste Übertragung in Output-Richtung unterstützen wird empfohlen die Übertragungsarrays für beide Kommunikationsrichtungen gleichermaßen zu gestalten.

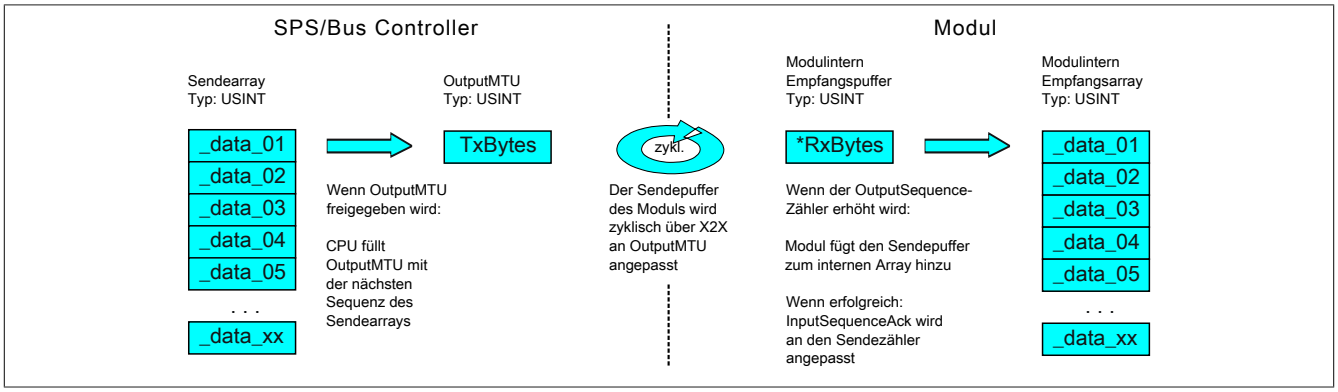


Abbildung 5: Kommunikation per Flatstream (Output)

Nachricht kleiner als OutputMTU

Die Länge der Nachricht sei zunächst kleiner als die OutputMTU. In diesem Fall würde eine Sequenz ausreichen, um die gesamte Nachricht und ein benötigtes Controlbyte zu übertragen.

Algorithmus

Zyklische Statusabfrage: *- Modul überwacht OutputSequenceCounter*
0) Zyklische Prüfungen: - CPU muss OutputSyncAck prüfen → falls OutputSyncAck = 0; OutputSyncBit zurücksetzen und Kanal resynchronisieren - CPU muss Freigabe der OutputMTU prüfen → falls OutputSequenceCounter > InputSequenceAck; MTU nicht freigegeben, weil letzte Sequenz noch nicht bestätigt
1) Vorbereitung (Sendearray anlegen): - CPU muss Nachricht auf zulässige Segmente aufteilen und entsprechende Controlbytes bilden - CPU muss Segmente und Controlbytes zu Sendearray zusammenfügen
2) Senden: - CPU überträgt das aktuelle Element des Sendearrays in die OutputMTU → OutputMTU wird zyklisch in den Sendepuffer des Moduls übertragen, aber noch nicht weiterverarbeitet - CPU muss OutputSequenceCounter erhöhen
Reaktion: *- Modul übernimmt die Bytes des internen Empfangspuffers und fügt sie an das interne Empfangsarray an* *- Modul sendet Bestätigung; schreibt Wert des OutputSequenceCounters auf OutputSequenceAck*
3) Abschluss: - CPU muss OutputSequenceAck überwachen → Eine Sequenz gilt erst dann als erfolgreich übertragen, wenn sie über das OutputSequenceAck bestätigt wurde. Um Übertragungsfehler auch bei der letzten Sequenz zu erkennen, muss sichergestellt werden, dass der *Abschluss* lange genug durchlaufen wird. **Hinweis:** Für eine exakte Überwachung der Kommunikationszeiten sollten die Taskzyklen gezählt werden, die seit der letzten Erhöhung des OutputSequenceCounters vergangen sind. Auf diese Weise kann die Anzahl der Buszyklen abgeschätzt werden, die bislang zur Übertragung benötigt wurden. Übersteigt der Überwachungszähler eine vorgegebene Schwelle, kann die Sequenz als verloren betrachtet werden. (Das Verhältnis von Bus- und Taskzyklus kann vom Anwender beeinflusst werden, sodass der Schwellwert individuell zu ermitteln ist.) - Weitere Sequenzen dürfen erst nach erfolgreicher Abschlussprüfung im nächsten Buszyklus versendet werden.

Nachricht größer als OutputMTU

Das Sendearray, welches im Programmablauf erstellt werden muss, besteht aus mehreren Elementen. Der Anwender muss die Control- und Datenbytes korrekt anordnen und die Arrayelemente nacheinander übertragen. Der Übertragungsalgorithmus bleibt gleich und wird ab dem Punkt *zyklische Prüfungen* wiederholt durchlaufen.

Allgemeines Ablaufdiagramm

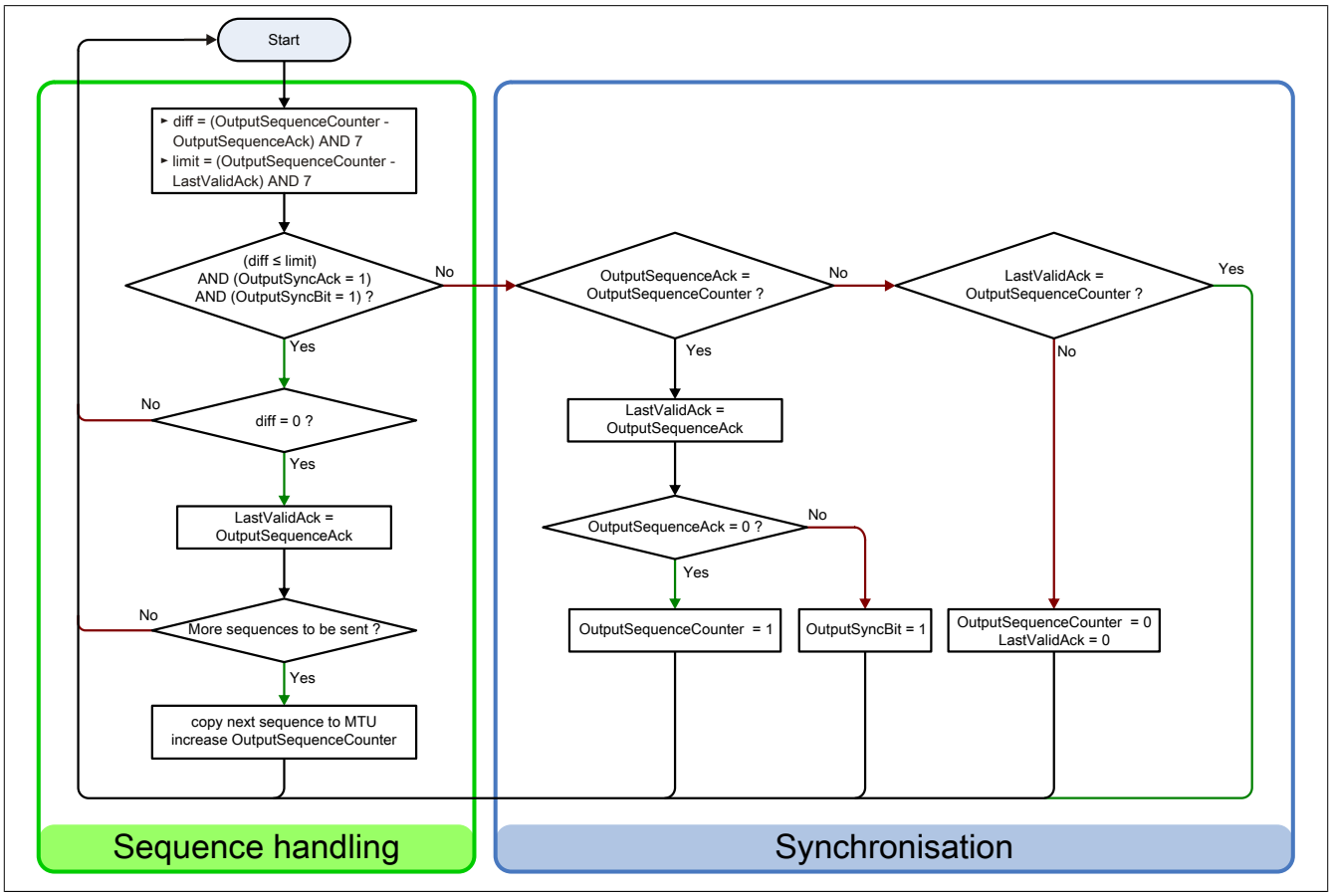


Abbildung 6: Ablaufdiagramm für Output-Richtung

8.8.4.6 Empfangen von Daten aus einem Modul (Input)

Beim Empfangen von Daten wird das Sendearray vom Modul generiert, über den Flatstream übertragen und muss auf dem Empfangsarray abgebildet werden. Die Struktur des ankommenden Datenstroms kann über das Modusregister eingestellt werden. Der Algorithmus zum Empfangen bleibt dabei aber unverändert.

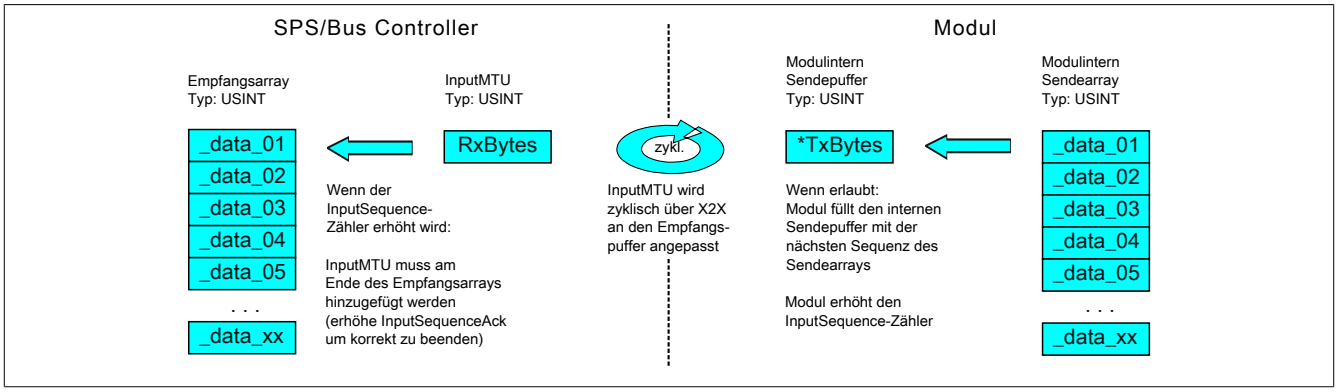


Abbildung 7: Kommunikation per Flatstream (Input)

Algorithmus

0) Zyklische Statusabfrage: - CPU muss InputSequenceCounter überwachen
Zyklische Prüfungen: *- Modul prüft InputSyncAck* *- Modul prüft InputSequenceAck*
Vorbereitung: *- Modul bildet Segmente bzw. Controlbytes und legt Sendearray an*
Aktion: *- Modul überträgt das aktuelle Element des internen Sendearrays in den internen Sendepuffer* *- Modul erhöht InputSequenceCounter*
1) Empfangen (sobald InputSequenceCounter erhöht): - CPU muss Daten aus InputMTU übernehmen und an das Ende des Empfangsarrays anfügen - CPU muss InputSequenceAck an InputSequenceCounter der aktuell verarbeiteten Sequenz angleichen
Abschluss: *- Modul überwacht InputSequenceAck* *→ Eine Sequenz gilt erst dann als erfolgreich übertragen, wenn sie über das InputSequenceAck bestätigt wurde.* *- Weitere Sequenzen werden erst nach erfolgreicher Abschlussprüfung im nächsten Buszyklus versendet.*

Allgemeines Ablaufdiagramm

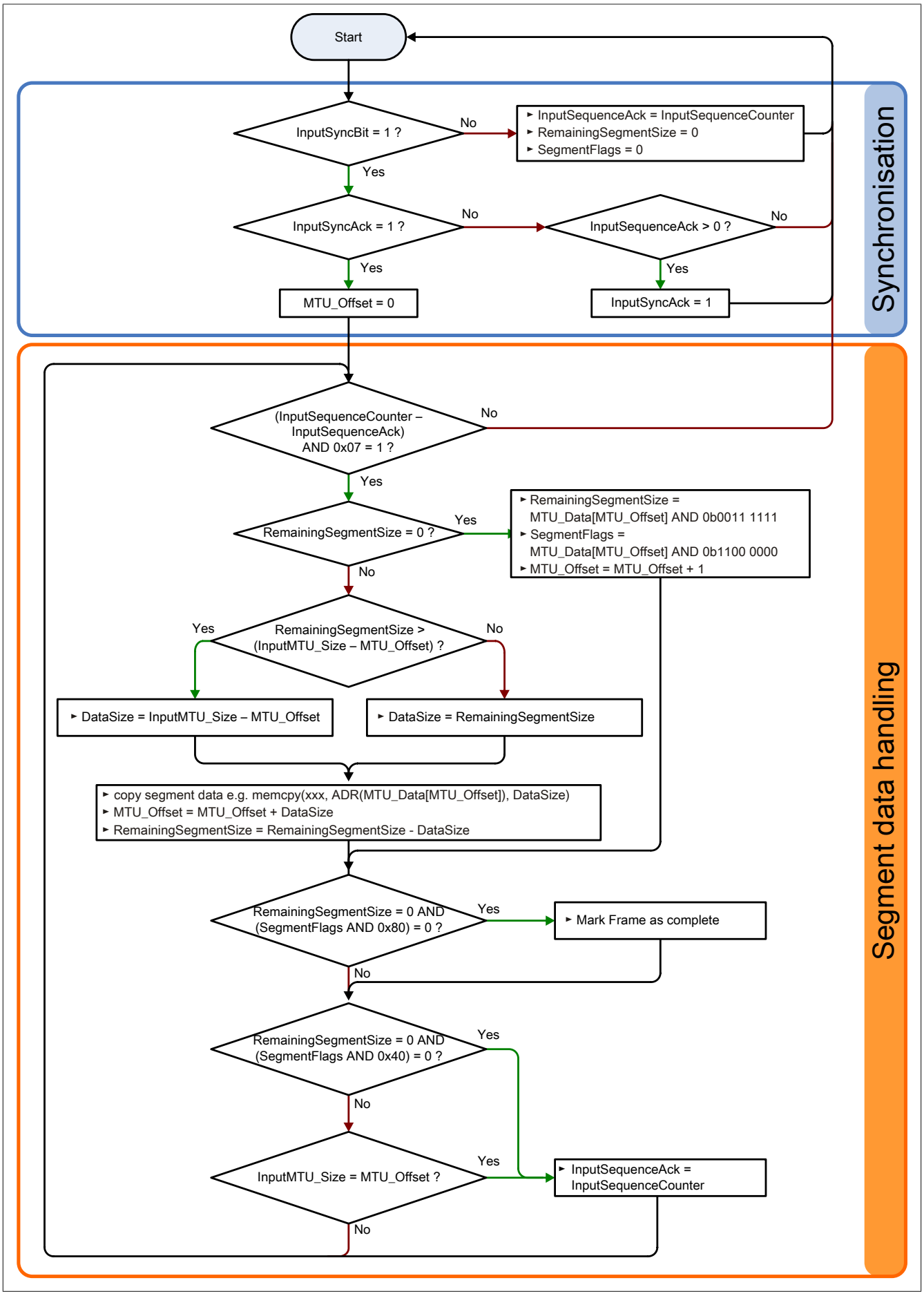


Abbildung 8: Ablaufdiagramm für Input-Richtung

8.8.4.7 Details

Es wird empfohlen die übertragenen Nachrichten in separate Empfangsarrays abzulegen

Nach der Übermittlung eines gesetzten MessageEndBits sollte das Folgesegment zum Empfangsarray hinzugefügt werden. Danach ist die Mitteilung vollständig und kann intern weiterverarbeitet werden. Für die nächste Nachricht sollte ein neues/separates Array angelegt werden.

Information:

Bei der Übertragung mit MultiSegmentMTUs können sich mehrere kurze Nachrichten in einer Sequenz befinden. Im Programmablauf muss sichergestellt sein, dass genügend Empfangsarrays verwaltet werden können. Das Acknowledge-Register darf erst nach Übernahme der gesamten Sequenz angepasst werden.

Wenn ein SequenceCounter um mehr als einen Zähler inkrementiert wird, liegt ein Fehler vor

Anmerkung: Beim Betrieb ohne Forward ist diese Situation sehr unwahrscheinlich.

In diesem Fall stoppt der Empfänger. Alle weiteren eintreffenden Sequenzen werden ignoriert, bis die Sendung mit dem korrekten SequenceCounter wiederholt wird. Durch diese Reaktion erhält der Sender keine Bestätigungen mehr für die abgesetzten Sequenzen. Über den SequenceAck der Gegenstelle kann der Sender die letzte erfolgreich übertragene Sequenz identifizieren und die Übertragung ab dieser Stelle fortsetzen.

Bestätigungen müssen auf Gültigkeit geprüft werden

Wenn der Empfänger eine Sequenz erfolgreich übernommen hat, muss sie bestätigt werden. Dazu übernimmt der Empfänger den mitgesendeten Wert des SequenceCounters und gleicht den SequenceAck daran an. Der Absender liest das SequenceAck und registriert die erfolgreiche Übermittlung. Falls dem Absender eine Sequenz bestätigt wird, die noch nicht abgesendet wurde, muss die Übertragung unterbrochen und der Kanal resynchronisiert werden. Die Synchronisationsbits werden zurückgesetzt und die aktuelle/unvollständige Nachricht wird verworfen. Sie muss nach der Resynchronisierung des Kanals erneut versendet werden.

8.8.4.8 Flatstream Modus

Name:
FlatstreamMode

In Input-Richtung wird das Sende-Array automatisch generiert. Dem Anwender werden über dieses Register 2 Optionen zur Verfügung gestellt, um eine kompaktere Anordnung beim eintreffenden Datenstrom zu erlauben. Nach der Aktivierung muss der Programmablauf zur Auswertung entsprechend angepasst werden.

Information:

Alle B&R Module, die den Flatstream-Modus anbieten, unterstützen in Output-Richtung die Optionen "große Segmente" und "MultiSegmentMTU". Nur für die Input-Richtung muss die kompakte Übertragung explizit erlaubt werden.

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	MultiSegmentMTU	0	Nicht erlaubt (Standard)
		1	Erlaubt
1	Große Segmente	0	Nicht erlaubt (Standard)
		1	Erlaubt
2 - 7	Reserviert		

Standard

Per Standard sind beide Optionen zur kompakten Übertragung in Input-Richtung deaktiviert.

1. Vom Modul werden nur Segmente gebildet, die mindestens ein Byte kleiner sind als die aktivierte MTU. Jede Sequenz beginnt mit einem Controlbyte, sodass der Datenstrom klar strukturiert ist und relativ einfach ausgewertet werden kann.
2. Weil die Länge einer Flatstream-Nachricht beliebig lang sein darf, füllt das letzte Segment der Mitteilung häufig nicht den gesamten Platz der MTU aus. Per Standard werden während eines solchen Übertragungszyklus die restlichen Bytes nicht verwendet.

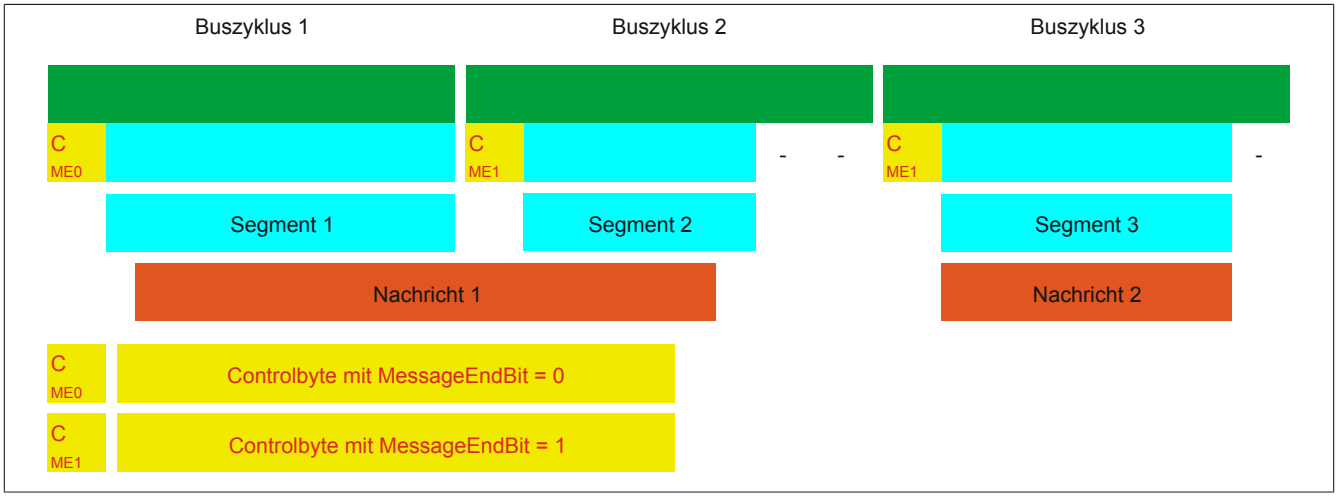


Abbildung 9: Anordnung von Nachrichten in der MTU (Standard)

MultiSegmentMTU erlaubt

Bei dieser Option wird die InputMTU vollständig befüllt (wenn genügend Daten anstehen). Die zuvor frei gebliebenen Rx-Bytes übertragen die nächsten Controlbytes bzw. deren Segmente. Auf diese Weise können die aktivierten Rx-Bytes effizienter genutzt werden.

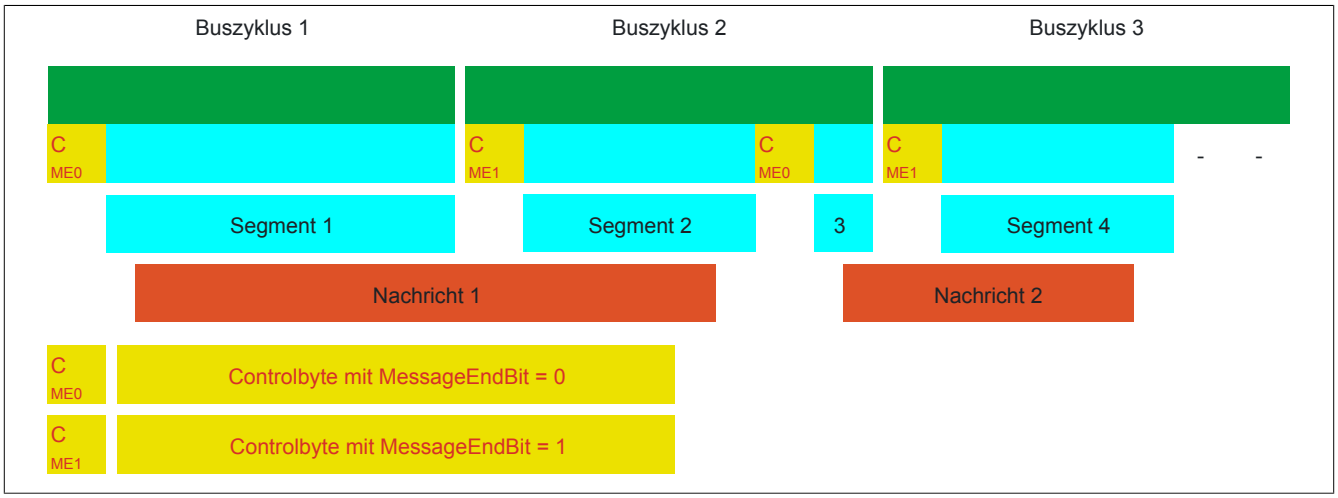


Abbildung 10: Anordnung von Nachrichten in der MTU (MultiSegmentMTU)

Große Segmente erlaubt

Bei der Übertragung sehr langer Mitteilungen bzw. bei der Aktivierung von nur wenigen Rx-Bytes müssen per Standard sehr viele Segmente gebildet werden. Das Bussystem wird stärker belastet als nötig, weil für jedes Segment ein zusätzliches Controlbyte erstellt und übertragen wird. Mit der Option "große Segmente" wird die Segmentlänge unabhängig von der InputMTU auf 63 Bytes begrenzt. Ein Segment darf sich über mehrere Sequenzen erstrecken, das heißt, es können auch reine Sequenzen ohne Controlbyte auftreten.

Information:

Die Möglichkeit eine Nachricht auf mehrere Segmente aufzuteilen bleibt erhalten, das heißt, wird diese Option genutzt und treten Nachrichten mit mehr als 63 Bytes auf, kann die Mitteilung weiterhin auf mehrere Segmente verteilt werden.

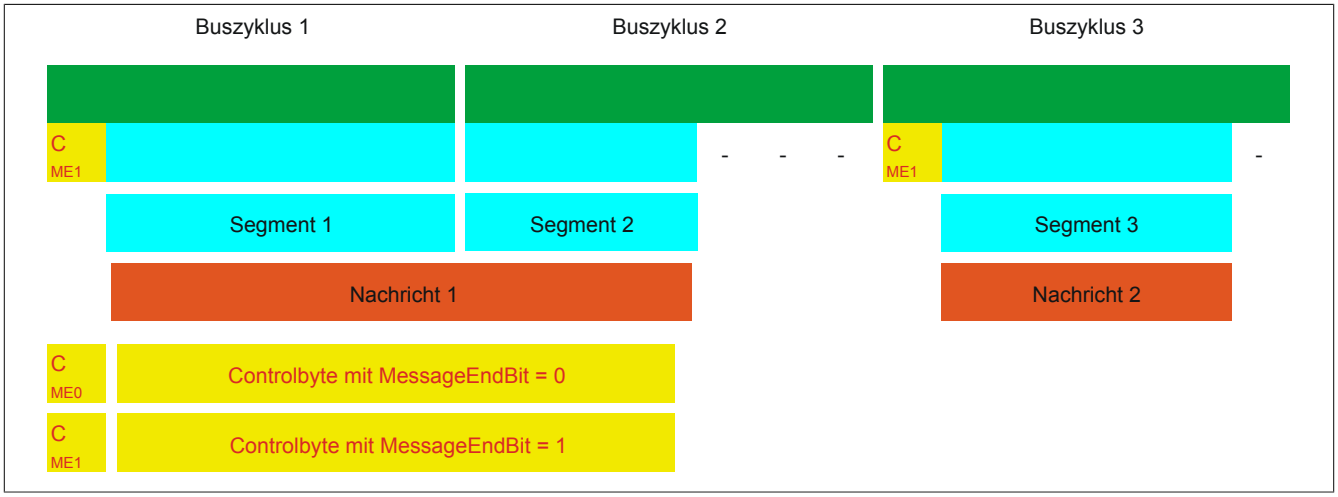


Abbildung 11: Anordnung von Nachrichten in der MTU (große Segmente)

Anwendung beider Optionen

Die beiden Optionen dürfen auch gleichzeitig angewendet werden.

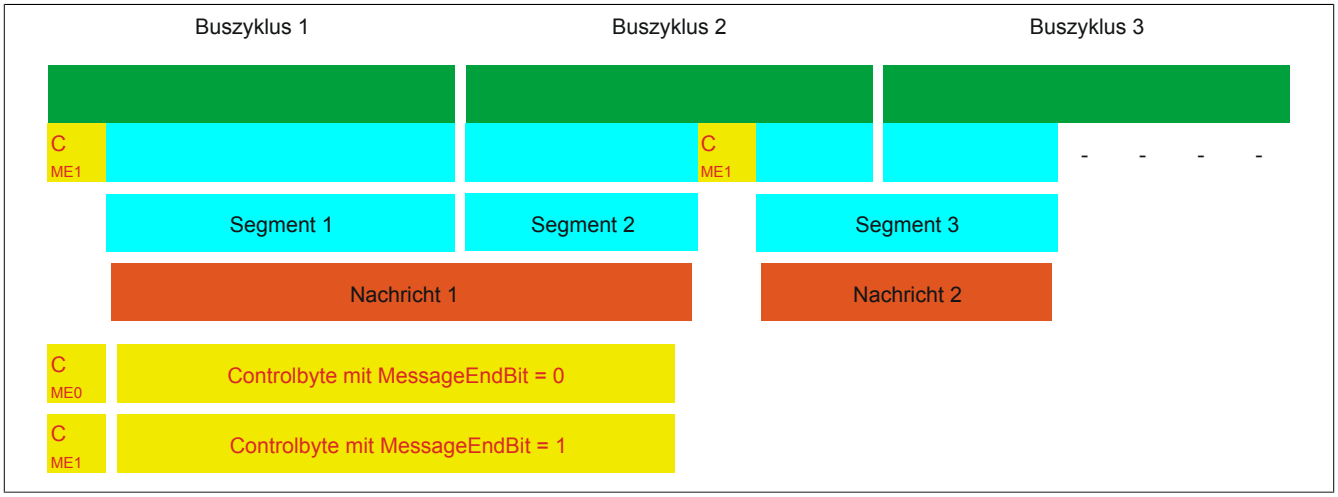


Abbildung 12: Anordnung von Nachrichten in der MTU (große Segmente und MultiSegmentMTU)

8.8.4.9 Anpassung des Flatstreams

Wenn die Strukturierung der Nachrichten verändert wurde, verändert sich auch die Anordnung der Daten im Sende-/Empfangsarray. Für das eingangs genannte Beispiel ergeben sich die folgenden Änderungen.

MultiSegmentMTU

Wenn MultiSegmentMTUs erlaubt sind, können "freie Stellen" in einer MTU genutzt werden. Diese "freien Stellen“ entstehen, wenn das letzte Segment einer Nachricht nicht die gesamte MTU ausnutzt. MultiSegmentMTUs ermöglichen die Verwendung dieser Bits, um die folgenden Controlbytes bzw. Segmente zu übertragen. Im Programmablauf wird das "nextCBPos"-Bit innerhalb des Controlbytes gesetzt, damit der Empfänger das nächste Controlbyte korrekt identifizieren kann.

Beispiel

Es werden 3 unabhängige Nachrichten (7 Bytes, 2 Bytes, 9 Bytes) über eine 7-Byte breite MTU übermittelt. Die Konfiguration erlaubt die Übertragung von MultiSegmentMTUs.

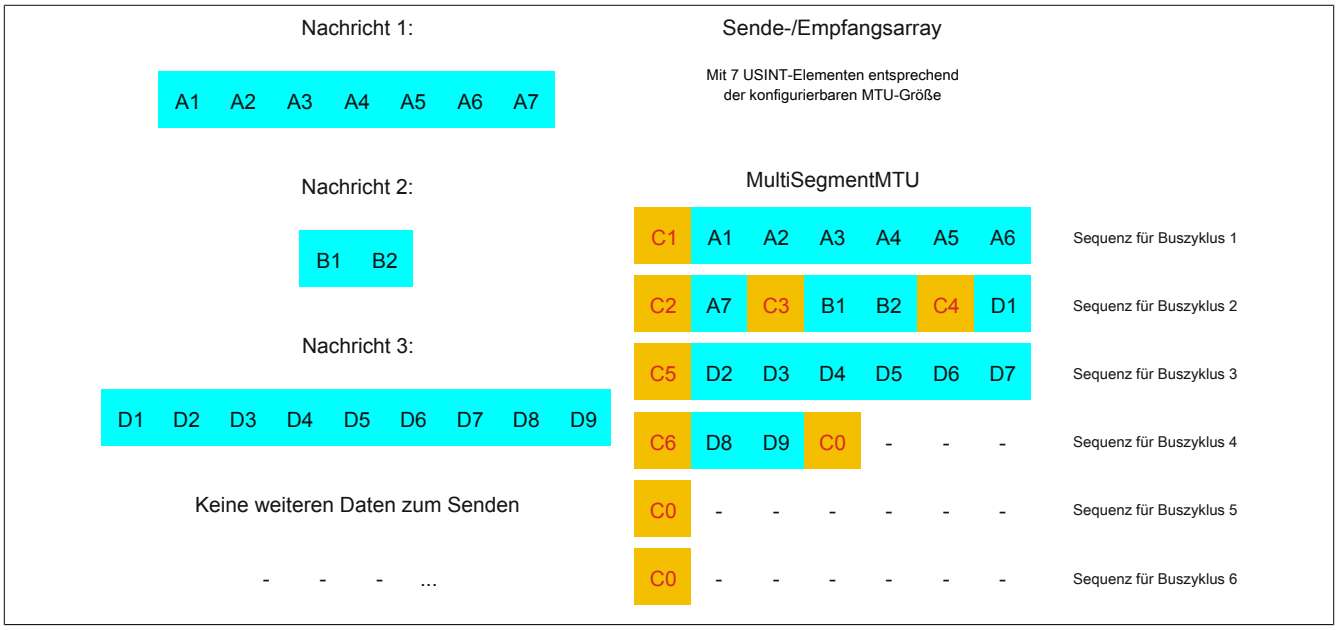


Abbildung 13: Sende-/Empfangsarray (MultiSegmentMTU)

Zunächst müssen die Nachrichten in Segmente geteilt werden. Wie in der Standardkonfiguration muss sichergestellt sein, dass jede Sequenz mit einem Controlbyte beginnt. Die freien Bits in der MTU am Ende einer Nachricht, werden allerdings mit Daten der Folgenachricht aufgefüllt. Bei dieser Option wird das Bit "nextCBPos" immer gesetzt, wenn im Anschluss an das Controlbyte Nutzdaten übertragen werden.

MTU = 7 Bytes → max. Segmentlänge 6 Bytes

- Nachricht 1 (7 Bytes)
  - ⇨ erstes Segment = Controlbyte + 6 Datenbytes (MTU voll)
  - ⇨ zweites Segment = Controlbyte + 1 Datenbyte (MTU noch 5 leere Bytes)
- Nachricht 2 (2 Bytes)
  - ⇨ erstes Segment = Controlbyte + 2 Datenbytes (MTU noch 2 leere Bytes)
- Nachricht 3 (9 Bytes)
  - ⇨ erstes Segment = Controlbyte + 1 Datenbyte (MTU voll)
  - ⇨ zweites Segment = Controlbyte + 6 Datenbytes (MTU voll)
  - ⇨ drittes Segment = Controlbyte + 2 Datenbytes (MTU noch 4 leere Bytes)
- Keine weiteren Nachrichten
  - ⇨ C0-Controlbyte

Für jedes gebildete Segment muss ein spezifisches Controlbyte generiert werden. Außerdem wird das Controlbyte C0 generiert, um die Kommunikation auf Standby halten zu können.

C1 (Controlbyte1)			**C2 (Controlbyte2)**			**C3 (Controlbyte3)**		
- 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
Controlbyte	Σ	70	Controlbyte	Σ	193	Controlbyte	Σ	194

Tabelle 5: Flatstream-Ermittlung der Controlbytes für Beispiel mit MultiSegmentMTU (Teil 1)

Warnung!

Die zweite Sequenz darf erst über den SequenceAck bestätigt werden, wenn sie vollständig verarbeitet wurde. Im Beispiel befinden sich 3 verschiedene Segmente innerhalb der zweiten Sequenz, das heißt, im Programmablauf müssen ausreichend Empfänger-Arrays gehandhabt werden können.

C4 (Controlbyte4)			**C5 (Controlbyte5)**			**C6 (Controlbyte6)**		
- 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
Controlbyte	Σ	7	Controlbyte	Σ	70	Controlbyte	Σ	194

Tabelle 6: Flatstream-Ermittlung der Controlbytes für Beispiel mit MultiSegmentMTU (Teil 2)

Große Segmente

Die Segmente werden auf maximal 63 Bytes begrenzt. Damit können sie größer sein als die aktive MTU. Diese großen Segmente werden bei der Übertragung auf mehrere Sequenzen aufgeteilt. Es können Sequenzen ohne Controlbyte auftreten, die vollständig mit Nutzdaten befüllt sind.

Information:

Um die Größe eines Datenpakets nicht ebenfalls auf 63 Bytes zu begrenzen, bleibt die Möglichkeit erhalten, eine Nachricht in mehrere Segmente zu untergliedern.

Beispiel

Es werden 3 unabhängige Nachrichten (7 Bytes, 2 Bytes, 9 Bytes) über eine 7-Byte breite MTU übermittelt. Die Konfiguration erlaubt die Übertragung von großen Segmenten.

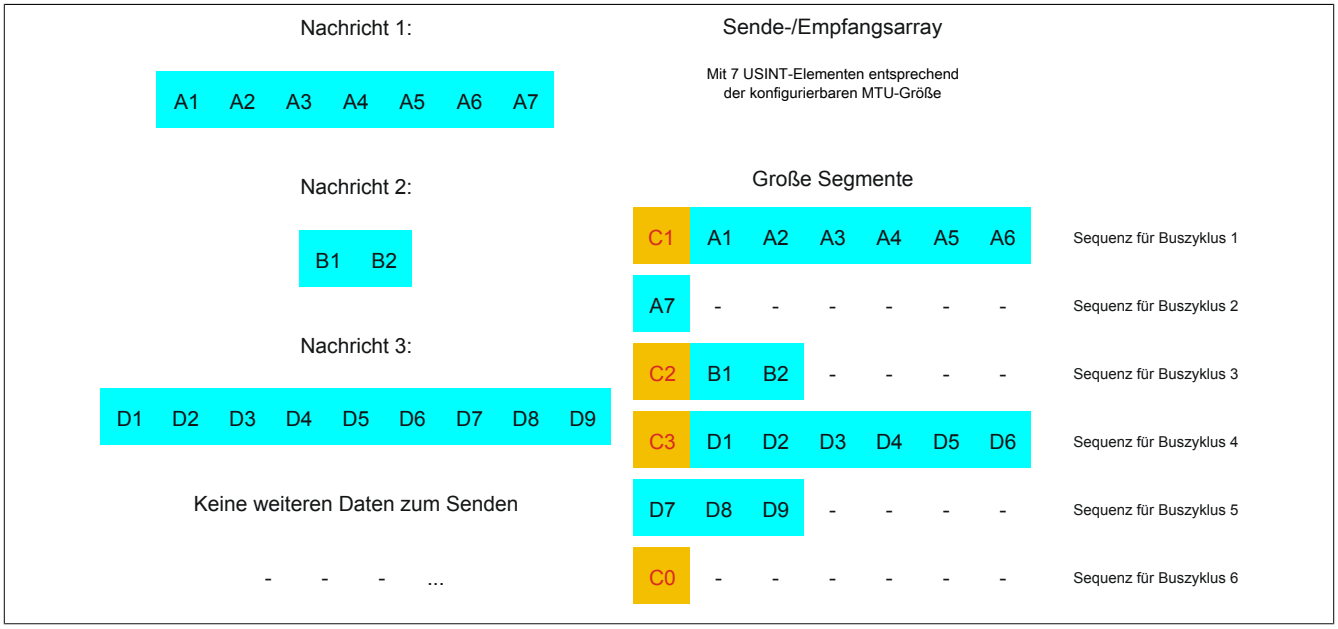


Abbildung 14: Sende-/Empfangsarray (große Segmente)

Zunächst müssen die Nachrichten in Segmente geteilt werden. Durch die Möglichkeit große Segmente zu bilden, müssen Nachrichten seltener geteilt werden, sodass weniger Controlbytes generiert werden müssen.

Große Segmente erlaubt → max. Segmentlänge 63 Bytes

- Nachricht 1 (7 Bytes)
  - ⇨ erstes Segment = Controlbyte + 7 Datenbytes
- Nachricht 2 (2 Bytes)
  - ⇨ erstes Segment = Controlbyte + 2 Datenbytes
- Nachricht 3 (9 Bytes)
  - ⇨ erstes Segment = Controlbyte + 9 Datenbytes
- Keine weiteren Nachrichten
  - ⇨ C0-Controlbyte

Für jedes gebildete Segment muss ein spezifisches Controlbyte generiert werden. Außerdem wird das Controlbyte C0 generiert, um die Kommunikation auf Standby halten zu können.

C1 (Controlbyte1)			C2 (Controlbyte2)			C3 (Controlbyte3)		
- 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
Controlbyte	Σ	135	Controlbyte	Σ	130	Controlbyte	Σ	137

Tabelle 7: Flatstream-Ermittlung der Controlbytes für Beispiel mit großen Segmenten

Große Segmente und MultiSegmentMTU

Beispiel

Es werden 3 unabhängige Nachrichten (7 Bytes, 2 Bytes, 9 Bytes) über eine 7-Byte breite MTU übermittelt. Die Konfiguration erlaubt sowohl die Übertragung von MultiSegmentMTUs als auch von großen Segmenten.

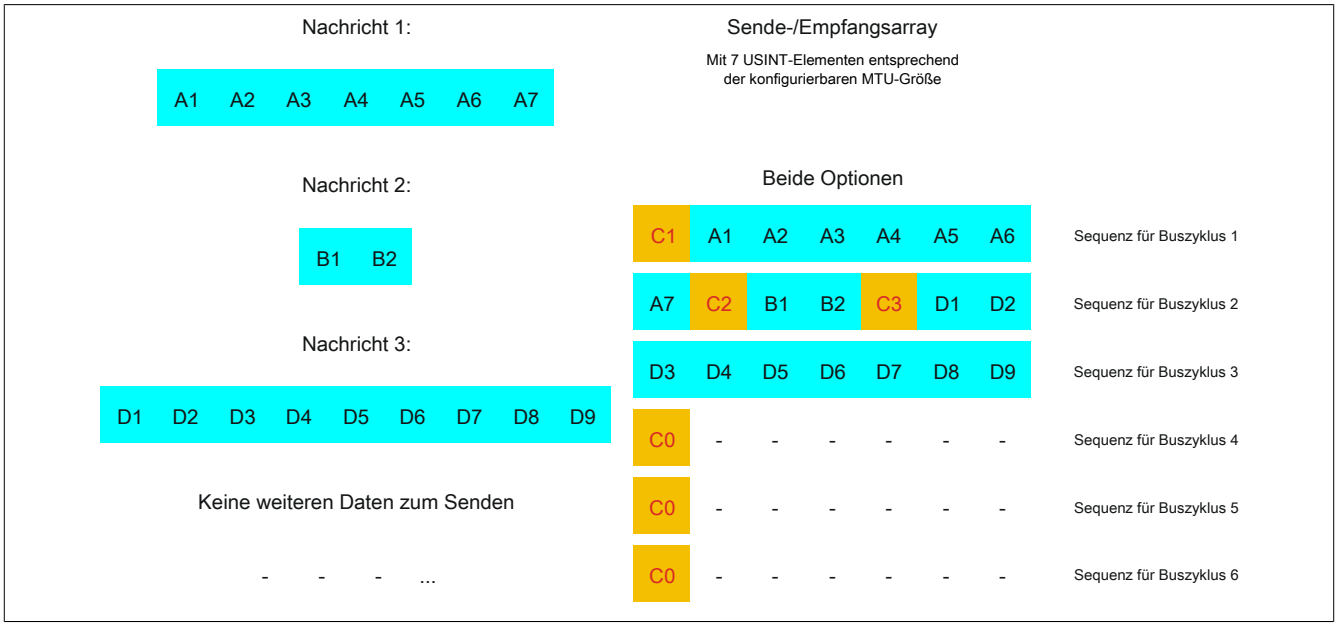


Abbildung 15: Sende-/Empfangsarray (große Segmente und MultiSegmentMTU)

Zunächst müssen die Nachrichten in Segmente geteilt werden. Wenn das letzte Segment einer Nachricht die MTU nicht komplett befüllt, darf sie für weitere Daten aus dem Datenstrom verwendet werden. Das Bit "nextCBPos" muss immer gesetzt werden, wenn das Controlbyte zu einem Segment mit Nutzdaten gehört.
Durch die Möglichkeit große Segmente zu bilden, müssen Nachrichten seltener geteilt werden, sodass weniger Controlbytes generiert werden müssen. Die Generierung der Controlbytes erfolgt auf die gleiche Weise, wie bei der Option "große Segmente".

Große Segmente erlaubt → max. Segmentlänge 63 Bytes

- Nachricht 1 (7 Bytes)
  - ⇨ erstes Segment = Controlbyte + 7 Datenbytes
- Nachricht 2 (2 Bytes)
  - ⇨ erstes Segment = Controlbyte + 2 Datenbytes
- Nachricht 3 (9 Bytes)
  - ⇨ erstes Segment = Controlbyte + 9 Datenbytes
- Keine weiteren Nachrichten
  - ⇨ C0-Controlbyte

Für jedes gebildete Segment muss ein spezifisches Controlbyte generiert werden. Außerdem wird das Controlbyte C0 generiert, um die Kommunikation auf Standby halten zu können.

C1 (Controlbyte1)			C2 (Controlbyte2)			C3 (Controlbyte3)		
- 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
Controlbyte	Σ	135	Controlbyte	Σ	130	Controlbyte	Σ	137

Tabelle 8: Flatstream-Ermittlung der Controlbytes für Beispiel mit großen Segmenten und MultiSegmentMTU

8.8.5 Die "Forward"-Funktion am Beispiel des X2X Link

Bei der "Forward"-Funktion handelt es sich um eine Methode, die Datenrate des Flatstreams deutlich zu erhöhen. Das grundsätzliche Prinzip wird auch in anderen technischen Bereichen angewandt, z. B. beim "Pipelining" für Mikroprozessoren.

8.8.5.1 Das Funktionsprinzip

Bei der Kommunikation mittels X2X Link werden 5 Teilschritte durchlaufen, um eine Flatstream-Sequenz zu übertragen. Eine erfolgreiche Sequenzübertragung benötigt deshalb mindestens 5 Buszyklen.

	Schritt I	Schritt II	Schritt III	Schritt IV	Schritt V
Aktionen	Sequenz aus Sendearray übertragen, SequenceCounter erhöhen	Zyklischer Abgleich MTU und Modulpuffer	Sequenz an Empfangsarray fügen, SequenceAck anpassen	Zyklischer Abgleich MTU und Modulpuffer	Prüfung des SequenceAck
Ressource	Sender (Task zum Versenden)	Bussystem (Richtung 1)	Empfänger (Task zum Empfangen)	Bussystem (Richtung 2)	Sender (Task zur Ack-Prüfung)

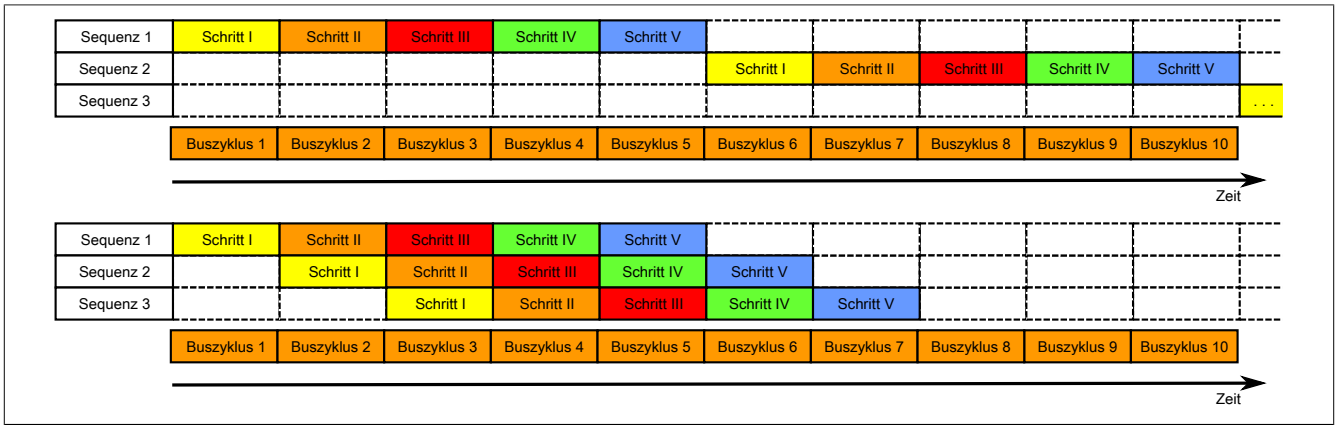


Abbildung 16: Vergleich Übertragung ohne bzw. mit Forward

Jeder der 5 Schritte (Tasks) beansprucht unterschiedliche Ressourcen. Ohne die Verwendung des Forward werden die Sequenzen nacheinander abgearbeitet. Jede Ressource ist nur dann aktiv, wenn sie für die aktuelle Teilaktion benötigt wird.
Beim Forward kann die Ressource, welche ihre Aufgabe abgearbeitet hat, bereits für die nächste Nachricht genutzt werden. Dazu wird die Bedingung zur MTU-Freigabe verändert. Die Sequenzen werden zeitgesteuert auf die MTU gelegt. Die Sendestation wartet nicht mehr auf die Bestätigung durch das SequenceAck und nutzt auf diese Weise die gegebene Bandbreite effizienter.
Im Idealfall arbeiten alle Ressourcen während jedes Buszyklus. Der Empfänger muss weiterhin jede erhaltene Sequenz bestätigen. Erst wenn das SequenceAck angepasst und vom Absender geprüft wurde, gilt die Sequenz als erfolgreich übertragen.

8.8.5.2 Konfiguration

Die Forward-Funktion muss nur für die Input-Richtung freigeschaltet werden. Zu diesem Zweck sind 2 weitere Register zu konfigurieren. Die Flatstream-Module wurden dahingehend optimiert, diese Funktion unterstützen zu können. In Output-Richtung kann die Forward-Funktion genutzt werden, sobald die Größe der OutputMTU vorgegeben ist.

8.8.5.2.1 Anzahl der unbestätigten Sequenzen

Name:
Forward

Über das Register "Forward" stellt der Anwender ein, wie viele unbestätigte Sequenzen das Modul abschicken darf.

Empfehlung:
X2X Link: max. 5
POWERLINK: max. 7

Datentyp	Werte
USINT	1 bis 7 Standard: 1

8.8.5.2.2 Verzögerungszeit

Name:
ForwardDelay

Mit dem Register "ForwardDelay" wird die Verzögerungszeit in µs vorgegeben. Das Modul muss nach dem Versand einer Sequenz diese Zeit abwarten, bevor es im darauf folgenden Buszyklus neue Daten in die MTU schreiben darf. Die Programmroutine zum Empfang von Sequenzen aus einem Modul kann somit auch in einer Taskklasse betrieben werden deren Zykluszeit langsamer ist als der Buszyklus.

Datentyp	Werte
UINT	0 bis 65535 [µs] Standard: 0

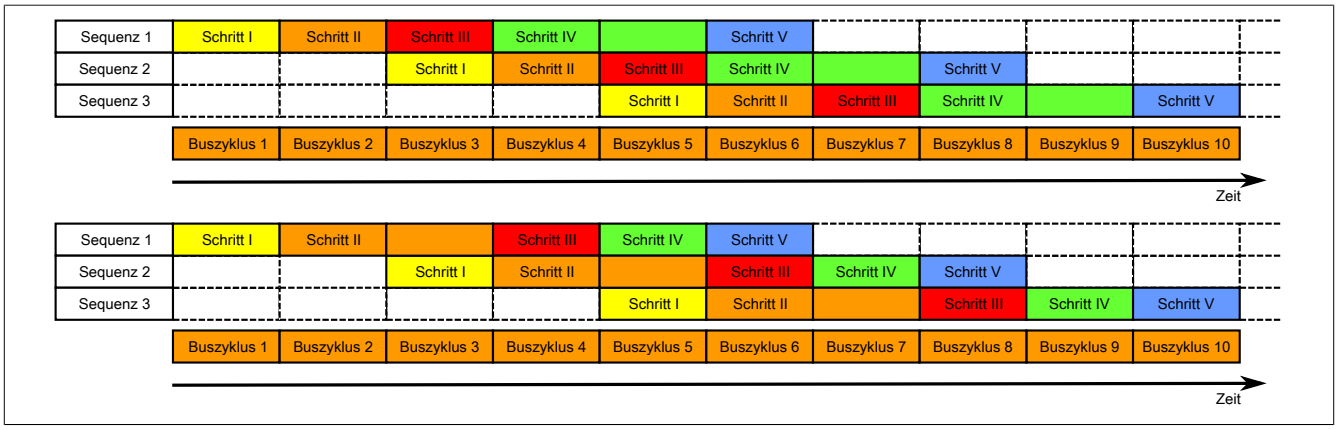


Abbildung 17: Auswirkung des ForwardDelay bei der Flatstream-Kommunikation mit Forward

Im Programmablauf muss sichergestellt werden, dass die CPU alle eintreffenden InputSequences bzw. InputMTUs verarbeitet. Der ForwardDelay-Wert bewirkt in Output-Richtung eine verzögerte Bestätigung und in Input-Richtung einen verzögerten Empfang. Auf diese Weise hat die CPU länger Zeit die eintreffende InputSequence bzw. InputMTU zu verarbeiten.

8.8.5.3 Senden und Empfangen mit Forward

Der grundsätzliche Algorithmus zum Senden bzw. Empfangen von Daten bleibt gleich. Durch den Forward können bis zu 7 unbestätigte Sequenzen abgesetzt werden. Sequenzen können gesendet werden, ohne die Bestätigung der vorangegangenen Nachricht abzuwarten. Da die Wartezeit zwischen Schreiben und Rückmeldung entfällt, können im gleichen Zeitraum erheblich mehr Daten übertragen werden.

Algorithmus zum Senden

Zyklische Statusabfrage: *- Modul überwacht OutputSequenceCounter*
0) Zyklische Prüfungen: - CPU muss OutputSyncAck prüfen → falls OutputSyncAck = 0; OutputSyncBit zurücksetzen und Kanal resynchronisieren - CPU muss Freigabe der OutputMTU prüfen → falls OutputSequenceCounter > OutputSequenceAck + 7, in diesem Fall nicht freigegeben, weil letzte Sequenz noch nicht quittiert
1) Vorbereitung (Sendearray anlegen): - CPU muss Nachricht auf zulässige Segmente aufteilen und entsprechende Controlbytes bilden - CPU muss Segmente und Controlbytes zu Sendearray zusammenfügen
2) Senden: - CPU muss aktuellen Teil des Sendearrays in die OutputMTU übertragen - CPU muss OutputSequenceCounter erhöhen, damit Sequenz vom Modul übernommen wird - CPU darf im nächsten Buszyklus erneut s*enden*, falls MTU freigegeben ist
Reaktion des Moduls, weil OutputSequenceCounter > OutputSequenceAck: *- Modul übernimmt Daten aus internem Empfangspuffer und fügt sie am Ende des internen Empfangsarrays an* *- Modul quittiert; aktuell empfangener Wert des OutputSequenceCounters auf OutputSequenceAck übertragen* *- Modul fragt Status wieder zyklisch ab*
3) Abschluss (Bestätigung): - CPU muss OutputSequenceAck zyklisch überprüfen → Eine Sequenz gilt erst dann als erfolgreich übertragen, wenn sie über das OutputSequenceAck bestätigt wurde. Um Übertragungsfehler auch bei der letzten Sequenz zu erkennen, muss sichergestellt werden, dass der Algorithmus lange genug durchlaufen wird. **Hinweis:** Für eine exakte Überwachung der Kommunikationszeiten sollten die Taskzyklen gezählt werden, die seit der letzten Erhöhung des OutputSequenceCounters vergangen sind. Auf diese Weise kann die Anzahl der Buszyklen abgeschätzt werden, die bislang zur Übertragung benötigt wurden. Übersteigt der Überwachungszähler eine vorgegebene Schwelle, kann die Sequenz als verloren betrachtet werden (das Verhältnis von Bus- und Taskzyklus kann vom Anwender beeinflusst werden, sodass der Schwellwert individuell zu ermitteln ist).

Algorithmus zum Empfangen

0) Zyklische Statusabfrage: - CPU muss InputSequenceCounter überwachen
Zyklische Prüfungen: *- Modul prüft InputSyncAck* *- Modul prüft InputMTU auf Freigabe* *→ Freigabekriterium: InputSequenceCounter > InputSequenceAck + Forward*
Vorbereitung: *- Modul bildet Controlbytes/Segmente und legt Sendearray an*
Aktion: *- Modul überträgt aktuellen Teil des Sendearrays in den Empfangspuffer* *- Modul erhöht InputSequenceCounter* *- Modul wartet auf neuen Buszyklus, nachdem Zeit aus ForwardDelay abgelaufen ist* *- Modul wiederholt Aktion, falls InputMTU freigegeben ist*
1) Empfangen (InputSequenceCounter > InputSequenceAck): - CPU muss Daten aus InputMTU übernehmen und an das Ende des Empfangsarrays anfügen - CPU muss InputSequenceAck an InputSequenceCounter der aktuell verarbeiteten Sequenz angleichen
Abschluss: *- Modul überwacht InputSequenceAck* *→ Eine Sequenz gilt erst dann als erfolgreich übertragen, wenn sie über das InputSequenceAck bestätigt wurde.*

Details/Hintergründe

1. SequenceCounter unzulässig groß (Zählerversatz)
   Fehlersituation: MTU nicht freigegeben
   Wenn beim Senden der Unterschied zwischen SequenceCounter und SequenceAck größer wird, als es erlaubt ist, liegt ein Übertragungsfehler vor. In diesem Fall müssen alle unbestätigten Sequenzen mit dem alten Wert des SequenceCounters wiederholt werden.
2. Prüfung einer Bestätigung
   Nach dem Empfang einer Bestätigung muss geprüft werden, ob die bestätigte Sequenz abgesendet wurde und bisher unbestätigt war. Falls eine Sequenz mehrfach bestätigt wird, liegt ein schwerwiegender Fehler vor. Der Kanal muss geschlossen und resynchronisiert werden (gleiches Verhalten wie ohne Forward).

   > **Information:**
   >
   > **In Ausnahmefällen kann das Modul bei der Verwendung des Forward den OutputSequenceAck um mehr als 1 erhöhen.**
   > **In diesem Fall liegt kein Fehler vor. Die CPU darf alle Sequenzen bis zur Bestätigten als erfolgreich übertragen betrachten.**

3. Sende- und Empfangsarrays
   Der Forward beeinflusst die Struktur des Sende- und Empfangsarrays nicht. Sie werden auf dieselbe Weise gebildet bzw. müssen auf dieselbe Weise ausgewertet werden.

8.8.5.4 Fehlerfall bei Verwendung des Forward

Im industriellen Umfeld werden in der Regel viele verschiedene Geräte unterschiedlicher Hersteller nebeneinander genutzt. Technische Geräte können sich gegenseitig durch ungewollte elektrische oder elektromagnetische Effekte störend beeinflussen. Unter Laborbedingungen können diese Situationen nur bis zu einem bestimmten Punkt nachempfunden und abgesichert werden.

Für die Übertragung per X2X Link wurden Vorkehrungen getroffen, falls es zu derartigen Beeinflussungen kommen sollte. Tritt beim Datentransfer z. B. eine unzulässige Prüfsumme auf, ignoriert das I/O-System die Daten dieses Buszyklus und der Empfänger erhält die letzten gültigen Daten erneut. Bei den herkömmlichen (zyklischen) Datenpunkten kann dieser Fehler oft ignoriert werden. Im darauffolgenden Zyklus wird der gleiche Datenpunkt wieder abgerufen, angepasst und übertragen.

Bei der Flatstream-Kommunikation mit aktiviertem Forward ist die Situation komplexer. Auch hier erhält der Empfänger ein weiteres mal die alten Daten, das heißt, die vorherigen Werte für SequenceAck/SequenceCounter und die alte MTU.

Ausfall einer Bestätigung (SequenceAck)

Wenn durch den Ausfall ein SequenceAck-Wert verloren geht, wurde die MTU bereits korrekt übertragen. Aus diesem Grund darf die nächste Sequenz vom Empfänger weiterverarbeitet werden. Der SequenceAck wird wieder an den mitgelieferten SequenceCounter angepasst und zum Absender zurückgeschickt. Für die Prüfung der eingehenden Bestätigungen folgt daraus, dass alle Sequenzen bis zur zuletzt Bestätigten erfolgreich übertragen sind (siehe Bild Sequenz 1, 2).

Ausfall einer Sendung (SequenceCounter, MTU)

Wenn durch den Ausfall eines Buszyklus der SequenceCounter-Wert bzw. die befüllte MTU verloren geht, kommen beim Empfänger keine Daten an. Zu diesem Zeitpunkt wirkt sich der Fehler noch nicht auf die Routine zum Absenden aus. Die zeitgesteuerte MTU wird wieder freigegeben und kann neu beschrieben werden.
Der Empfänger erhält SequenceCounter-Werte, die mehrfach inkrementiert sind. Damit das Empfangsarray korrekt zusammengestellt wird, darf der Empfänger nur Sendungen verarbeiten, die einen um eins erhöhten SequenceCounter besitzen. Die eintreffenden Sequenzen müssen ignoriert werden, das heißt, der Empfänger stoppt und gibt keine neuen Bestätigungen zurück.
Wenn die maximale Anzahl an unbestätigten Sequenzen abgesendet wurde und keine Bestätigungen zurück kommen, muss der Sender die betroffenen SequenceCounter und die dazugehörigen MTUs wiederholen (siehe Bild Sequenzen 3 und 4).

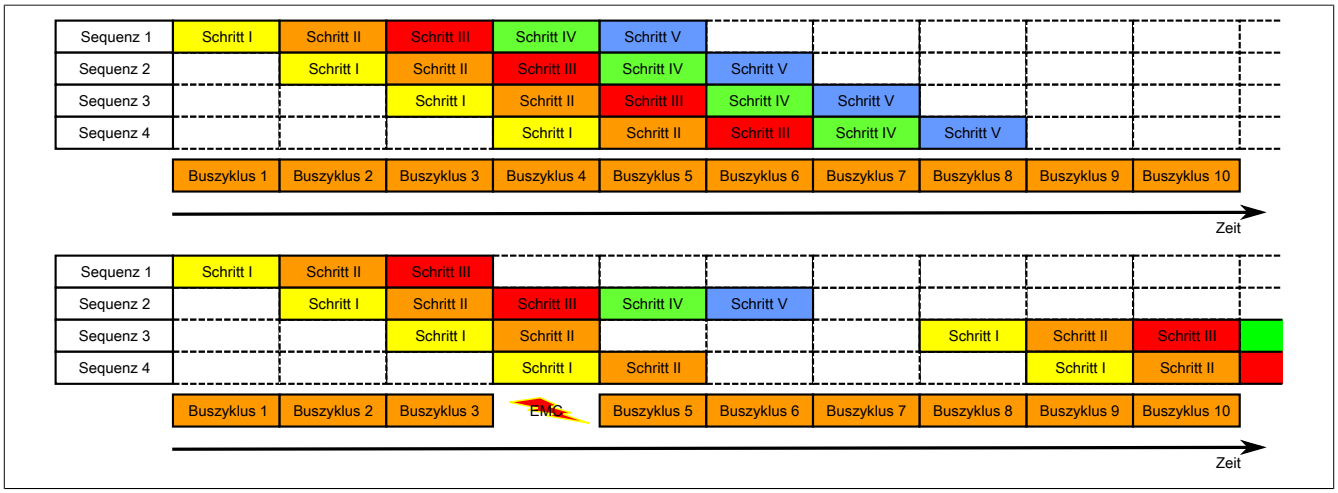


Abbildung 18: Auswirkung eines ausgefallenen Buszyklus

Ausfall der Bestätigung

Bei Sequenz 1 ging aufgrund der Störung die Bestätigung verloren. Im Schritt V der Sequenz 2 werden deshalb die Sequenzen 1 und 2 bestätigt.

Ausfall einer Sendung

Bei Sequenz 3 ging aufgrund der Störung die gesamte Sendung verloren. Der Empfänger stoppt und gibt keine Bestätigungen mehr zurück.
Der Sender sendet zunächst weiter, bis er die max. erlaubte Anzahl an unbestätigten Sendungen abgesetzt hat. Je nach Konfiguration beginnt er frühestens 5 Buszyklen später, die vergeblich abgesendeten Sendungen zu wiederholen.

8.9 M-Bus on Flatstream

Bei der Flatstream-Kommunikation arbeitet das Modul als Bridge zwischen dem X2X Master und einem intelligenten Feldgerät, das an das Modul angeschlossen ist. Der Flatstream-Modus kann sowohl für Punkt-zu-Punkt-Verbindungen als auch bei Bussystemen genutzt werden. Spezifische Algorithmen wie Timeout- oder Prüfsummenüberwachung werden in der Regel automatisch verwaltet. Dem Anwender sind diese Details im Normalbetrieb nicht direkt zugänglich.

Handhabung

Die M-Bus Spezifikation unterscheidet vier verschiedene Frametypen. Aus Sicht der Applikation werden bei der Verwendung des M-Bus über Flatstreams ausschließlich "Long Frames" erzeugt und übertragen. Durch die flexible Gestaltung des M-Bus Protokolls muss der Anwender bei jeder Anfrage die entsprechende Konfiguration des Slaves mitschicken.

Aufbau des Flatstreams		
In-/OutputSequence	Rx-/TxBytes	
(unverändert)	Controlbyte (unverändert)	M-Bus Daten (Flatstream)

8.9.1 Flatstream in Output-Richtung

Die Flatstream-Anfrage

Das Standardprotokoll sieht vor, dass eine Datenanfrage per Flatstream aus einem Hauptteil und zwei sogenannten Index Records besteht.
Ein Index Record ist aus einer Einleitung mit diversen Informationen und einem anschließenden Parameterblock zusammengesetzt.

8.9.1.1 Einleitung

Der Hauptteil dient in erster Linie zur Vergabe einer Synchronisierungsnummer und zur Anmeldung des Protokolltyps.

Hinweis 1
Bei Anmeldung eines undefinierten Protokolltyps arbeitet das Modul mit dem Standardprotokoll.

Hinweis 2
Da zur Zeit nur ein Protokolltyp definiert ist, sollten die Konfigurationsbytes für den Protokolltyp mit 1 definiert werden. Auf diese Weise wird sichergestellt, dass später Erweiterungen im Protokoll erfolgen können, ohne inkompatibel zu bestehenden Projekten zu werden.

Byte	Name	Wert	Beschreibung
1	Framenummer: Zur Synchronisierung in der Applikation	0 - 255	Die Framenummer wird in der Antwort des Moduls wiederholt. Auf diese Weise kann die spätere Antwort des Moduls eindeutig zur Anfrage zugeordnet werden.
2	Index Record Count "i"	2!	Anzahl der folgenden Index Records
3	Protokolltyp	0	Nativer M-Bus (Pegelwandlermodus) - siehe Abschnitt "Nativer M-Bus"
		1	Datenabfrage (Rohdaten/Parameter)
4	Reserviert	1!	
…	Index Record (Konfiguration)		
…	Index Record (Datenanfrage)		

Nativer M-Bus

Der Protokolltyp "Nativer M-Bus" bietet eine universelle Kommunikationsmöglichkeit innerhalb des M-Bus Netzwerkes. Er kann für den Zusammenbau und das Versenden von M-Bus Frames in der Applikation verwendet werden.

Eine herkömmliche Abfrage von Daten ist über die Rohdaten- bzw. Parameterabfrage möglich.

8.9.1.2 Index Record 0

Konfigurationsblock

Im Konfigurationsteil müssen die Schnittstellenparameter vereinbart werden, die das Verhalten des Moduls im M-Bus Netzwerk definieren.

Information:

Beim Standardprotokoll muss der Index Record zur Konfiguration bei jeder Anfrage neu gesendet werden.

Einleitung

Byte	Name	Wert	Beschreibung
1	Index Record Type	0!	Konfiguration des Modul-Interface
2	Counter (Config Parameter)	5!	Anzahl der nachfolgenden M-Bus Parameter
3	Länge des Paramerblocks - Low	19!	Länge der Index-Record-Beschreibung
4	Länge des Paramerblocks - High	0!	Länge der Index-Record-Beschreibung

Parameterblock

Konfigurationsparameter 0 - Adressierungsart

Byte	Name	Wert	Beschreibung
1	Parameternummer	0!	
2	Länge	1!	
3	Adressierungsart	1	Adressierung via Primäradresse
		2	Adressierung via Sekundäradresse

Konfigurationsparameter 1 - Adresse

Byte	Name	Wert	Beschreibung
4	Parameternummer	1!	
5	Länge	4!	
6	Adresse - LowLow	1 - 255	Primäradresse
7	Adresse - LowHigh	0 - 255	0!, falls primäre Adressierung
8	Adresse - HighLow	0 - 255	0!, falls primäre Adressierung
9	Adresse - HighHigh	0 - 255	0!, falls primäre Adressierung

Konfigurationsparameter 2 - Übertragungsrate

Byte	Name	Wert	Beschreibung
10	Parameternummer	2!	
11	Länge	2!	
12	Übertragungsrate Low	0 - 255	Verifizierte Übertragungsraten 300 Bit/s, 2400 Bit/s, 9600 Bit/s
13	Übertragungsrate High	0 - 255	

Konfigurationsparameter 3 - TimeoutOffset

Byte	Name	Wert	Beschreibung
14	Parameternummer	3!	
15	Länge	1!	
16	TimeoutOffset	0 - 255	Zusätzliche Zeit zur Timeout-Überwachung am M-Bus (Auflösung: 10 ms)

Konfigurationsparameter 4 - Zusatzframes

Byte	Name	Wert	Beschreibung
17	Parameternummer	4!	
18	Länge	1!	
19	M-Bus Optionen	Bit 0 = 1	Init Frame senden
		Bit 1 = 1	Application Reset senden
		Bit 6 = 1	Frame Count Bit setzen[1)]
		Bit 7 = 1	Medium und Version anfordern

1) Einige M-Bus Slaves nutzen dieses Bit zur Umschaltung auf einen weiteren Datensatz.

8.9.1.3 Index Record 1

Datenanfrageblock

Im Anfrageteil werden die M-Bus Parameter angefordert, die vom Speicher des M-Bus Slaves abgerufen werden sollen. Der Anwender kann bestimmte Parameter des Slaves oder den gesamten Slave-Speicher verlangen.

Einleitung

Byte	Name	Wert	Beschreibung
1	Index Record Type	1!	Datenanfrage für M-Bus Slave
2	Counter (Data Parameter) = (d + 1)	0	• Kommunikation per nativem M-Bus • M-Bus Rohdaten auslesen
		1 - 20	Anzahl der auszulesenden Parameter
3	Länge des Folgeblocks - Low	0 - 255	• Länge des zu sendenden M-Bus Frames • Länge des Prameterblocks • 0! bei Rohdatenabfrage
4	Länge des Folgeblocks - High	0 - 255	• Länge des zu sendenden M-Bus Frames • Länge des Prameterblocks • 0! bei Rohdatenabfrage
…	Je nach Anfrage: • Nativer M-Bus Frame • Parameterblock		*Entfällt bei Rohdatenabfrage = 0*

M-Bus Frame

Zu sendender M-Bus Frame

Byte	Name	Wert	Beschreibung
1	TxByte 1	0 - 255	Byte 1 in Ausgangsrichtung
2	TxByte 2	0 - 255	Byte 2 in Ausgangsrichtung
n	TxByte n	0 - 255	Byte n in Ausgangsrichtung

Parameterblock

Datenparameter 0

Byte	Name	Wert	Beschreibung
1	Parameternummer	0!	
2	Datenindex	1 - 48	Datenindex im M-Bus Frame

Datenparameter 1

Byte	Name	Wert	Beschreibung
2	Parameternummer	1!	
3	Datenindex	1 - 48	Datenindex im M-Bus Frame

Datenparameter d

Byte	Name	Wert	Beschreibung
…	Parameternummer	d!	
…	Datenindex	1 - 48	Datenindex im M-Bus Frame

8.9.2 Flatstream in Input-Richtung

Flatstream-Antwort

Das Standardprotokoll unterscheidet je nach Anfrage drei verschiedene Antworten.

8.9.2.1 Antwort - Fehler

Die Fehlerantwort erfolgt, wenn das Modul eine ungültige oder unvollständige Anfrage erhält.

Byte	Name	Wert	Beschreibung
1	Framenummer: Zur Synchronisatiton in der Applikation	0 - 255	Die Framenummer wird in der Antwort des Moduls wiederholt. Auf diese Weise kann die Antwort des Moduls eindeutig zur Anfrage zugeordnet werden.
2	Fehlercode - LowLow	0 - 255	Siehe Fehlercodetabelle
3	Fehlercode - LowHigh	0 - 255	Siehe Fehlercodetabelle
4	Fehlercode - HighLow	0 - 255	Siehe Fehlercodetabelle
5	Fehlercode - HighHigh	0 - 255	Siehe Fehlercodetabelle
6	Zusatzinformationen - LowLow	0 - 255	Optional
7	Zusatzinformationen - LowHigh	0 - 255	Optional
8	Zusatzinformationen - HighLow	0 - 255	Optional
9	Zusatzinformationen - HighHigh	0 - 255	Optional

Fehlercodes

Fehlercode und Name	Fehlerbeschreibung
0x11111111	Ein M-Bus Zähler antwortet nicht auf eine Datenanforderung. Dies kann verschiedene Ursachen haben: • Zähler ist nicht angesteckt • Zähler ist defekt • Zähler mit gewählten Adressierungsparametern existiert nicht am Bus
0x22222222	Dieser Fehlercode wird gesendet, wenn per Sekundäradresse adressiert wird und der ausgewählte Zähler nicht antwortet.
0x33333333	Wenn eine ungültige Übertragungsrate per Stream mitgeschickt wird, wird diese nicht ausgewertet. Es wird kein M-Bus Frame versendet, das Flatstream Interface antwortet direkt mit diesem Fehlercode.
0x44444444	Wenn am Bus beim Abfragen der Daten eine Kollision auftritt, wird die Datenabfrage beendet und dieser Fehlercode zurückgeschickt.
0x55555555	Kommunikationsabbruch wegen Überlauf (Siehe Bit 4 in Abschnitt "M-Bus Status" auf Seite 13)
0x66666666	Bevor die Daten vom M-Bus Zähler ausgewertet werden, wird die Checksumme des M-Bus Frames überprüft. Wenn diese nicht stimmt, werden die empfangenen Daten nicht weiter verarbeitet sondern gleich der Fehlercode zur CPU gesendet.
0x77777777	Der Stream (CPU -> IOM) ist nicht korrekt. Möglicherweise stimmt eine Parameteranzahl nicht. Der Stream wird sehr genau geprüft, ein falscher Stream wird also nie verwertet.
0x88888888	Überlast während M-Bus Kommunikation
0x99999999	Kommunikationsabbruch wegen Pegelwandler (Siehe Bit 5 in Abschnitt "M-Bus Status" auf Seite 13)
0xAAAAAAAA	Interpretation der Slavedaten nicht möglich. Der verwendete MBus-Slave ist nicht kompatibel zur Parameterabfrage. Der MBus-Slave muss mit Hilfe des nativem MBus-Protokoll oder per Rohdatenabfrage verwendet werden.

Zusatzinformationen

Zusatzinformationen	Fehlerbeschreibung
0x00000001	Anzahl der Index Records ist kleiner als 2
0x00000002	Streamlänge passt nicht
0x00000004	Indexnummern stimmen nicht
0x00000008	Anzahl der Parameter pro Index Record falsch
0x00000010	Indexlänge zu klein
0x00000020	Parameternummer beim Index Record 0 falsch
0x00000040	Parameterlänge beim Index Record 0 falsch
0x00000080	Ungültiger Adressierungstyp
0x00000100	Ungültige Adresse
0x00000200	Ungültige Übertragungsrate
0x00000400	Ungültiger TimeoutOffset
0x00000800	Ungültige Zusatzframe-Konfiguration

8.9.2.2 Antwort - nativer M-Bus

Diese Antwort korrespondiert mit einem erfolgreich abgesetzten M-Bus Frame, der innerhalb der Applikation erstellt wurde.

Byte	Name	Wert	Beschreibung
1	Framenummer: Zur Synchronisation in der Applikation	0 - 255	Die Framenummer wird in der Antwort des Moduls wiederholt. Auf diese Weise kann die Antwort des Moduls eindeutig zur Anfrage zugeordnet werden.
2	Reserviert	0	
…	Antwort		

Antwort

Byte	Name	Wert	Beschreibung
1	RxByte 1	0 - 255	Byte 1 in Eingangsrichtung
2	RxByte 2	0 - 255	Byte 2 in Eingangsrichtung
n	RxByte n	0 - 255	Byte n in Eingangsrichtung

8.9.2.3 Antwort - Rohdaten

Die Rohdatenantwort erfolgt, wenn der gesamte Speicher des M-Bus Slaves angefordert wurde.

Byte	Name	Wert	Beschreibung
1	Framenummer: Zur Synchronisation in der Applikation	0 - 255	Die Framenummer wird in der Antwort des Moduls wiederholt. Auf diese Weise kann die Antwort des Moduls eindeutig zur Anfrage zugeordnet werden.
2	M-Bus Status	0 - 255	Statusinformationen aus M-Bus Header
3	Rohdatenframe	0 - 255	Umfasst alle Bytes, die vom M-Bus Slave versendet werden.
…		0 - 255	

8.9.2.4 Antwort - Parameter

Die Parameterantwort erfolgt, wenn ein oder mehrere Parameter eines M-Bus Slaves angefordert wurden.

Byte	Name	Wert	Beschreibung
1	Framenummer: Zur Synchronisation in der Applikation	0 - 255	Die Framenummer wird in der Antwort des Moduls wiederholt. Auf diese Weise kann die Antwort des Moduls eindeutig zur Anfrage zugeordnet werden.
2	M-Bus Status	0 - 255	Statusinformationen aus M-Bus Header
3	Parameter Count "p"	0 - 255	Anzahl der eintreffenden Parameter
4	M-Bus Adresse	0 - 255	Primäradresse
5	Seriennummer - LowLow	0 - 255	Sekundäradresse
6	Seriennummer - LowHigh	0 - 255	Sekundäradresse
7	Seriennummer - HighLow	0 - 255	Sekundäradresse
8	Seriennummer - HighHigh	0 - 255	Sekundäradresse
9	VendorID - Low \ Version	0 - 255	Siehe "MBus-Option (IndexRecord 0)" auf Seite 44
10	VendorID - High \ Medium	0 - 255	Siehe "MBus-Option (IndexRecord 0)" auf Seite 44
11	Datenstruktur (M-Bus)	1	Datenstruktur fix
		2	Datenstruktur variabel
…	Eintreffendender Parameter 1 bis p		*Entfällt falls Parameter Count = 0*

Eintreffendender Parameter

Byte	Name	Wert	Beschreibung
1	Medium	0 - 255	Medium des folgenden Zählerwertes
2	Index	0 - 255	Index des folgenden Zählerwertes
3	Datenlänge	1 - 8	Länge des Zählerwertes
		255	Falls Parameternummer ungültig
4	DIF	0 - 255	0!, falls fixe Datenstruktur
5	VIF	0 - 255	0!, falls fixe Datenstruktur
6	Zählerwert	0 - 255	LowLowLowLowLowLowLowLow
…		…	…
13		0 - 255	HighHighHighHighHighHighHigh

8.10 Minimale Zykluszeit

Die minimale Zykluszeit gibt an, bis zu welcher Zeit der Buszyklus heruntergefahren werden kann, ohne dass Kommunikationsfehler auftreten. Es ist zu beachten, dass durch sehr schnelle Zyklen die Restzeit zur Behandlung der Überwachungen, Diagnosen und azyklischen Befehle verringert wird.

Minimale Zykluszeit
200 µs

8.11 Minimale I/O-Updatezeit

Die minimale I/O-Updatezeit gibt an, bis zu welcher Zeit der Buszyklus heruntergefahren werden kann, so dass in jedem Zyklus ein I/O-Update erfolgt.

Minimale I/O-Updatezeit
1 s