X20(c)CS1020

1 Allgemeines

Neben Standard-I/Os sind häufig komplexe Geräte dezentral anzubinden. Genau für diesen Fall sind die X20CS Kommunikationsmodule gedacht. Sie sind als normale X20 Elektronikmodule an jeder Stelle der dezentralen Backplane einsetzbar.

- RS232 Schnittstelle für das serielle dezentrale Anbinden komplexer Geräte an das X20 System

2 Coated Module

Coated Module sind X20 Module mit einer Schutzbeschichtung der Elektronikbaugruppe. Die Beschichtung schützt X20c Module vor Betauung und Schadgasen.
Die Elektronik der Module ist vollständig funktionskompatibel zu den entsprechenden X20 Modulen.

In diesem Datenblatt werden zur Vereinfachung nur Bilder und Modulbezeichnungen der unbeschichteten Module verwendet.

Die Beschichtung wurde nach folgenden Normen qualifiziert:

- Betauung: BMW GS 95011-4, 2x 1 Zyklus
- Schadgas: EN 60068-2-60, Methode 4, Exposition 21 Tage

3 Bestelldaten

Bestellnummer	Kurzbeschreibung	Abbildung
	Kommunikation im X20 Elektronikmodul	
X20CS1020	X20 Schnittstellenmodul, 1 RS232-Schnittstelle, max. 115,2 kBit/s	
X20cCS1020	X20 Schnittstellenmodul, beschichtet , 1 RS232-Schnittstelle, max. 115,2 kBit/s	
	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	
X20cBM11	X20 Busmodul, beschichtet, 24 VDC codiert, interne I/O-Versorgung durchverbunden	
	Feldklemmen	
X20TB06	X20 Feldklemme, 6-polig, 24 VDC codiert	
X20TB12	X20 Feldklemme, 12-polig, 24 VDC codiert	

Tabelle 1: X20CS1020, X20cCS1020 - Bestelldaten

4 Technische Daten

Bestellnummer	X20CS1020	X20cCS1020
Kurzbeschreibung		
Kommunikationsmodul	1x RS232	
Allgemeines		
B&R ID-Code	0x1FCF	0xE7F2
Statusanzeigen	Datenübertragung, Betriebszustand, Modulstatus	
Diagnose		
Modul Run/Error	Ja, per Status-LED und SW-Status	
Datenübertragung	Ja, per Status-LED	
Leistungsaufnahme		
Bus	0,01 W	
I/O-intern	1,44 W	
Zusätzliche Verlustleistung durch Aktoren (ohmsch) [W]	-	
Zulassungen		
CE	Ja	
KC	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	
DNV GL	Temperature: **B** (0 - 55 °C) Humidity: **B** (up to 100%) Vibration: **B** (4 g) EMC: **B** (bridge and open deck)	
LR	ENV1	-
Schnittstellen		
Schnittstelle IF1		
Signal	RS232	
Ausführung	Kontaktierung über 12-polige Feldklemme X20TB12	
max. Reichweite	900 m	
Übertragungsrate	max. 115,2 kBit/s	
FIFO	1 kByte	
Handshakeleitungen	RTS, CTS	
Controller	UART Typ 16C550 kompatibel	
Elektrische Eigenschaften		
Potenzialtrennung	RS232 (IF1) zu Bus getrennt RS232 (IF1) 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	Siehe Abschnitt "Derating"	
Lagerung	-40 bis 85°C	
Transport	-40 bis 85°C	
Luftfeuchtigkeit		
Betrieb	5 bis 95%, nicht kondensierend	Bis 100%, kondensierend
Lagerung	5 bis 95%, nicht kondensierend	
Transport	5 bis 95%, nicht kondensierend	
Mechanische Eigenschaften		
Anmerkung	Feldklemme 1x X20TB06 oder X20TB12 gesondert bestellen Busmodul 1x X20BM11 gesondert bestellen	Feldklemme 1x X20TB06 oder X20TB12 gesondert bestellen Busmodul 1x X20cBM11 gesondert bestellen
Rastermaß	12,5 $^{+0,2}$ mm	

Tabelle 2: X20CS1020, X20cCS1020 - Technische Daten

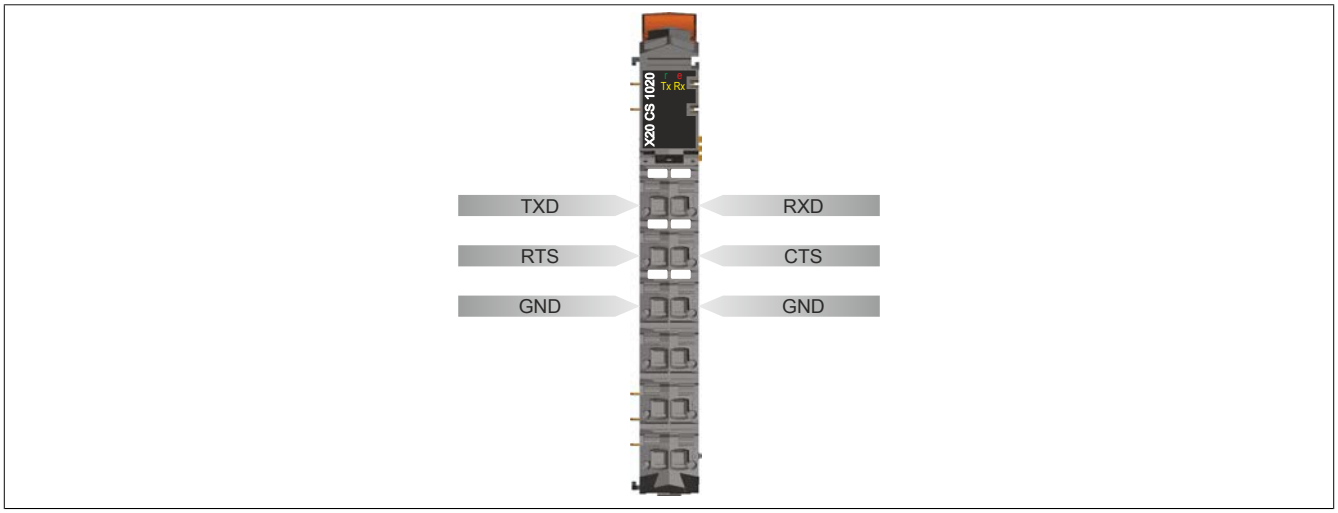
5 Status-LEDs

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 RESET
			Double Flash	Modus BOOT (während Firmware-Update)[1)]
			Blinkend	Modus PREOPERATIONAL
			Ein	Modus RUN
	e	Rot	Aus	Modul nicht versorgt oder alles in Ordnung
			Single Flash	I/O Fehler ist aufgetreten, siehe "Statusbits Fehlermeldungen" auf Seite 14
			Ein	Fehler- oder Resetzustand
	e + r	Rot ein / grüner Single Flash		Firmware ist ungültig
	Tx	Gelb	Ein	Das Modul sendet Daten über die RS232-Schnittstelle
	Rx	Gelb	Ein	Das Modul empfängt Daten über die RS232-Schnittstelle

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

6 Anschlussbelegung

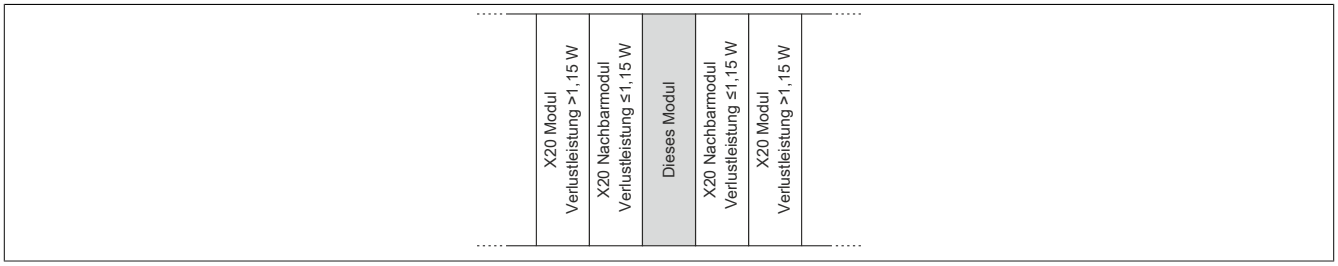


7 Derating

Bei einem Betrieb unter 55°C ist kein Derating zu beachten.

Bei einem Betrieb über 55°C dürfen die Module links und rechts von diesem Modul eine maximale Verlustleistung von 1,15 W haben!

Ein Beispiel zur Berechnung der Verlustleistung von I/O-Modulen ist im X20 Anwenderhandbuch, Abschnitt "Mechanische und elektrische Konfiguration - Verlustleistung von I/O-Modulen" zu finden.



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 2 - Stream und Funktionsmodell 254 - Cyclicstream

Die Funktionsmodelle "Stream" und "Cyclicstream" nutzen einen modulspezifischen Treiber des Betriebssystems. Die Schnittstelle kann mit Hilfe der Bibliothek "DvFrame" gesteuert und während der Laufzeit umkonfiguriert werden.

Funktionsmodell "Stream"

Beim Funktionsmodell "Stream" kommuniziert die CPU mit dem Modul azyklisch. Die Schnittstelle ist relativ komfortabel zu bedienen, arbeitet allerdings zeitlich unbestimmt.

Funktionsmodell "Cyclicstream"

Das Funktionsmodell "Cyclicstream" wurde zu einem späteren Zeitpunkt implementiert. Aus Sicht der Applikation gibt es keine Unterschiede zum Funktionsmodell "Stream". Intern werden jedoch zyklische I/O-Register genutzt, sodass die Kommunikation zeitlich determiniert abläuft.

Information:

- **Um die Funktionsmodelle "Stream" und "Cyclicstream" nutzen zu können, ist die Verwendung von B&R Steuerungen des Typs "SG4" notwendig.**
- **Diese Funktionsmodelle können nur im X2X Link und in POWERLINK-Netzwerken verwendet werden.**

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Modul - Konfiguration						
-	AsynSize	-				
Statusmeldungen – Konfiguration						
50	CfO_RxStateIgnoreMask	UINT				•
6273	CfO_ErrorID0007	USINT				•
Statusmeldungen – Kommunikation						
6145	ErrorByte	USINT	•			
	StartBitError	Bit 0				
	StopBitError	Bit 1				
	ParityError	Bit 2				
	RXoverrun	Bit 3				
6209	ErrorQuitByte	USINT			•	
	QuitStartBitError	Bit 0				
	QuitStopBitError	Bit 1				
	QuitParityError	Bit 2				
	QuitRXoverrun	Bit 3				

8.3 Funktionsmodell 254 - Flatstream

Der Flatstream ermöglicht eine unabhängige Kommunikation zwischen einem X2X-Master und dem Modul. Für das Modul wurde diese Schnittstelle als separates Funktionsmodell implementiert. Die seriellen Informationen werden mittels zyklischer Ein- und Ausgangsregister übertragen. Zur Steuerung des Datenstroms werden die sogenannten Sequenz- und Steuerbytes genutzt (siehe "Die Flatstream-Kommunikation" auf Seite 15).

Bei Verwendung des Funktionsmodells Flatstream kann der Anwender wählen, ob er die Automation Studio Bibliothek "AsFltGen" zur Implementierung nutzt oder die Flatstream-Behandlung individuell an die Anforderungen der Applikation anpassen möchte.

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Serielle Schnittstelle – Konfiguration						
1	phyMode	USINT				•
12	phyBaud	UDINT				•
3	phyData	USINT				•
5	phyStop	USINT				•
7	phyParity	USINT				•
Handshake – Konfiguration						
66	rxlLock	UINT				•
70	rxlUnlock	UINT				•
34	hssXOn	UINT				•
38	hssXOff	UINT				•
42	hssPeriod	UINT				•
19	hshTxF	USINT				•
29	hshRxF	USINT				•
27	hshSet	USINT				•
25	hshClr	USINT				•
17	hshInv	USINT				•
Frame – Konfiguration						
74	rxCto	UINT				•
106	txCto	UINT				•
78	rxEomSize	UINT				•
110	txEomSize	UINT				•
Index * 4 + 82	rxEomCharN (Index N = 0 bis 3)	UINT				•
Index * 4 + 114	txEomCharN (Index N = 0 bis 3)	UINT				•
Statusmeldungen – Konfiguration						
50	CfO_RxStateIgnoreMask	UINT				•
6273	CfO_ErrorID0007	USINT				•
Statusmeldungen – Kommunikation						
6145	ErrorByte	USINT	•			
	StartBitError	Bit 0				
	StopBitError	Bit 1				
	ParityError	Bit 2				
	RXoverrun	Bit 3				
6209	ErrorQuitByte	USINT			•	
	QuitStartBitError	Bit 0				
	QuitStopBitError	Bit 1				
	QuitParityError	Bit 2				
	QuitRXoverrun	Bit 3				
Flatstream						
225	OutputMTU	USINT				•
227	InputMTU	USINT				•
229	Mode	USINT				•
231	Forward	USINT				•
238	ForwardDelay	UINT				•
128	InputSequence	USINT	•			
Index + 128	RxByteN (Index N = 1 bis 27)	USINT	•			
160	OutputSequence	USINT			•	
Index + 160	TxByteN (Index N = 1 bis 27)	USINT			•	

8.4 Funktionsmodell 254 - Bus Controller

Das Funktionsmodell "Bus Controller" entspricht dem Funktionsmodell "Flatstream" in reduzierter Form. Statt bis zu 27 Tx- bzw. Rx-Bytes können max. 7 Tx- bzw. Rx-Bytes genutzt werden.

Register	Offset[1]	Name	Datentyp	Lesen		Schreiben	
				Zyklisch	Azyklisch	Zyklisch	Azyklisch
Serielle Schnittstelle – Konfiguration							
257	-	phyMode_CANIO	USINT				•
268	-	phyBaud_CANIO	UDINT				•
259	-	phyData_CANIO	USINT				•
261	-	phyStop_CANIO	USINT				•
263	-	phyParity_CANIO	USINT				•
Handshake – Konfiguration							
322	-	rxILock_CANIO	UINT				•
326	-	rxIUnlock_CANIO	UINT				•
290	-	hssXOn_CANIO	UINT				•
294	-	hssXOff_CANIO	UINT				•
298	-	hssPeriod_CANIO	UINT				•
275	-	hshTxF_CANIO	USINT				•
285	-	hshRxF_CANIO	USINT				•
281	-	hshClr_CANIO	USINT				•
283	-	hshSet_CANIO	USINT				•
287	-	hshFrm_CANIO	USINT				•
273	-	hshInv_CANIO	USINT				•
Frame – Konfiguration							
330	-	rxCto_CANIO	UINT				•
362	-	txCto_CANIO	UINT				•
334	-	rxEomSize_CANIO	UINT				•
366	-	txEomSize_CANIO	UINT				•
Index*4 + 338	-	rxEomCharN (N = 0 bis 3)	UINT				•
Index*4 + 370	-	txEomCharN (N = 0 bis 3)	UINT				•
Statusmeldungen – Konfiguration							
306	-	CfO_RxStateIgnoreMask_CANIO	UINT				•
6273	-	CfO_ErrorID0007	USINT				•
Statusmeldungen – Kommunikation							
6145	-	ErrorByte	USINT		•		
		StartBitError	Bit 0				
		StopBitError	Bit 1				
		ParityError	Bit 2				
		RXoverrun	Bit 3				
6209	-	ErrorQuitByte	USINT				•
		QuitStartBitError	Bit 0				
		QuitStopBitError	Bit 1				
		QuitParityError	Bit 2				
		QuitRXoverrun	Bit 3				
Flatstream							
225	-	OutputMTU	USINT				•
227	-	InputMTU	USINT				•
229	-	Mode	USINT				•
231	-	Forward	USINT				•
238	-	ForwardDelay	UINT				•
128	0	InputSequence	USINT	•			
Index + 128	Index	RxByteN (Index N = 1 bis 7)	USINT	•			
160	0	OutputSequence	USINT			•	
Index + 160	Index	TxByteN (Index N = 1 bis 7)	USINT			•	

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

8.4.1 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".

8.4.2 CAN-I/O Bus Controller

Das Modul belegt an CAN-I/O 1 analogen logischen Steckplatz.

8.5 Serielle Schnittstelle - Konfiguration

Für den Betrieb der seriellen Schnittstelle müssen vom Anwender 5 Register konfiguriert werden.

8.5.1 Mode_IF

Name:
phyMode
phyMode_CANIO

Mit Hilfe dieses Registers wird der aktuelle Betriebsmodus der Schnittstelle vereinbart.

Das Aktivieren der Schnittstelle darf erst nach der vollständigen Konfiguration der anderen Register erfolgen. Falls eine Parameteränderung notwendig ist, muss die Schnittstelle zuerst deaktiviert werden.

Datentyp	Werte	Bedeutung
USINT	0	RS232-Schnittstelle deaktiviert (Bus Controller Default)
	2	RS232-Schnittstelle aktiv

8.5.2 Baudrate_IF

Name:
phyBaud
phyBaud_CANIO

Mit Hilfe dieses Register wird die Baudrate der Schnittstelle in Bit/s eingestellt.

Datentyp	Werte	Bedeutung
UDINT	1200	1,2 kBaud
	2400	2,4 kBaud
	4800	4,8 kBaud
	9600	9,6 kBaud
	19200	19,2 kBaud
	38400	38,4 kBaud
	57600	57,6 kBaud (Bus Controller Default)
	115200	115,2 kBaud

8.5.3 Databit_IF

Name:
phyData
phyData_CANIO

Mit Hilfe dieses Register wird die Anzahl der zu übertragenden Bits pro Zeichen vorgegeben.

Datentyp	Werte	Bedeutung
USINT	7	7 Datenbits
	8	8 Datenbits (Bus Controller Default)

8.5.4 Stoppbit_IF

Name:
phyStop
phyStop_CANIO

Mit Hilfe dieses Register wird die Anzahl der Stoppbits vorgeben.

Datentyp	Werte	Bedeutung
USINT	2	1 Stoppbit (Bus Controller Default)
	4	2 Stoppbits

8.5.5 Parity_IF

Name:
phyParity
phyParity_CANIO

Mit Hilfe dieses Register wird die Art der Paritätsprüfung festgelegt. Die möglichen Werte sind ASCII-codiert.

Datentyp	Werte	Bedeutung
USINT	48	"0" - (Low) Bit immer 0
	49	"1" - (High) Bit immer 1
	69	"E" - (Even) Gerades Parity (Bus Controller Default)
	78	"N" - (No) Kein Bit
	79	"O" - (Odd) Ungerades Parity

8.6 Handshake - Konfiguration

Um den reibungslose Ablauf der seriellen Kommunikation zu gewährleisten, muss bekannt gegeben werden, wie groß der zu nutzende Empfangspuffer im Modul ist. Außerdem kann der Anwender einen soft- bzw. hardwareseitigen Handshake-Algorithmus vereinbaren.

8.6.1 Sperren des Empfangspuffers

Name:
rxlLock
rxlLock_CANIO

In diesem Register wird der obere Schwellwert des Empfangspuffer eingestellt.

Mit Hilfe der beiden Register "Lock"- bzw. "Unlock" kann die sogenannte "Flusskontrolle" zur Überwachung der Kommunikation genutzt werden. Überschreitet die Datenmenge im Eingang des Moduls den Wert des "Lock"-Registers, schaltet die Flusskontrolle in den Zustand "passiv". Um wieder in den Zustand "aktiv" bzw. "empfangsbereit" zu gelangen, muss die Datenmenge im Empfangspuffer unter den Vorgabewert des "Unlock"-Registers sinken.

Information:

Da durch diese Register das Verhalten eines Schmitt-Triggers nachgebildet wird, muss der Wert des "Lock"-Registers größer sein als der Wert des "Unlock"-Registers.

Datentyp	Werte	Bedeutung
UINT	0 bis 4095	Oberer Schwellwert des Empfangspuffers; Bus Controller Default: 1024

8.6.2 Entsperren des Empfangspuffers

Name:
rxlUnlock
rxlUnlock_CANIO

Mit diesem Register wird der untere Schwellwert des Empfangspuffers eingestellt.

Mit Hilfe der beiden Register "Lock"- bzw. "Unlock" kann die sogenannte "Flusskontrolle" zur Überwachung der Kommunikation genutzt werden. Überschreitet die Datenmenge im Eingang des Moduls den Wert des "Lock"-Registers, schaltet die Flusskontrolle in den Zustand "passiv". Um wieder in den Zustand "aktiv" bzw. "empfangsbereit" zu gelangen, muss die Datenmenge im Empfangspuffer unter den Vorgabewert des "Unlock"-Registers sinken.

Information:

Da durch diese Register das Verhalten eines Schmitt-Triggers nachgebildet wird, muss der Wert des "Lock"-Registers größer sein als der Wert des "Unlock"-Registers.

Datentyp	Werte	Bedeutung
UINT	0 bis 4095	Unterer Schwellwert des Empfangspuffers; Bus Controller Default: 512

8.6.3 RTS-Auswertung

Name:
hshRxF
hshRxF_CANIO

In diesem Register kann die Steuerung der Hardware-Handshake Leitung RTS in Abhängigkeit vom Füllstand des Empfangspuffers konfiguriert werden.

Mit Hilfe der beiden Register "TxF"- bzw "RxF"-Registers kann die Flusskontrolle für die Eingangs- bzw. Ausgangsrichtung aktiviert werden. Dabei wird die Kommunikation über einen Ringpuffer abgewickelt.

Information:

Es darf nur ein hsh-Register zur Steuerung der RTS-Leitung konfiguriert werden.

Datentyp	Werte	Bedeutung
USINT	0	RTS-Leitung frei verfügbar für andere Flusssteuerungsmethoden (Bus Controller Default)
	16	RTS-Leitung wird vom Füllstand des Empfangspuffers gesteuert

8.6.4 CTS-Auswertung

Name:
hshTxF
hshTxF_CANIO

In diesem Register wird die Auswertung der HW-Handshake Leitung CTS konfiguriert. Eine korrekte Verdrahtung zur Gegenstation bei aktiver CTS-Abfrage ist zu beachten.

Mit Hilfe der beiden Register "TxF"- bzw "RxF"-Registers kann die Flusskontrolle für die Eingangs- bzw. Ausgangsrichtung aktiviert werden. Dabei wird die Kommunikation über einen Ringpuffer abgewickelt.

Datentyp	Werte	Bedeutung
USINT	0	CTS-Leitung wird ignoriert, es kann immer gesendet werden (Bus Controller Default)
	1	CTS-Leitung aktiv und wird zur Flusssteuerung verwendet, Sendefreigabe von der Gegenstation

8.6.5 Software Handshake starten

Name:
hssXOn
hssXOn_CANIO

In diesem Register kann das XOn-Zeichen eingestellt werden. Standard ist der Wert 17, es kann aber jeder andere Wert konfiguriert werden.

Mit Hilfe der beiden Register "Xon" und "Xoff" kann ein softwareseitiger Handshake für die Flusssteuerung initiiert werden. Dazu muss in beiden Registern ein gültiges ASCII-Zeichen konfiguriert werden.

Datentyp	Werte	Bedeutung
UINT	0 bis 255	XOn ASCII-Zeichen
	65535	Kein Software Handshake (Bus Controller Default)

8.6.6 Software Handshake beenden

Name:
hssXOff
hssXOff_CANIO

In diesem Register kann das XOff-Zeichen eingestellt werden. Standard ist der Wert 19, es kann aber jeder andere Wert konfiguriert werden.

Mit Hilfe der beiden Register "Xon" und "Xoff" kann ein softwareseitiger Handshake für die Flusssteuerung initiiert werden. Dazu muss in beiden Registern ein gültiges ASCII-Zeichen konfiguriert werden.

Datentyp	Werte	Bedeutung
UINT	0 bis 255	XOff ASCII-Zeichen
	65535	Kein Software Handshake (Bus Controller Default)

8.6.7 Wiederholung des Handshakes

Name:
hssPeriod
hssPeriod_CANIO

Einige Anwendungen verlangen bei softwareseitigen Handshakes eine periodische Wiederholung des aktuellen Status. Zu diesem Zweck kann im diesem Register die Wiederholzeit in ms vorgegeben werden.

Datentyp	Werte	Bedeutung
UINT	0	Automatische Statuswiederholung deaktiviert
	500 bis 10000	Wiederholzeit in ms. Bus Controller Default: 5000

8.6.8 Handshake manuell aktivieren

Name:
hshSet
hshSet_CANIO

Mit Hilfe der beiden Register "Set" und "Clr" kann der Handshake über die Anwendung manuell verwaltet werden.

Durch dieses Register kann der Ausgangspegel der Hardware-Handshake-Leitung RTS zwingend auf aktiv gehalten werden.

Information:

Es darf nur ein hsh-Register zur Steuerung der RTS-Leitung konfiguriert werden.

Datentyp	Werte	Bedeutung
USINT	0	RTS-Leitung frei verfügbar für andere Flusssteuerungsmethoden (Bus Controller Default)
	16	RTS-Leitung ist aktiviert

8.6.9 Handshake manuell deaktivieren

Name:
hshClr
hshClr_CANIO

Mit Hilfe der beiden Register "Set" und "Clr" kann der Handshake über die Anwendung manuell verwaltet werden.

Durch dieses Register kann der Ausgangspegel der Hardware-Handshake-Leitung RTS zwingend auf passiv gehalten werden.

Information:

Es darf nur ein hsh-Register zur Steuerung der RTS-Leitung konfiguriert werden.

Datentyp	Werte	Bedeutung
USINT	0	RTS-Leitung frei verfügbar für andere Flusssteuerungsmethoden (Bus Controller Default)
	16	RTS-Leitung ist deaktiviert

8.6.10 Frameerkennung

Name:
hshFrm
hshFrm_CANIO

Mit diesem Register wird die hardwareseitige Frameerkennung generell aktiviert. Die RTS-Leitung ist aktiv, solange Daten gesendet werden. Dieser Tx-Framing-Modus kann zu Steuerung von externen Schnittstellenumsetzern verwendet werden.

Information:

Es darf nur ein hsh-Register zur Steuerung der RTS-Leitung konfiguriert werden.

Datentyp	Werte	Bedeutung
USINT	0	RTS-Leitung frei verfügbar für andere Flusssteuerungsmethoden (Bus Controller Default)
	16	RTS-Leitung Tx-Framing eingeschaltet
	80	RTS-Leitung Tx-Framing eingeschaltet (ohne Echo)

8.6.11 Invertieren von RTS/CTS

Name:
hshInv
hshInv_CANIO

Mit Hilfe dieses Registers können die Signale der RTS bzw. CTS logisch invertiert werden.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	0

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	CTS-Signal	0	Invertierung aus (Bus Controller Default)
		1	Invertierung ein
1 - 3	Reserviert	0	
4	RTS-Signal	0	Invertierung aus (Bus Controller Default)
		1	Invertierung ein
5 - 7	Reserviert	0	

8.7 Frame - Konfiguration

Um die gesendeten Tx-Frames korrekt zu bilden und die empfangenen Rx-Frames richtig zu interpretieren, können unterschiedliche Nachrichen-Endekennungen festgelegt werden.

8.7.1 Beenden bei Empfangs-Zeitüberschreitung

Name:
rxCto
rxCto_CANIO

In diesem Register wird die Zeitdauer bis zum Auslösen einer Empfangs-Zeitüberschreitung eingestellt.

Die Nachricht gilt als beendet, wenn für die vereinbarte Dauer keine Übertragung stattfindet.
Die Zeitangabe wird dabei in Zeichen angegeben, um von der Übertragungsrate unabhängig zu sein. Dafür wird die Anzahl der Zeichen mit der Zeitspanne multipliziert, die zur Übertragung eines Zeichens benötigt wird.

Datentyp	Werte	Bedeutung
UINT	0	Funktion deaktiviert
	1 bis 65535	Empfangs-Zeitüberschreitung in Zeichen; Bus Controller Default: 4

8.7.2 Beenden bei Sende-Zeitüberschreitung

Name:
txCto
txCto_CANIO

In diesem Register wird die Zeitdauer bis zum Auslösen einer Sende-Zeitüberschreitung eingestellt.

Die Nachricht gilt als beendet, wenn für die vereinbarte Dauer keine Übertragung stattfindet.
Die Zeitangabe wird dabei in Zeichen angegeben, um von der Übertragungsrate unabhängig zu sein. Dafür wird die Anzahl der Zeichen mit der Zeitspanne multipliziert, die zur Übertragung eines Zeichens benötigt wird.

Datentyp	Werte	Bedeutung
UINT	0	Funktion deaktiviert
	1 bis 65535	Sende-Zeitüberschreitung in Zeichen; Bus Controller Default: 5

8.7.3 Maximale Empfangs-Byteanzahl

Name:
rxEomSize
rxEomSize_CANIO

Mit diesem Register wird die maximale Byteanzahl des Empfangsframes konfiguriert.

Die Nachricht gilt als beendet, sobald ein Frame der eingestellten Größe in Bytes übertragen wurde. Die größte mögliche Framelänge entspricht dem Empfangspuffer von 4096 Bytes. Größere Frames führen zum Fehler Receive Overrun.

Datentyp	Werte	Bedeutung
UINT	0	Funktion deaktiviert
	1 bis 4096	Konfigurierbare Empfangsframelänge in Zeichen; Bus Controller Default: 256

8.7.4 Maximale Sende-Byteanzahl

Name:
txEomSize
txEomSize_CANIO

Mit diesem Register wird die maximale Byteanzahl des Sendeframes konfiguriert.

Die Nachricht gilt als beendet, sobald ein Frame der eingestellten Größe in Bytes übertragen wurde. Die größte mögliche Framelänge entspricht dem Sendepuffer von 4096 Bytes. Nach Senden des Frames wird die konfigurierte Sendepause eingehalten.

Datentyp	Werte	Bedeutung
UINT	0	Funktion deaktiviert
	1 bis 4096	Konfigurierbare Sendeframelänge in Zeichen; Bus Controller Default: 4096

8.7.5 Empfangsabschlusszeichen definieren

Name:
rxEomChar0 bis rxEomChar3
rxEomChar0_CANIO bis rxEomChar3_CANIO

In jedem Registern kann ein mögliches Empfangsabschlusszeichen konfiguriert werden.

Die Nachricht gilt als beendet, sobald eines der definierten Zeichen übertragen wird.

Datentyp	Werte	Bedeutung
UINT	0 bis 255	Abschlusszeichen des Frames ASCII Code
	65535	Funktion deaktiviert (Bus Controller Default)

8.7.6 Sendeabschlusszeichen definieren

Name:
txEomChar0 bis txEomChar3
txEomChar0_CANIO bis txEomChar3_CANIO

In jedem Registern kann ein mögliches Sendeabschlusszeichen konfiguriert werden.

Die Nachricht gilt als beendet, sobald eines der definierten Zeichen übertragen wird.

Datentyp	Werte	Bedeutung
UINT	0 bis 255	Abschlusszeichen des Frames ASCII Code
	65535	Funktion deaktiviert (Bus Controller Default)

8.8 Statusmeldungen - Konfiguration

Mit Hilfe der Statusmeldungen erhält der Anwender Informationen über die aktuelle Situation im nachgelagerten seriellen Netzwerk.

8.8.1 Fehlererkennung einstellen

Name:
CfO_RxStateIgnoreMask
CfO_RxStateIgnoreMask_CANIO

Dieses Register wirkt sich direkt auf die Arbeitsweise des UART aus. Mit Hilfe des Low Bytes kann die Fehlererkennung grundsätzlich deaktiviert werden. Falls die Fehlererkennung nicht deaktiviert wurde, kann über das High Byte eingestellt werden, dass ein erkannter Fehler als Ende der Nachricht interpretiert werden soll.

Datentyp	Werte	Bus Controller Default
UINT	Siehe Bitstruktur	0

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0 - 3	Reserviert	0	
4	StartBitError	0	Fehlerhaftes Startbit erkennen (Bus Controller Default)
		1	Ingnorieren
5	StopBitError	0	Fehlerhaftes Stoppbit erkennen (Bus Controller Default)
		1	Ingnorieren
6	ParityError	0	Fehlerhaftes Paritätsbit erkennen (Bus Controller Default)
		1	Ingnorieren
7	RXoverrun	0	Überlauf in Empfangsrichtung erkennen (Bus Controller Default)
		1	Ingnorieren
8 - 11	Reserviert	0	
12	StartBitError entspricht Frame-Ende (wenn Bit 4 = 0)	0	Fehler nur modulintern anzeigen (Bus Controller Default)
		1	Zusätzlich Frame-Ende signalisieren
13	StopBitError entspricht Frame-Ende (wenn Bit 5 = 0)	0	Fehler nur modulintern anzeigen (Bus Controller Default)
		1	Zusätzlich Frame-Ende signalisieren
14	ParityError entspricht Frame-Ende (wenn Bit 6 = 0)	0	Fehler nur modulintern anzeigen (Bus Controller Default)
		1	Zusätzlich Frame-Ende signalisieren
15	RXoverrun entspricht Frame-Ende (wenn Bit 7 = 0)	0	Fehler nur modulintern anzeigen (Bus Controller Default)
		1	Zusätzlich Frame-Ende signalisieren

8.8.2 Fehler an Anwendung weiterreichen

Name:
CfO_ErrorID0007

In diesem Register kann eingestellt werden, welche Fehlermeldungen an die Anwendung weitergereicht werden.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	0

Bitstruktur:

Bit	Bezeichnung		Information
0	StartBitError	0	Ignorieren (Bus Controller Default)
		1	Fehlerhaftes Startbit anzeigen
1	StopBitError	0	Ignorieren (Bus Controller Default)
		1	Fehlerhaftes Stoppbit anzeigen
2	ParityError	0	Ignorieren (Bus Controller Default)
		1	Fehlerhaftes Paritätsbit anzeigen
3	RXoverrun	0	Ignorieren (Bus Controller Default)
		1	Überlauf in Empfangsrichtung anzeigen
4 - 7	Reserviert	0	

8.9 Statusmeldungen - Kommunikation

Nach der Konfiguration können bis zu vier Statusmeldungen in der Anwendung ausgewertet werden.

8.9.1 Statusbits Fehlermeldungen

Name:
StartBitError
StopBitError
ParityError
RXoverrun

Mit Hilfe dieses Registers werden die Einzelbits übertragen, die einen Fehler anzeigen. Tritt einer der Fehler auf, so wird das entsprechende Bit gesetzt und gehalten bis eine Quittierung erfolgt.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	StartBitError	0	Kein Fehler
		1	Startbit-Fehler aufgetreten[1)]
1	StopBitError	0	Kein Fehler
		1	Stopbit-Fehler aufgetreten[1)]
2	ParityError	0	Kein Fehler
		1	Paritybit-Fehler aufgetreten[1)]
3	RXoverrun	0	Kein Fehler
		1	Empfangspufferüberlauf aufgetreten[2)]
4 - 7	Reserviert	0	

1) Dieser Fehler kann z. B. durch nicht zusammen passende Schnittstellenkonfigurationen oder Probleme mit der Verkabelung entstehen.
2) Mit diesem Datenpunkt wird ein Empfangspufferüberlauf gemeldet. Die Pufferkapazität am Modul ist ausgeschöpft und alle nachfolgenden Daten an der Schnittstelle gehen verloren. Ein Überlauf bedeutet immer, dass die am Modul empfangenen Daten nicht schnell genug vom übergeordnetem System ausgelesen werden.
Abhilfe kann hier getroffen werden durch eine Zykluszeitoptimierung aller beteiligten Übertragungsstrecken bzw. Taskklassen und die Verwendung der vorhandenen Handshake Möglichkeiten.

8.9.2 Quittieren der Statusbits

Name:
QuitStartBitError
QuitStopBitError
QuitParityError
QuitRXoverrun

Mit Hilfe dieses Registers werden die Einzelbits übertragen, die einen angezeigten Fehlerzustand quittieren. Nachdem eines der Fehlerbits gesetzt wurde, kann es über das entsprechende Quittierungsbit zurückgesetzt werden. Ist der Fehler noch aktiv anstehend, wird das Fehlerstatusbit nicht gelöscht. Das Quittierungsbit kann erst rückgesetzt werden, wenn das Fehlerstatusbit nicht mehr gesetzt ist.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	QuitStartBitError	0	Keine Quittierung
		1	Startbit-Fehler quittieren
1	QuitStopBitError	0	Keine Quittierung
		1	Stopbit-Fehler quittieren
2	QuitParityError	0	Keine Quittierung
		1	Paritybit-Fehler quittieren
3	QuitRXoverrun	0	Keine Quittierung
		1	Empfangspufferüberlauf Fehler quittieren
4 - 7	Reserviert	0	

8.10 Die Flatstream-Kommunikation

8.10.1 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.

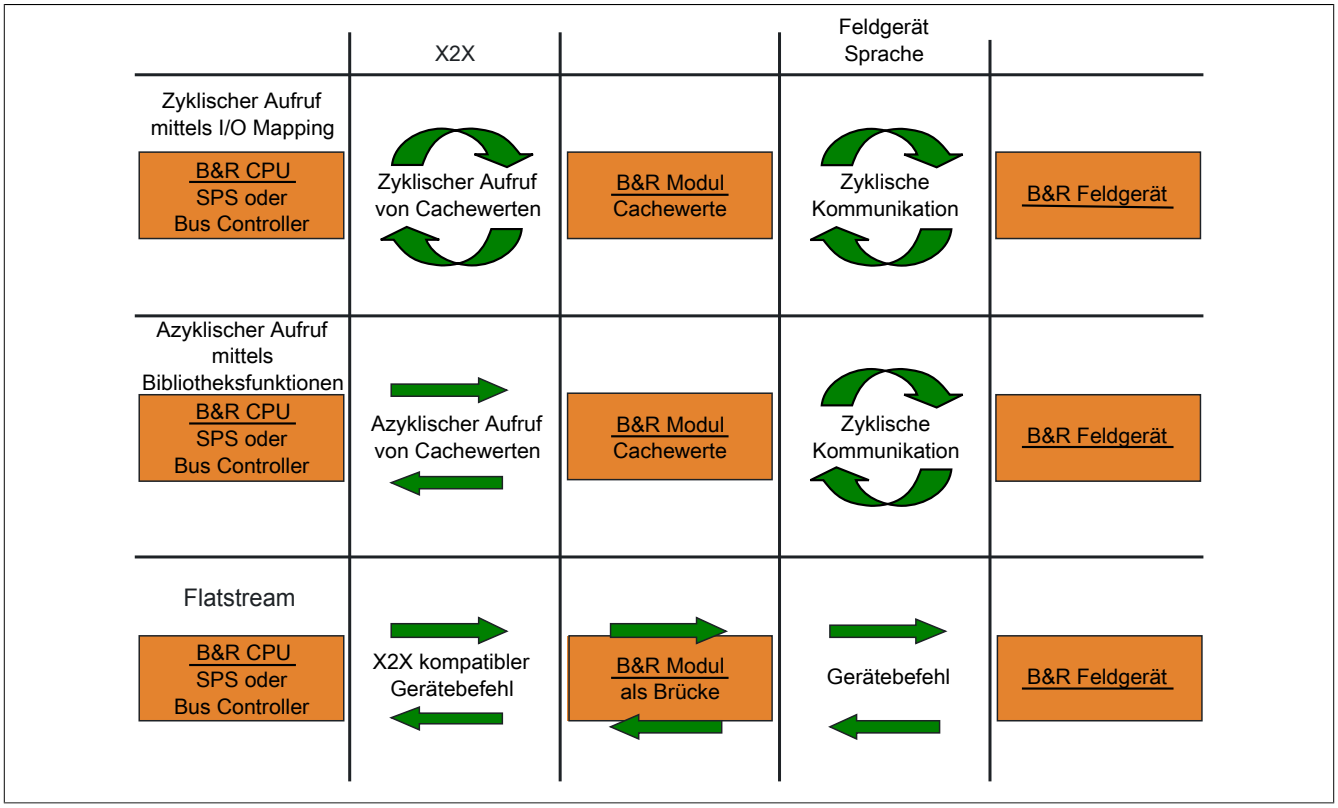


Abbildung 1: 3 Arten der Kommunikation

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.

8.10.2 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.

8.10.3 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.

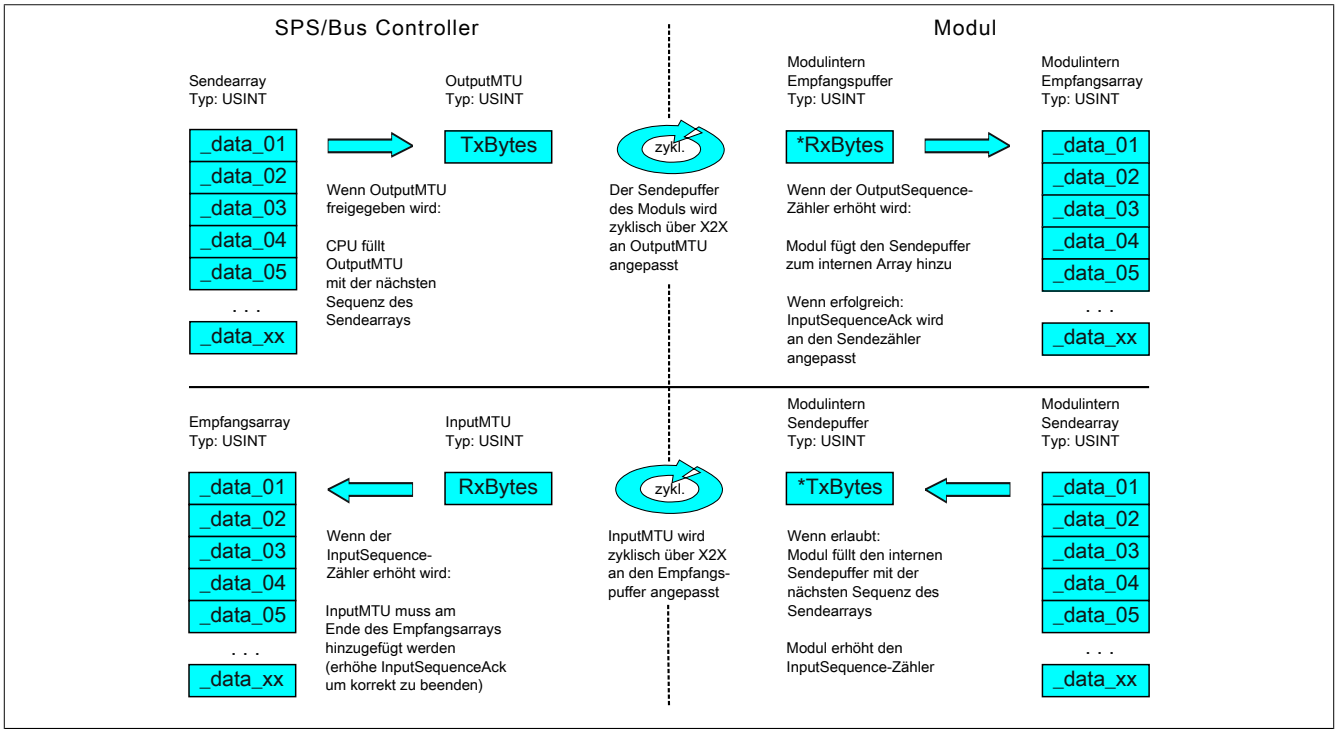


Abbildung 2: Kommunikation per Flatstream

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.

8.10.4 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.

8.10.4.1 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.

8.10.4.1.1 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)

8.10.4.2 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.

8.10.4.2.1 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

8.10.4.2.2 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

8.10.4.2.3 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

8.10.4.2.4 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.

8.10.4.2.5 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.

8.10.4.2.6 Beziehung zwischen Output- und InputSequence

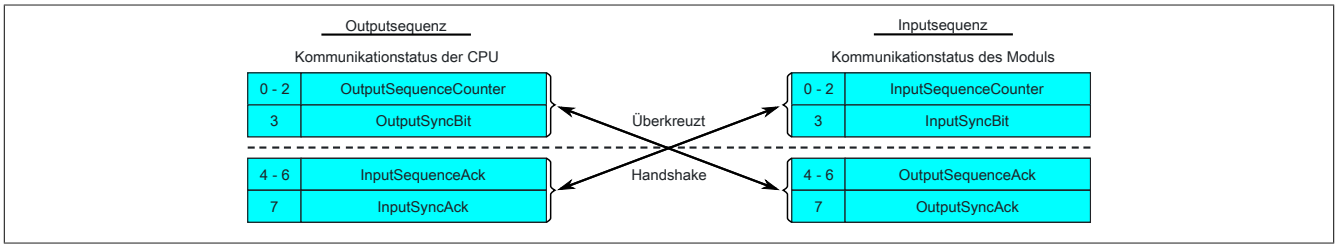


Abbildung 3: Zusammenhang zwischen Output- und InputSequence

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.

8.10.4.3 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.

8.10.4.4 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.

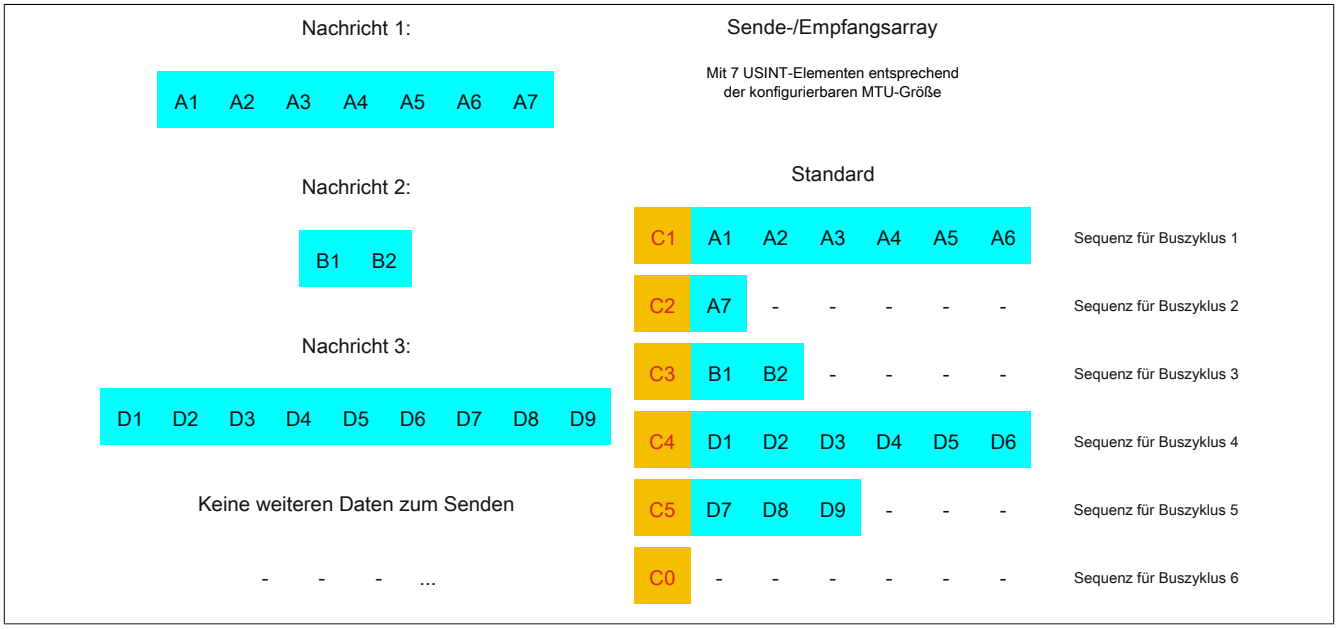


Abbildung 4: Sende-/Empfangsarray (Standard)

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)

8.10.4.5 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.

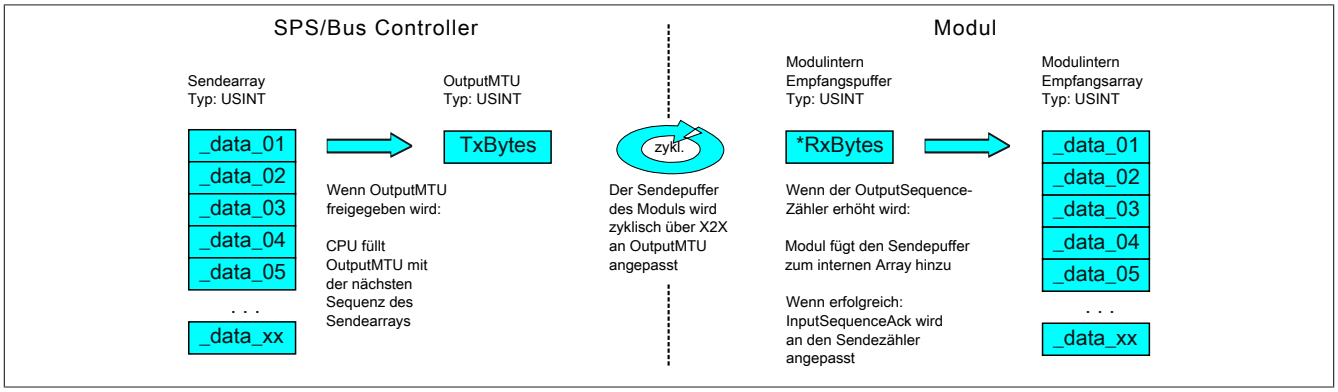


Abbildung 5: Kommunikation per Flatstream (Output)

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

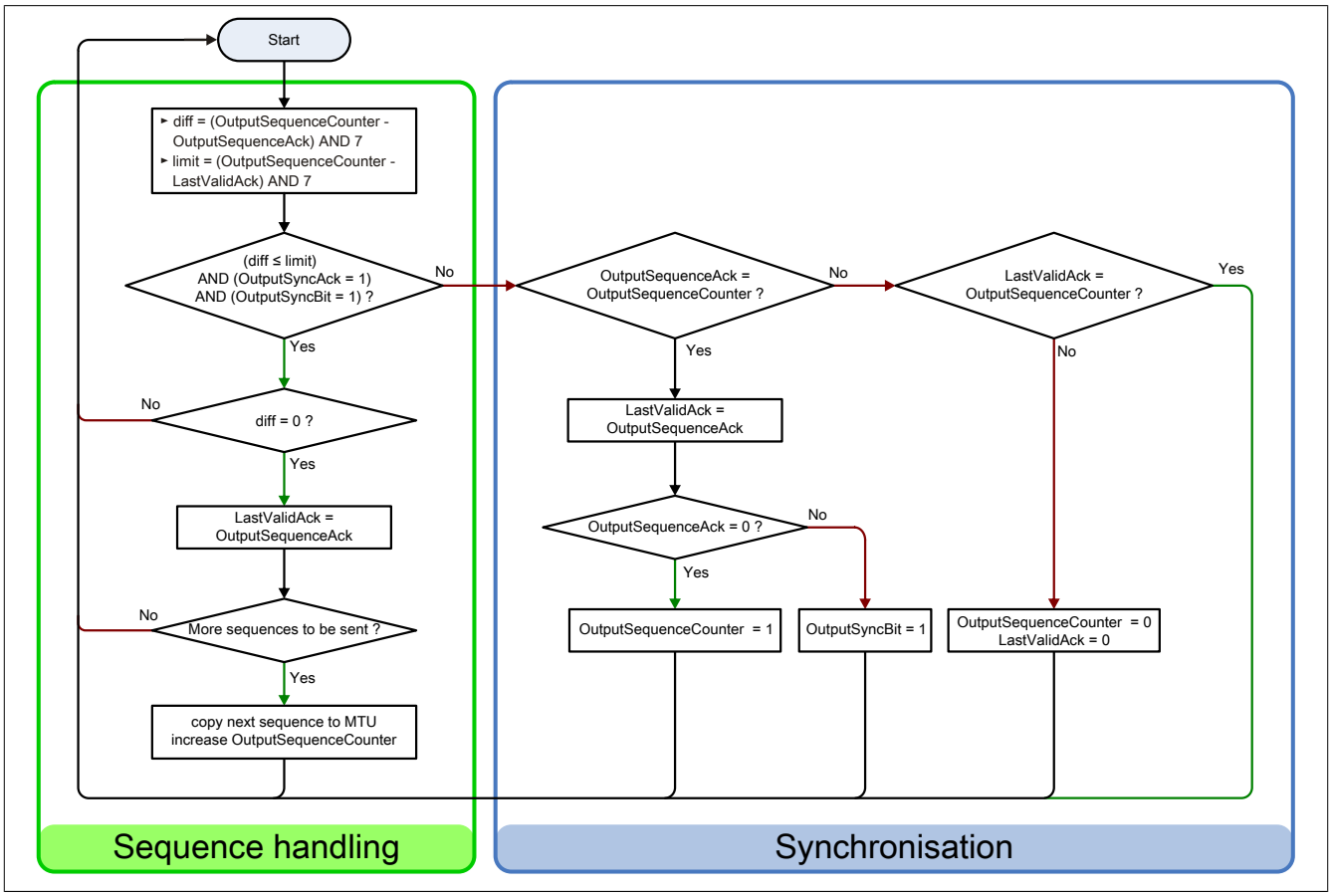


Abbildung 6: Ablaufdiagramm für Output-Richtung

8.10.4.6 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.

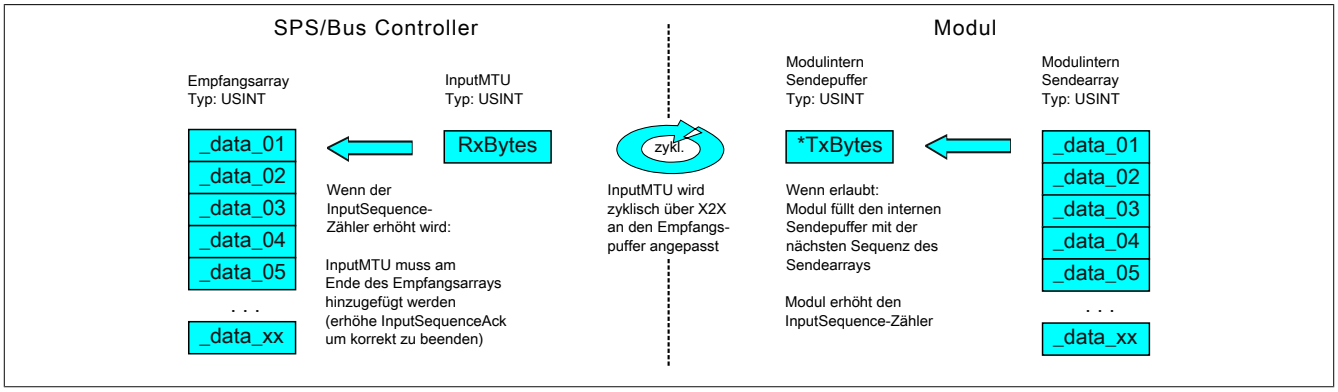


Abbildung 7: Kommunikation per Flatstream (Input)

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

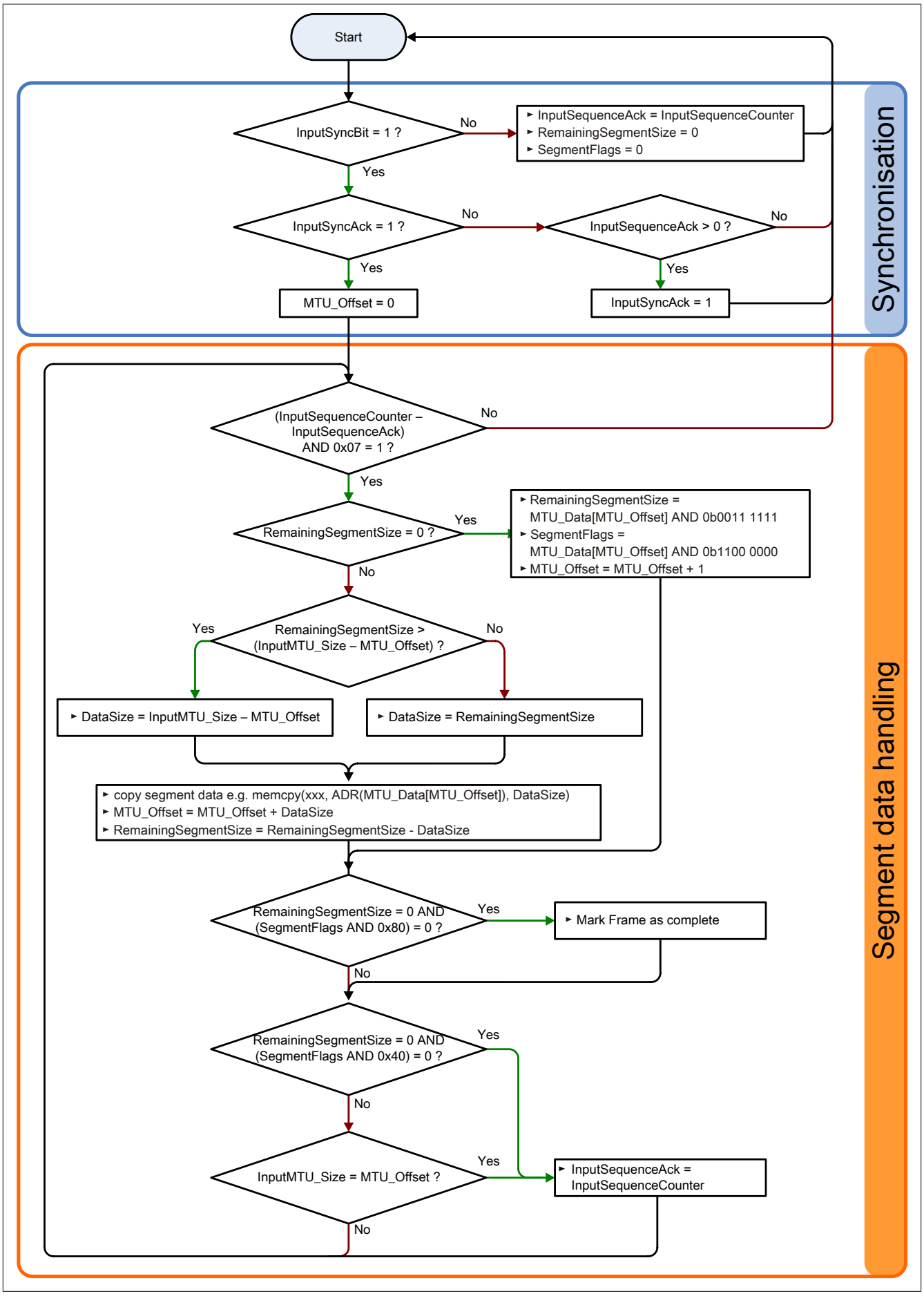


Abbildung 8: Ablaufdiagramm für Input-Richtung

8.10.4.7 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.

8.10.4.8 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.

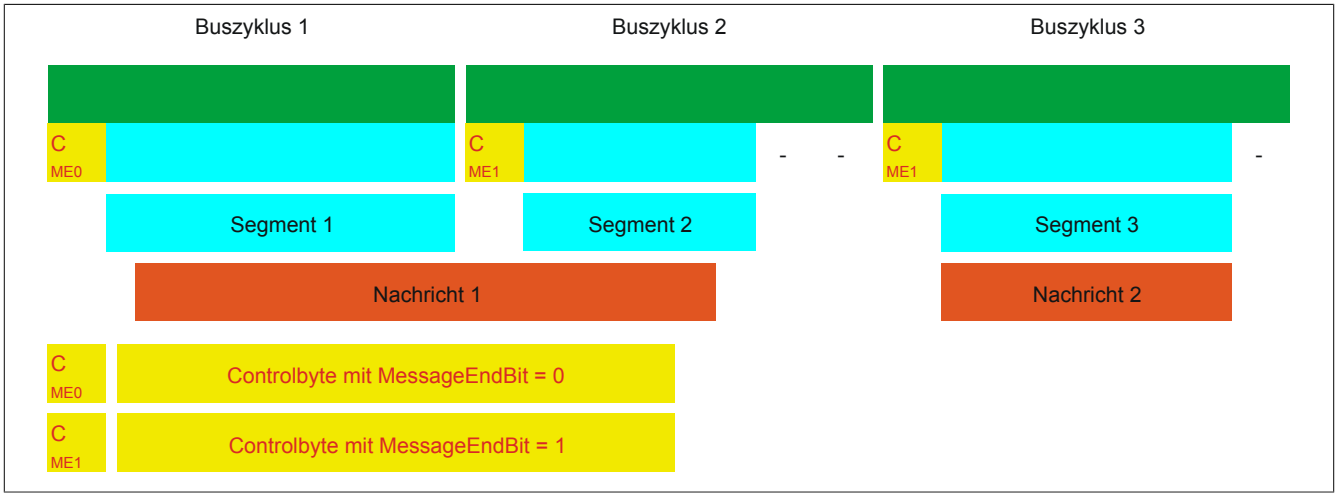


Abbildung 9: Anordnung von Nachrichten in der MTU (Standard)

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.

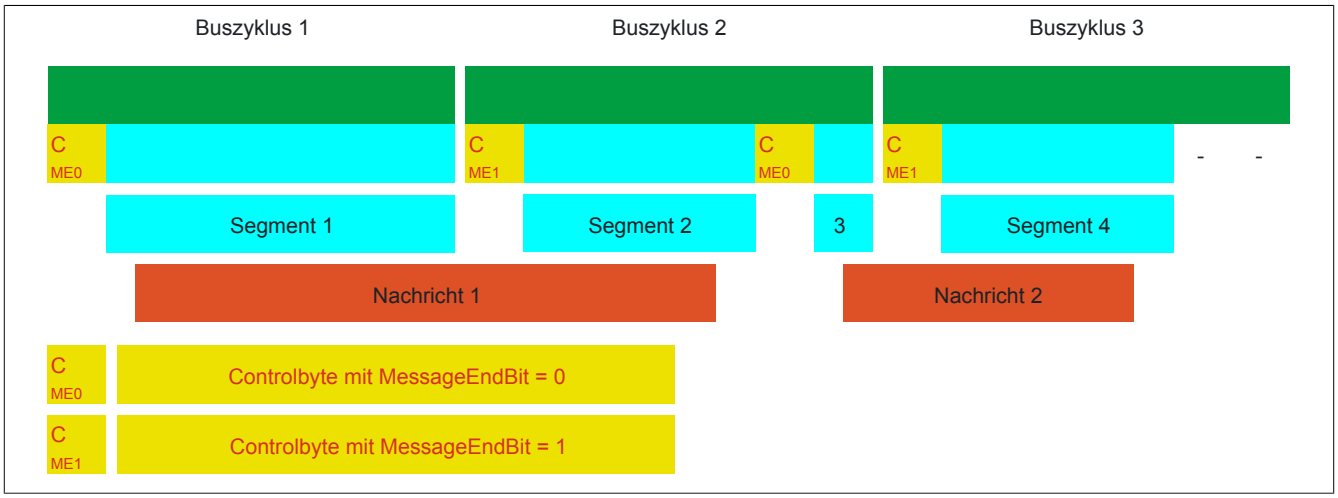


Abbildung 10: Anordnung von Nachrichten in der MTU (MultiSegmentMTU)

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.

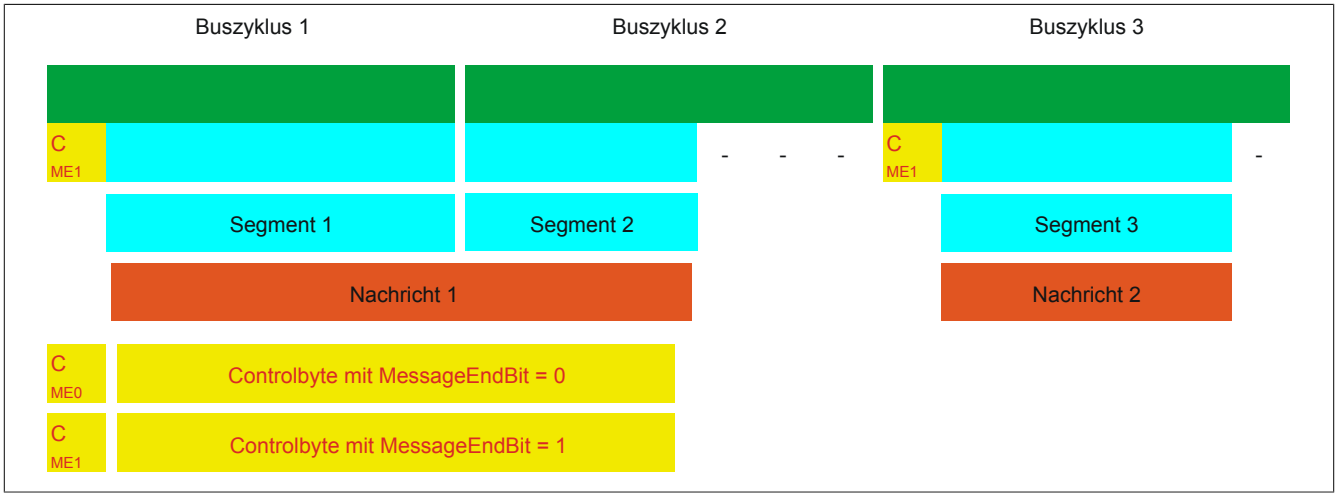


Abbildung 11: Anordnung von Nachrichten in der MTU (große Segmente)

Anwendung beider Optionen

Die beiden Optionen dürfen auch gleichzeitig angewendet werden.

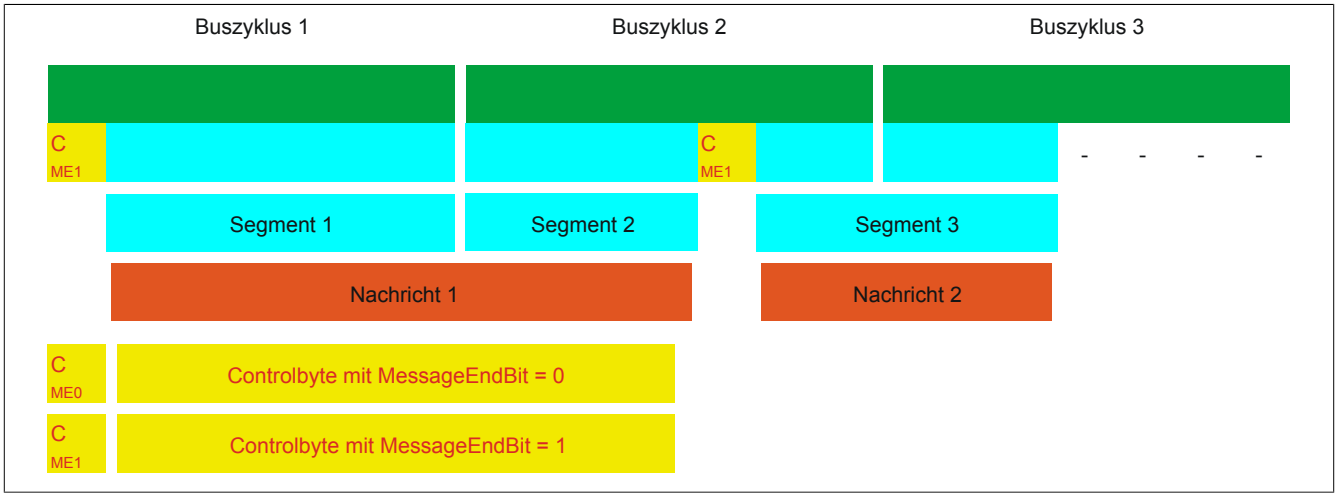


Abbildung 12: Anordnung von Nachrichten in der MTU (große Segmente und MultiSegmentMTU)

8.10.4.9 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.

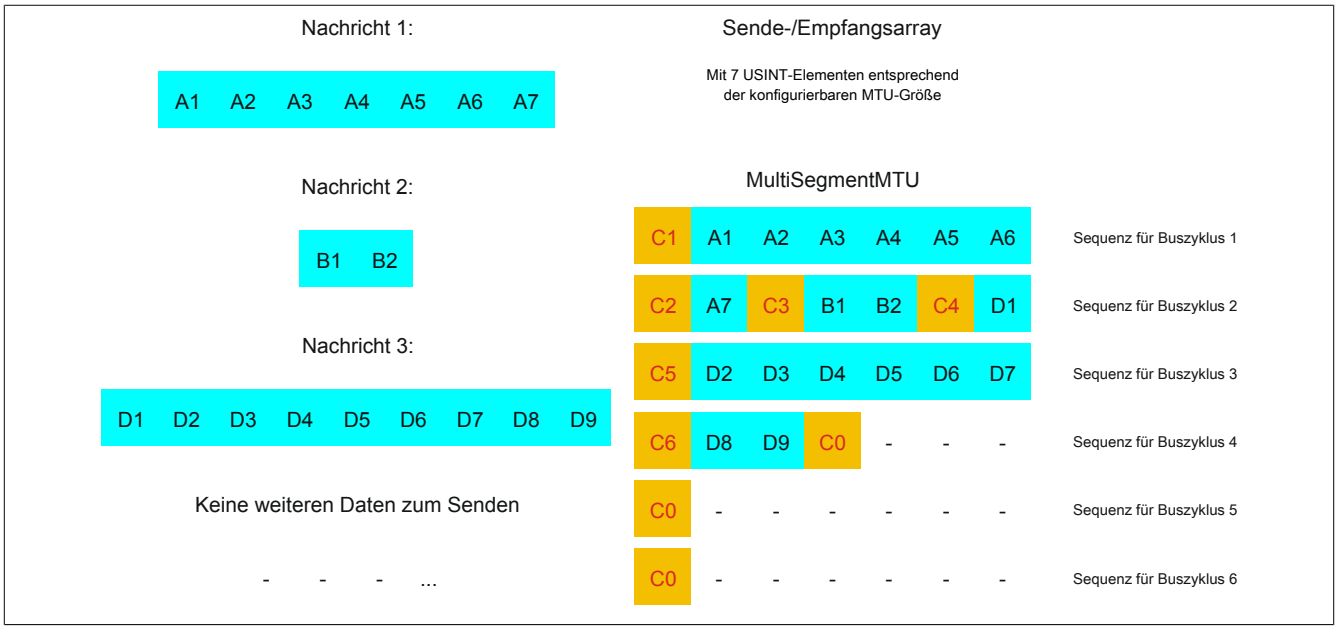


Abbildung 13: Sende-/Empfangsarray (MultiSegmentMTU)

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.

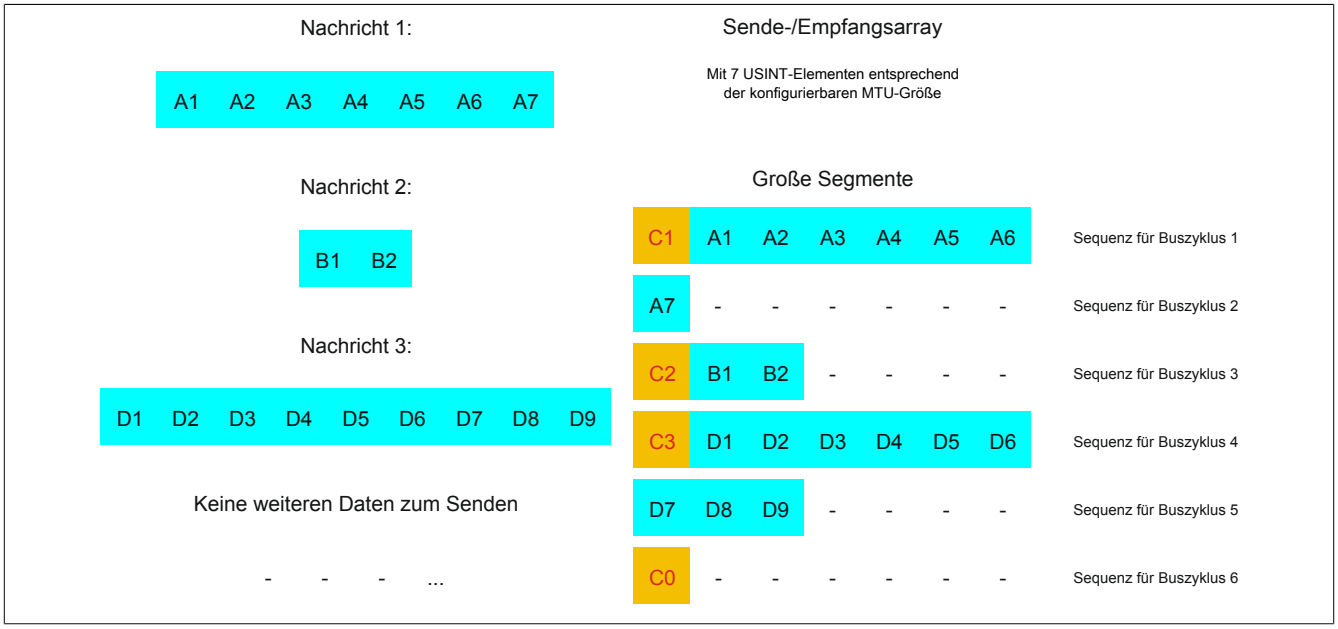


Abbildung 14: Sende-/Empfangsarray (große Segmente)

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.

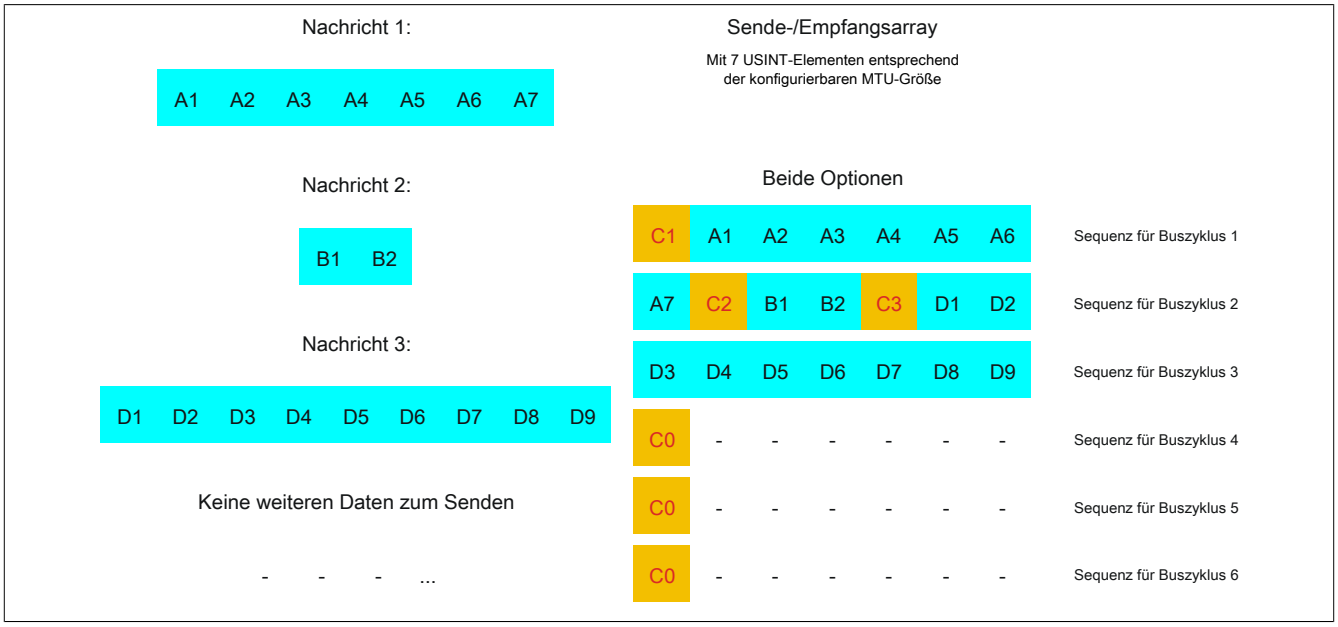


Abbildung 15: Sende-/Empfangsarray (große Segmente und MultiSegmentMTU)

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

8.10.5 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.

8.10.5.1 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)

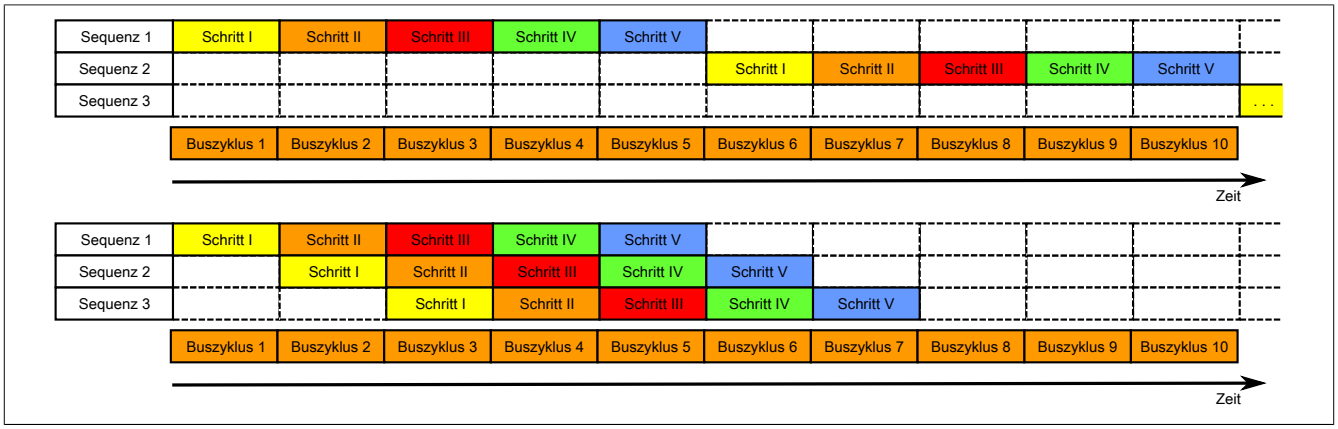


Abbildung 16: Vergleich Übertragung ohne bzw. mit Forward

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.

8.10.5.2 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.

8.10.5.2.1 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

8.10.5.2.2 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

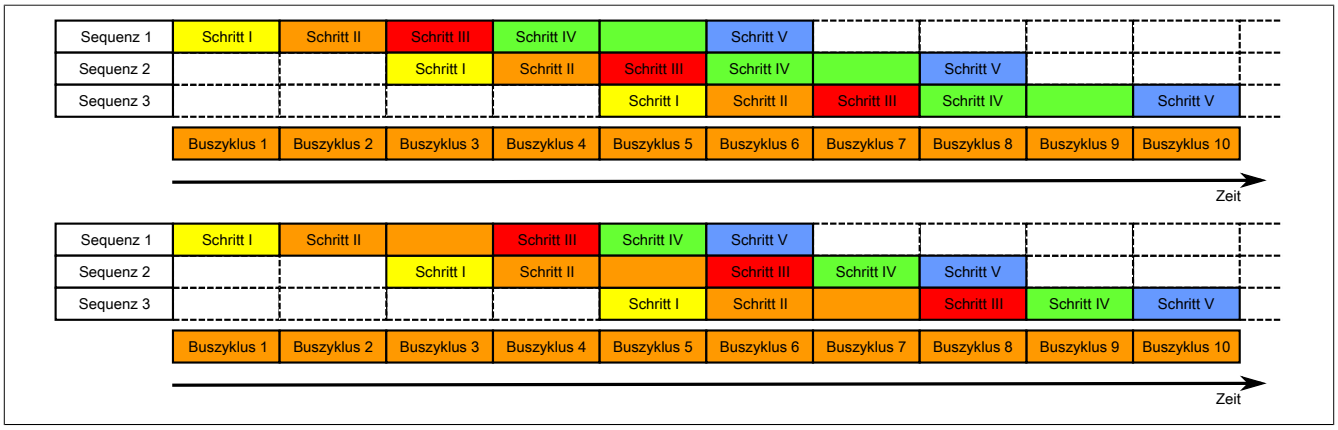


Abbildung 17: Auswirkung des ForwardDelay bei der Flatstream-Kommunikation mit Forward

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.

8.10.5.3 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.

8.10.5.4 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).

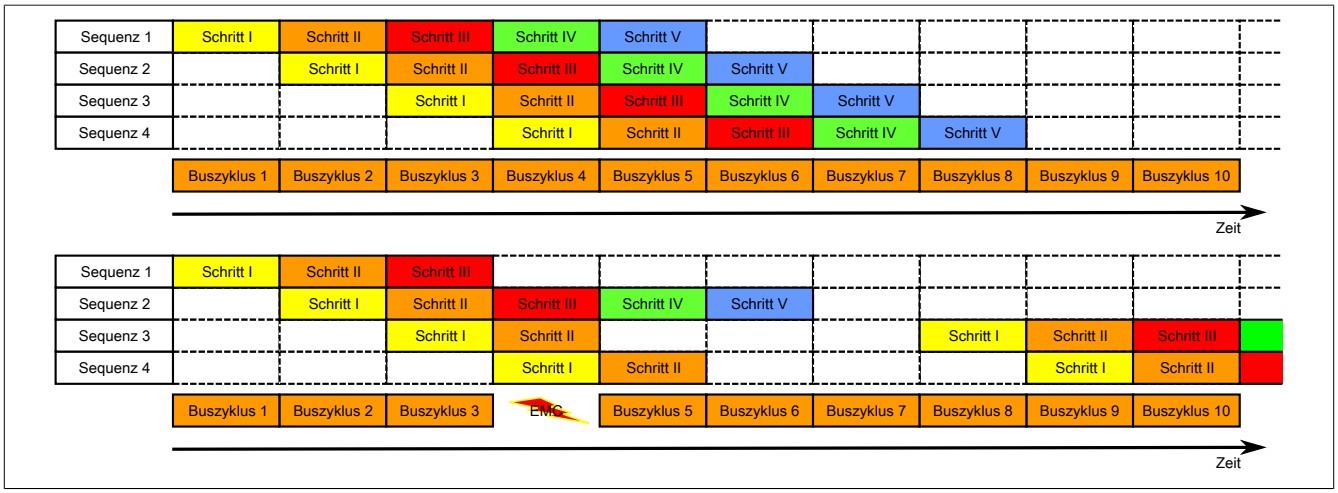


Abbildung 18: Auswirkung eines ausgefallenen Buszyklus

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.

8.11 Serial on Flatstream

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

Im seriellen Netzwerk tritt das Modul stets als Master (DTE) auf. Um eine fehlerfreie Signalübertragung zu gewährleisten, können verschiedene Anpassungen vorgenommen werden.
Der Anwender kann z. B. einen Handshake-Algorithmus definieren oder die Baudrate einstellen, um die Qualität der Übertragung an die Belange der Anwendung anzupassen.

Handhabung

Bei der Nutzung des Flatstreams muss die generelle Struktur des Flatstream-Frames eingehalten werden.

Ein-/Ausgangs-Sequenz	Tx/Rx-Bytes	
(unverändert)	Controlbyte(unverändert)	Serielles-Frame (ohne Handshake o.Ä.)

8.12 Azyklische Framegröße

Name:
AsynSize

Bei Verwendung des Streams werden die Daten intern zwischen Modul und CPU ausgetauscht. Zu diesem Zweck wird eine definierte Anzahl an azyklischen Bytes für diesen Steckplatz reserviert.
Die Erhöhung der azyklischen Framegröße führt zu einem gesteigerten Datendurchsatz auf diesem Steckplatz.

Information:

Es handelt sich bei dieser Konfiguration um eine Treibereinstellung, welche während der Laufzeit nicht verändert werden kann!

Datentyp	Werte	Information
-	8 bis 28	Azyklische Framegröße in Byte. Default = 24

8.13 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

8.14 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
200 µs