X20CM4800X

1 Bestelldaten

Bestellnummer	Kurzbeschreibung	Abbildung
	Sonstige Funktionen	
X20CM4800X	X20 Analoges Eingangsmodul, Schwingungsmessung, 4 IEPE-Analogeingänge, 50 kHz Abtastfrequenz, 24 Bit Wandlerauflösung	
	Erforderliches Zubehör	
	Busmodule	
X20BM11	X20 Busmodul, 24 VDC codiert, interne I/O-Versorgung durchverbunden	
	Feldklemmen	
X20TB12	X20 Feldklemme, 12-polig, 24 VDC codiert	
	Optionales Zubehör	
	Sensoren	
0ACS100A.00-1	Beschleunigungssensor, nominale Empfindlichkeit 100 mV/g, Ausgang oben	
0ACS100A.90-1	Beschleunigungssensor, nominale Empfindlichkeit 100 mV/g, Ausgang seitlich	
	Sensorkabel	
0ACC0020.01-1	Kabel für Beschleunigungssensor, Länge 2 m, 2x 0,34mm², M12 Buchse sensorseitig, schleppkettentauglich, UL zugelassen	
0ACC0050.01-1	Kabel für Beschleunigungssensor, Länge 5 m, 2x 0,34 mm², M12 Buchse sensorseitig, schleppkettentauglich, UL zugelassen	
0ACC0100.01-1	Kabel für Beschleunigungssensor, Länge 10 m, 2x 0,34mm², M12 Buchse sensorseitig, schleppkettentauglich, UL zugelassen	
0ACC0150.01-1	Kabel für Beschleunigungssensor, Länge 15 m, 2x 0,34mm², M12 Buchse sensorseitig, schleppkettentauglich, UL zugelassen	
0ACC0200.01-1	Kabel für Beschleunigungssensor, Länge 20 m, 2x 0,34mm², M12 Buchse sensorseitig, schleppkettentauglich, UL zugelassen	
0ACC0500.01-1	Kabel für Beschleunigungssensor, Länge 50 m, 2x 0,34 mm², M12 Buchse sensorseitig, schleppkettentauglich, UL zugelassen	
0ACC1000.01-1	Kabel für Beschleunigungssensor, Länge 100 m, 2x 0,34 mm², M12 Buchse sensorseitig, schleppkettentauglich, UL zugelassen	

Tabelle 1: X20CM4800X - Bestelldaten

2 Modulbeschreibung

Das Modul ist für die Vibrationsmessung an Maschinen und Anlagen sowie der weiteren Auswertung der Daten auf der Steuerung vorgesehen. Das Modul verfügt über 4 Eingangskanäle, wobei die gewählte Abtastrate pro Eingang zwischen 200 Hz und 50 kHz eingestellt werden kann.

Funktionen:

- Konfiguration der Eingänge
- Vibrationen messen
- NetTime-Zeitstempel

Wird eine Messung gestartet, so werden die Vibrationen mit der eingestellten Abtastung erfasst und mit einer einstellbaren Datenauflösung von 16, 24 oder 32 Bit übertragen.

Beim Start der Messung wird ein NetTime-Zeitstempel erzeugt. Mit diesem kann jeder erfasste Messwert einer eindeutigen Zeit zugeordnet werden. Wird die Messung gestoppt, wird ein weiterer NetTime-Zeitstempel erzeugt.

3 Technische Daten

Bestellnummer	X20CM4800X
Kurzbeschreibung	
I/O-Modul	X20 4-Kanal analoges Eingangsmodul zur Vibrationsmessung von Condition Monitoring Aufgaben
Allgemeines	
Isolationsspannung zwischen Kanal und Bus	500 V_{eff}
Nennspannung	24 VDC ±20%
B&R ID-Code	0xF1C5
Statusanzeigen	Run, Error, Vibrationseingänge 1 bis 4
Diagnose	
Modul Run/Error	Ja, per Status-LED und SW-Status
Leistungsaufnahme	
Bus	0,01 W
I/O-intern	1,5 W
Zulassungen	
CE	Ja
EAC	Ja
Analoge Eingänge	
Anzahl	4
Eingangsart	IEPE-Sensor: Beschleunigung
Digitale Wandlerauflösung	24 Bit
Drahtbrucherkennung	
pro Kanal	Kleiner 8 V oder größer 14 V für mehr als 4 s
Zulässiges Eingangssignal	±10 VAC
Wandlungsverfahren	Sigma Delta
Typ	Vibrationseingang
Abtastfrequenz	Einstellbar von 200 Hz bis 50 kHz
Eingangshochpasseckfrequenz	34 mHz
Eingangstiefpasseckfrequenz	23 kHz
Sensorversorgung	IEPE, 5 mA Konstantstromquelle (4,9 bis 5,5 mA), pro Kanal abschaltbar
Elektrische Eigenschaften	
Potenzialtrennung	Kanal zu Bus getrennt Kanal zu Kanal 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 55°C
Derating	Siehe Abschnitt "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: X20CM4800X - Technische Daten

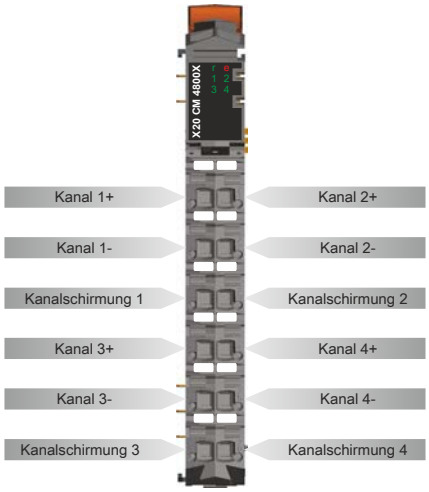
4 Status-LEDs

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 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
			Ein	Warnung, Fehler oder Reset-Status
			Single Flash	Drahtbruch am aktiven Kanal
			Double Flash	Pufferüberlauf
	e + r	Rot Grün	Ein Single Flash	Firmware ungültig
	1 - 4	Grün	Aus	Kanal ist inaktiv
			Ein	Kanal ist aktiv

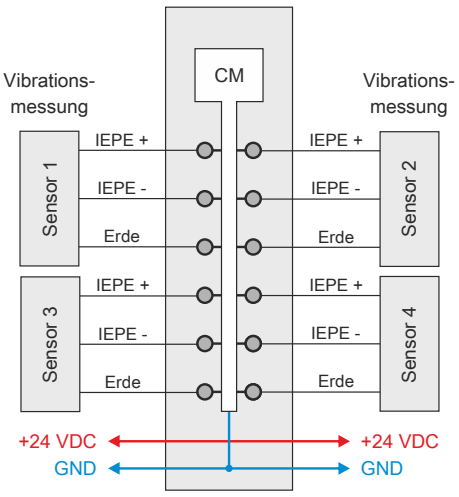
1) Je nach Konfiguration kann ein Firmware-Update bis zu mehreren Minuten benötigen.

5 Anschlussbelegung

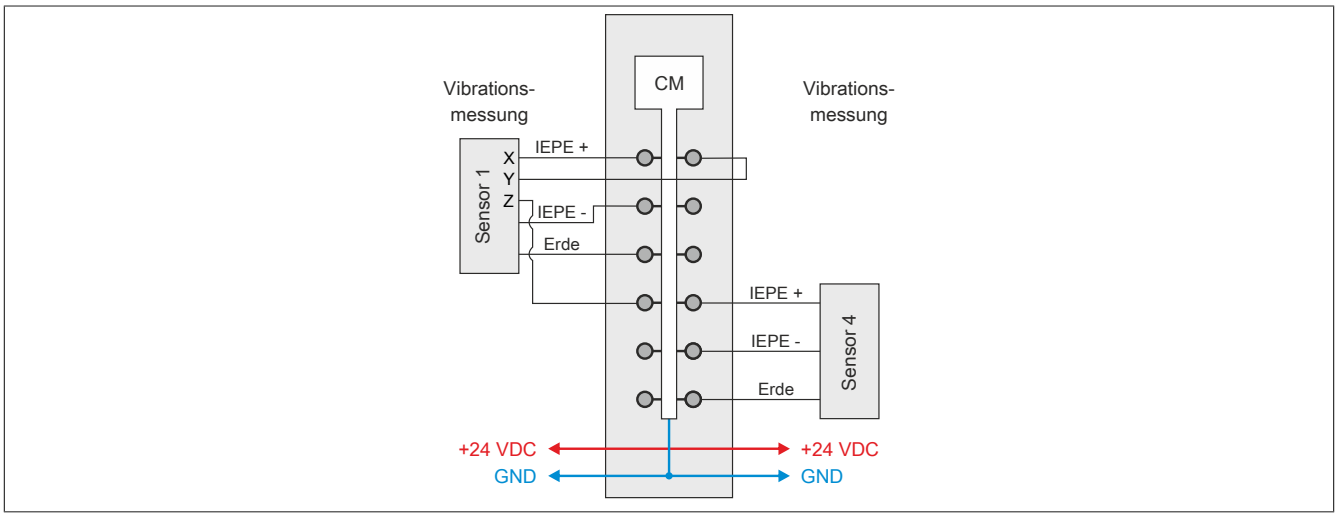


6 Anschlussbeispiel

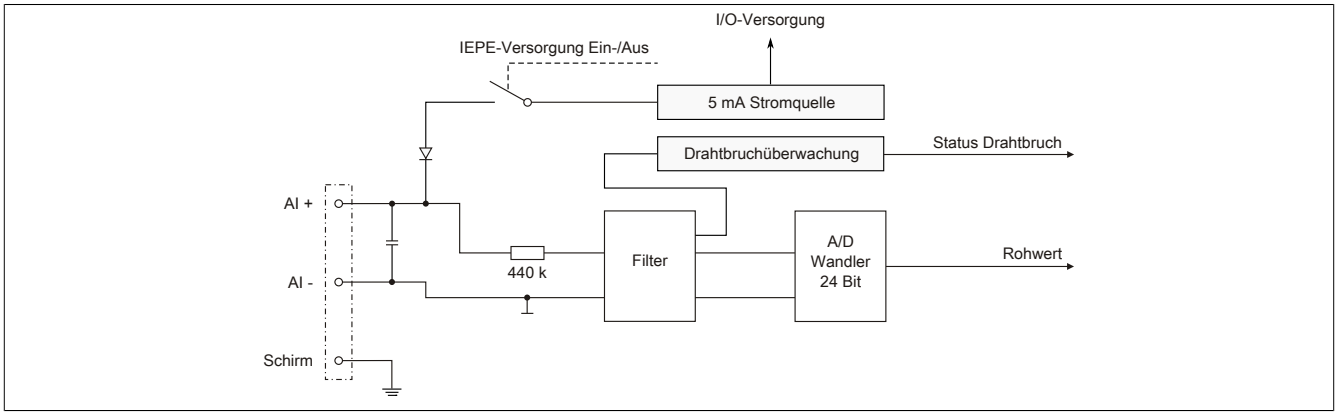
Anschluss von 1-Achsen-Sensoren



Anschluss von 1- und 3-Achsen-Sensor

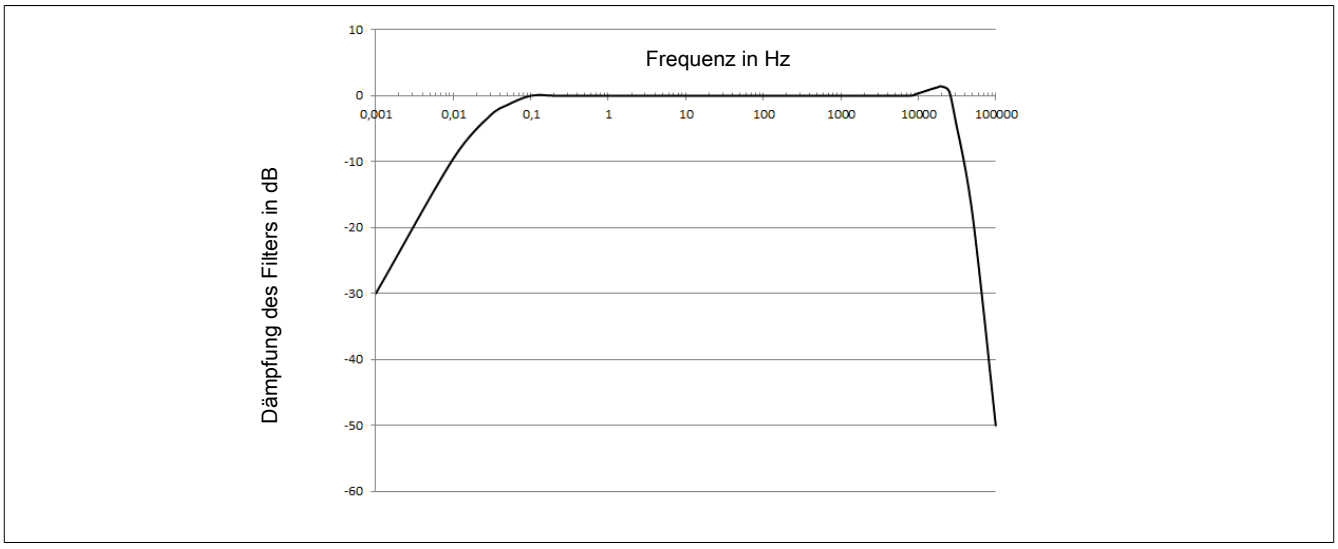


7 Eingangsschema



8 Gainverlauf

Die nachfolgende Grafik zeigt den typischen Gainverlauf des Moduls.



9 Derating

Je nach maximal auftretender Umgebungstemperatur sind neben dem Modul zusätzliche Blindmodule vorzusehen. Bei einer senkrechten Einbaulage der Module sind die Blindmodule bereits bei 5°C weniger Umgebungstemperatur zu verwenden.

Maximale Umgebungstemperatur

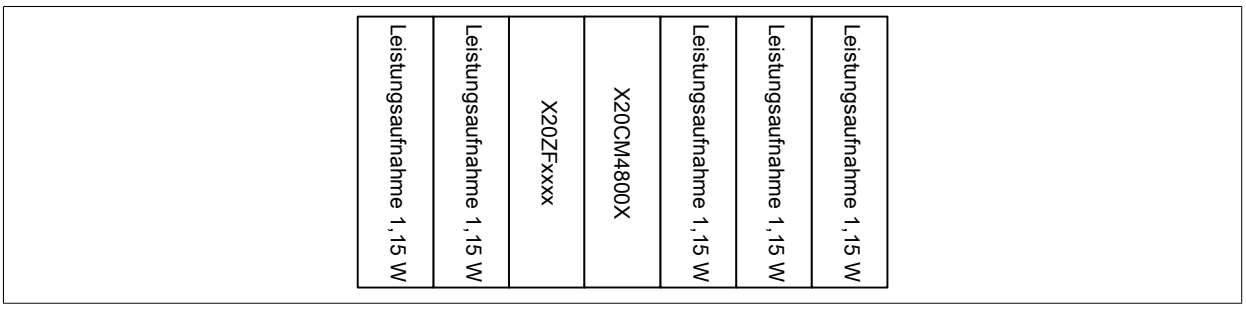
Bis Waagrecht 45°C / Senkrecht 40°C

- Es brauchen keine Vorkehrungen getroffen zu werden.

Bis Waagrecht 50°C / Senkrecht 45°C

- Abhängig von der verwendeten Versorgungsspannung ist ein Blindmodul vorzusehen.

  - 24 V Versorgungsspannung: Kein Blindmodul notwendig
  - 28,8 V Versorgungsspannung: 1 Blindmodul davor einfügen

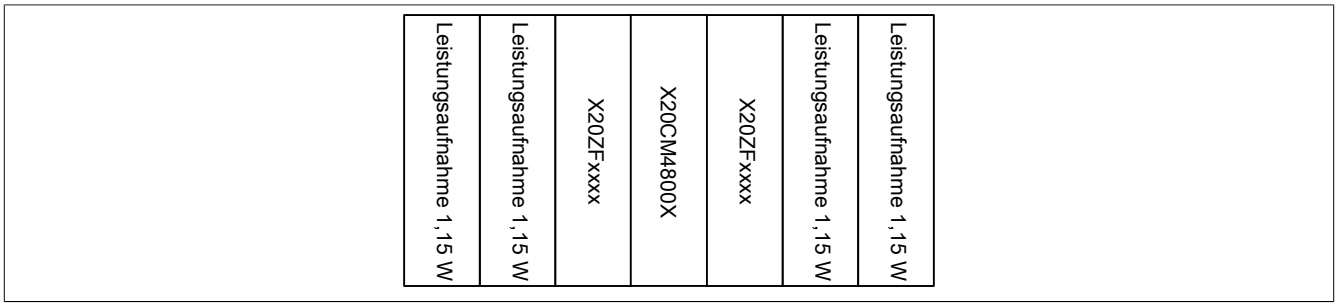


Bis Waagrecht 55°C / Senkrecht 50°C

- Es ist immer 1 Blindmodul davor vorzusehen (siehe Grafik oben).

Bis Waagrecht 60°C / Senkrecht 55°C

- Es sind immer 2 Blindmodule vorzusehen.



10 Funktionsbeschreibung

10.1 Konfiguration der Eingänge

Für die Übertragung der Rohdaten am X2X Link steht nur eine begrenzte Anzahl an Bytes zur Verfügung, die unter allen aktiven Kanäle aufgeteilt werden. Um den verwendeten Kanälen mehr Bytes zur Verfügung zu stellen, können unbenützte Kanäle deshalb abgeschaltet werden.

Weiters kann die IEPE-Sensorversorung pro Kanal extra aktiviert werden. Wird ein Sensor auf mehrere Kanäle verbunden, so muss die Versorgung nur bei einem Kanal aktiviert werden.

Information:

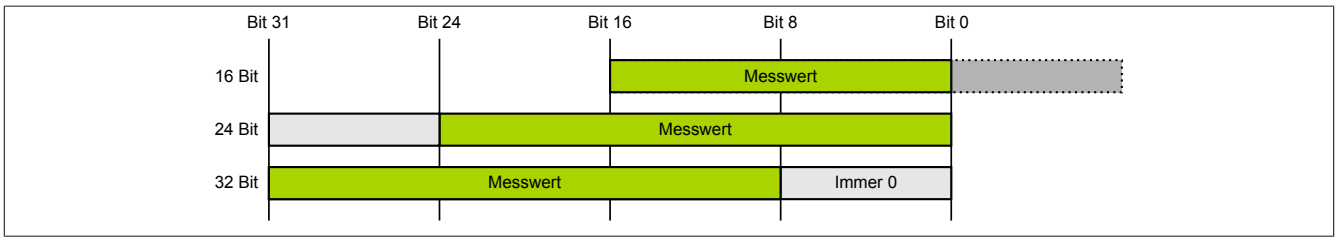
Die Register sind im Abschnitt "Kanalkonfiguration" auf Seite 10 beschrieben.

10.1.1 Datenauflösung

Die Daten werden vom Modul entsprechend der eingestellten Datenauflösung skaliert:

Modus	Datenformat	Maximalwert +10 V	Minimalwert -10 V	Skalierung
16 Bit	INT	32767 (0x7FFF)	-32767 (0x8000)	8 Bit nach rechts verschoben
24 Bit	DINT	8388607 (0x7FFFFF)	-8388608 (0x800000)	Interne Wandlerauflösung des Moduls[1)]
32 Bit	DINT	2147483392 (0x7FFFFF00)	-2147483648 (0x80000000)	8 Bit nach links verschoben; Bit 0 bis 7 sind immer 0

1) Die 3 Bytes müssen von der Applikation in einen DINT übertragen werden. Bei negativen Werten sind die Bits 25 bis 31 auf 1 zu setzen.



10.2 Vibrationen messen

Am Modul können bis zu 4 Vibrationssensoren angeschlossen werden. Die Rohdaten der Sensoren werden mit der eingestellten Abtastrate und Auflösung erfasst.

Die Verarbeitung und Auswertung der übertragenen Daten muss jedoch in der Applikation durchgeführt werden. Für eine Beschreibung, wie die Rohdaten in einen Vibrationswert umgerechnet werden, siehe "Umrechnung von Rohdaten auf [mg]" auf Seite 7.

Information:

Modulintern findet keine Vorverarbeitung der Daten statt.

10.2.1 Umrechnung von Rohdaten auf [mg]

Durch die folgende Formeln kann der Rohwert in einen Vibrationswert [mg] umgerechnet werden:

$$\text{Sensorauflösung} = 0{,}0001 * \frac{V}{\text{mg}}$$

$$\text{Vibrationswert [mg]} = \text{Rohwert} * \left(\frac{\text{Maximalauflösung\_10V\_Analogin}}{\text{Maximalwert\_10V\_Digitalin}}\right) * \frac{1}{\text{Sensorauflösung}}$$

Information:

Die Maximalauflösung richtet sich immer nach dem verwendeten Sensor. Das Modul arbeitet im Bereich von ±10 V. Bei einem 100 mV/g Sensor entspricht dies einem Maximalwert von ±100 g. Bei einem 50 mV/g Sensor liegt der Maximalwert bei ±200 g.

Beispiel

Es wird ein 100 mV/g Sensor verwendet, das Modul liefert als Rohwert den Wert 4608 und die Datenauflösung ist auf 24 Bit konfiguriert. Damit ergeben sich folgende Werte für die Berechnung:

- Rohwert = 4608
- Maximalwert_10V_Analogin = 10 V (Modulwert, nicht änderbar)
- Maximalwert_10V_Digitalin bei 24 Bit = 8388607 (siehe "Datenauflösung" auf Seite 6)
- Sensorauflösung = 100 mV/g = 0,1 V/g = 0,0001 V/mg

$$\text{Vibrationswert [mg]} = 4608 * \left(\frac{10\ \text{V}}{8388607}\right) * \frac{1}{0{,}0001\ \text{V/mg}} = 54{,}93\ \text{mg}$$

10.2.2 Flatstream

Die Datenschnittstelle für die Rohdaten basiert auf der Flatstream-Kommunikation. Die Bedienung erfolgt über die Bibliothek "AsFltGen".

Information:

Jeder Kanal verfügt über einen eigenen Flatstream.

Informationen zur Bibliothek "AsFltGen" können der Automation Help entnommen werden.

Die benötigten Bytes für die Übertragung der Rohdaten am Flatstream richten sich nach der eingestellten Abtastrate, der eingestellten Datenauflösung und der verwendeten Bus Zykluszeit.

Information:

Werden die Rohdaten nicht schnell genug zur Steuerung übertragen, so werden diese im Modul pro Kanal zwischengespeichert. Sobald der Zwischenspeicher gefüllt ist, kommt es zur BufferOverflow-Fehlermeldung.

Berechnungsbeispiel für benötigte Byteanzahl

Für eine Messauswertung werden folgenden Einstellungen durchgeführt:

- Buszykluszeit: 2 ms
- Abtastrate: 2 kHz (alle 500 µs ein Wert)
- Datenauflösung: 16 Bit (2 Byte)

In der Buszykluszeit von 2 ms ergeben sich somit 4 Werte zu je 2 Byte – insgesamt 8 Bytes. Entsprechend dieser Berechnung muss die Konfiguration der Flatstream-Datenschnittstelle erfolgen.

Wird die Buszykluszeit von 2 ms auf 1 ms verkleinert, dann verkleinert sich dadurch die Anzahl der benötigten Bytes für die Rohdaten auf die Hälfte; das heißt 4 Bytes. Dadurch stehen andere Kanälen mehr Bytes zur Verfügung.

Einstellen der MTU-Größe

Die einzustellende MTU-Größe für den Flatstream ergibt sich aus den berechneten Datenbytes + 1 Controlbyte. Somit muss die Input-MTU Größe auf mindestens 8 + 1 = 9 Byte bzw. im zweiten Fall auf 4 + 1 = 5 Byte konfiguriert werden.

Da dies nur der Idealwert ist, sind noch Reservebytes einzuplanen, um eventuelle Timingtoleranzen oder Kommunikationsfehler auszugleichen. Ansonsten kann es vorkommen, dass zeitlich nicht alle Daten übertragen werden können und es irgendwann zum Bufferoverflow kommt.

10.3 NetTime

Wird eine Messung für einen Kanal gestartet so wird automatisch ein Zeitstempel für den ersten Rohwert ermittelt. Über die konfigurierte Abtastrate kann somit zu jedem Rohwert ein eindeutiger Zeitbezug hergestellt werden.

Zusätzlich wird auch nach Beendigung der Messung wieder automatisch ein Zeitstempel ermittelt.

Information:

Die Zeitstempel für die weiteren Rohwerte müssen applikativ ermittelt werden. Das Modul stellt nur den Start- bzw. Endzeitstempel zur Verfügung.

Information:

Die Register sind im Abschnitt "NetTime" auf Seite 12 beschrieben.

11 Registerbeschreibung

11.1 Funktionsmodell 1 - Standard

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Konfiguration						
770	CfO_ChannelConfig01 CfO_ChannelConfig01_rb	UINT		•		•
774	CfO_ChannelConfig02 CfO_ChannelConfig02_rb	UINT		•		•
778	CfO_ChannelConfig03 CfO_ChannelConfig03_rb	UINT		•		•
782	CfO_ChannelConfig04 CfO_ChannelConfig04_rb	UINT		•		•
Kommunikation						
1	Messungen starten/stoppen und Pufferüberlauf	USINT			•	
	Measurement01	Bit 0				
	...	..				
	Measurement04	Bit 3				
	BufferOverflowAck01	Bit 4				
	...	...				
	BufferOverflowAck04	Bit 7				
2	Status des Moduls	UINT	•			
	MeasurementState01	Bit 0				
	...	...				
	MeasurementState04	Bit 3				
	BufferOverflow01	Bit 4				
	...	...				
	BufferOverflow04	Bit 7				
	BrokenWire01	Bit 8				
	...	...				
	BrokenWire04	Bit 11				
Nettime						
788	Nettime_StartMeasCh01	DINT		•		
796	Nettime_StartMeasCh02	DINT		•		
804	Nettime_StartMeasCh03	DINT		•		
812	Nettime_StartMeasCh04	DINT		•		
820	Nettime_StopMeasCh01	DINT		•		
828	Nettime_StopMeasCh02	DINT		•		
836	Nettime_StopMeasCh03	DINT		•		
844	Nettime_StopMeasCh04	DINT		•		
Flatstream						
Kanal 1						
1281	Ch1_CfO_OutputMTU	USINT				•
1283	Ch1_CfO_InputMTU	USINT				•
1285	Ch1_CfO_FlatstreamMode	USINT				•
1287	Ch1_CfO_Forward	USINT				•
1290	Ch1_CfO_ForwardDelay	UINT				•
1536	Ch1_CfO_InputSequence	USINT	•			
1536 + Index	Ch1_CfO_RxByteN (Index N = 1 bis 25)	USINT	•			
1536	Ch1_CfO_OutputSequence	USINT			•	
1536 + Index	Ch1_CfO_TxByteN (Index N = 1 bis 3)	USINT			•	
Kanal 2						
1297	Ch2_CfO_OutputMTU	USINT				•
1299	Ch2_CfO_InputMTU	USINT				•
1301	Ch2_CfO_FlatstreamMode	USINT				•
1303	Ch2_CfO_Forward	USINT				•
1306	Ch2_CfO_ForwardDelay	UINT				•
1792	Ch2_CfO_InputSequence	USINT	•			
1792 + Index	Ch2_CfO_RxByteN (Index N = 1 bis 25)	USINT	•			
1792	Ch2_CfO_OutputSequence	USINT			•	
1792 + Index	Ch2_CfO_TxByteN (Index N = 1 bis 3)	USINT			•	
Kanal 3						
1313	Ch3_CfO_OutputMTU	USINT				•
1315	Ch3_CfO_InputMTU	USINT				•
1317	Ch3_CfO_FlatstreamMode	USINT				•
1319	Ch3_CfO_Forward	USINT				•
1322	Ch3_CfO_ForwardDelay	UINT				•
2048	Ch3_CfO_InputSequence	USINT	•			
2048+ Index	Ch3_CfO_RxByteN (Index N = 1 bis 25)	USINT	•			
2048	Ch3_CfO_OutputSequence	USINT			•	
2048 + Index	Ch3_CfO_TxByteN (Index N = 1 bis 3)	USINT			•	

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Kanal 4						
1329	Ch4_CfO_OutputMTU	USINT				•
1331	Ch4_CfO_InputMTU	USINT				•
1333	Ch4_CfO_FlatstreamMode	USINT				•
1335	Ch4_CfO_Forward	USINT				•
1338	Ch4_CfO_ForwardDelay	UINT				•
2304	Ch4_CfO_InputSequence	USINT	•			
2304 + Index	Ch4_CfO_RxByteN (Index N = 1 bis 25)	USINT	•			
2404	Ch4_CfO_OutputSequence	USINT			•	
2404 + Index	Ch4_CfO_TxByteN (Index N = 1 bis 3)	USINT			•	

11.2 Konfiguration

11.2.1 Kanalkonfiguration

Name:
CfO_ChannelConfig01 bis CfO_ChannelConfig04
CfO_ChannelConfig01_rb CfO_ChannelConfig04_rb

In diesen Registern können die jeweiligen Kanäle konfiguriert werden.

Datentyp	Werte
UINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	Kanal de-/aktivieren	0	Kanal deaktiviert
		1	Kanal aktiviert
1	Sensorversorgung	0	Sensorversorgung deaktiviert
		1	Sensorversorgung aktiviert
2 - 3	Datenauflösung[1)]	0	32 Bit
		1	24 Bit
		2	16 Bit
4 - 7	Abtastrate[2)]	0	50000 Hz
		1	25000 Hz
		2	10000 Hz
		3	5000 Hz
		4	2500 Hz
		5	2000 Hz
		6	1000 Hz
		7	500 Hz
		8	200 Hz
8 - 15	Reserviert	0	

1) Maximal- bzw. Minimalwert der jeweiligen Auflösung entspricht ±10 VAC.
2) Die Abtastrate eines analogen Signals bezogen auf 1 s. Die Angabe erfolgt in [Hz].
Beispiele:

- Die Abtastung eines analogen Signals 1x pro Sekunde entspricht einer Abtastrate von 1 Hz.
- Die Abtastung eines analogen Signals 1x pro ms entspricht einer Abtastrate von 1 kHz.

11.3 Kommunikation

11.3.1 Messungen starten/stoppen und Pufferüberlauf

Name:
Measurement01 bis Measurement04
BufferOverflowAck01 bis BufferOverflowAck04

In diesem Register können die Messungen gestartet bzw gestoppt werden. Zustätzlich kann ein eventueller Pufferüberlauf quittiert werden.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	Measurement01	0	Messung Kanal 1 stoppen
		1	Messung Kanal 1 starten
...		...	
3	Measurement04	0	Messung Kanal 4 stoppen
		1	Messung Kanal 4 starten
4	BufferOverflowAck01	0	Pufferüberlauf Kanal 1 nicht quittieren
		1	Pufferüberlauf Kanal 1 quittieren
...		...	
7	BufferOverflowAck04	0	Pufferüberlauf Kanal 4 nicht quittieren
		1	Pufferüberlauf Kanal 4 quittieren

11.3.2 Status des Moduls

Name:
MeasurementState01 bis MeasurementState04
BufferOverflow01 bis BufferOverflow04
BrokenWire01 bis BrokenWire04

In diesem Register wird der Status des Moduls angegeben.

Datentyp	Werte
UINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	MeasurementState01	0	Messung Kanal 1 gestoppt
		1	Messung Kanal 1 läuft
...		...	
3	MeasurementState04	0	Messung Kanal 4 gestoppt
		1	Messung Kanal 4 läuft
4	BufferOverflow01	0	Kein Pufferüberlauf Kanal 1
		1	Pufferüberlauf Kanal 1
...		...	
7	BufferOverflow04	0	Kein Pufferüberlauf Kanal 4
		1	Pufferüberlauf Kanal 4
8	BrokenWire01	0	Kein Fehler Kanal 1
		1	Drahtbruch Kanal 1
...		...	
11	BrokenWire04	0	Kein Fehler Kanal 4
		1	Drahtbruch Kanal 4
12 - 15	Reserviert	-	

BufferOverflow

Jeder Kanal ist mit einem internen Puffer von 50 kByte ausgestattet. Zu einem Pufferüberlauf kommt es, je nach eingestellter Abtastrate, Datenauflösung und MTU-Größe, frühestens nach ca. 250 ms, wenn die Daten nicht per Flatstream übertragen werden. Der Überlauf muss durch die Applikation mit Hilfe des Registers BufferOverflowAck0x quittiert werden.

11.4 NetTime

11.4.1 Zeitpunkt der ersten gültigen Abtastung

Name:
Nettime_StartMeasCh01 bis Nettime_StartMeasCh04

Nach Starten der Messungen wird in dieses Register der Zeitstempel der ersten gültigen Abtastung geschrieben.

Datentyp	Werte	Information
DINT	-2.147.483.648 bis 2.147.483.647	Zeitstempel in µs

11.4.2 Zeitpunkt der letzten gültigen Abtastung

Name:
Nettime_StopMeasCh01 bis Nettime_StopMeasCh04

In dieses Register wird der Zeitstempel der letzten gültigen Abtastung geschrieben. Nach Beendigung der Messung steht damit der Zeitstempel der letzten gültigen Abtastung einer Messung zur Verfügung.

Datentyp	Werte	Information
DINT	-2.147.483.648 bis 2.147.483.647	Zeitstempel in µs

11.4.3 NetTime Technology

Unter NetTime versteht man die Möglichkeit Systemzeiten zwischen einzelnen Komponenten der Steuerung bzw. Netzwerks (CPU, I/O-Module, X2X Link, POWERKLINK usw.) exakt aufeinander abzustimmen und zu übertragen.

Damit kann von Ereignissen der Zeitpunkt des Auftritts systemweit µs-genau bestimmt werden. Ebenso können anstehende Ereignisse exakt zu einem vorgegebenen Zeitpunkt ausgeführt werden.

NetTime TECHNOLOGY

11.4.3.1 Zeitinformationen

In der Steuerung bzw. im Netzwerk sind verschiedene Zeitinformationen vorhanden:

- Systemzeit (auf der SPS, APC usw.)
- X2X Link Zeit (für jedes X2X Link Netzwerk)
- POWERLINK-Zeit (für jedes POWERLINK-Netzwerk)
- Zeitdatenpunkte von I/O-Modulen

Die NetTime basiert auf 32 Bit Zähler, welche im µs-Takt erhöht werden. Das Vorzeichen der Zeitinformation wechselt nach 35 min 47 s 483 ms 648 µs und zu einem Überlauf kommt es nach 71 min 34 s 967 ms 296 µs.

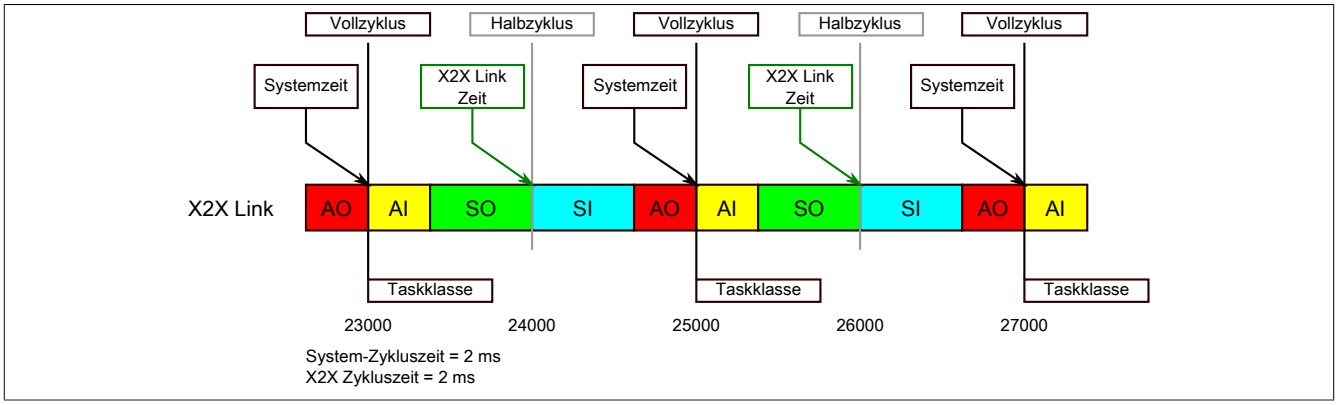
Die Initialisierung der Zeiten erfolgt auf Basis der Systemzeit während des Hochlaufs des X2X Links, der I/O-Module bzw. der POWERLINK-Schnittstelle.

Aktuelle Zeitinformationen in der Applikation können auch über die Bibliothek AsIOTime ermittelt werden.

11.4.3.1.1 SPS/Controller-Datenpunkte

Die NetTime I/O-Datenpunkte der SPS oder des Controllers werden zu jedem Systemtakt gelatcht und zur Verfügung gestellt.

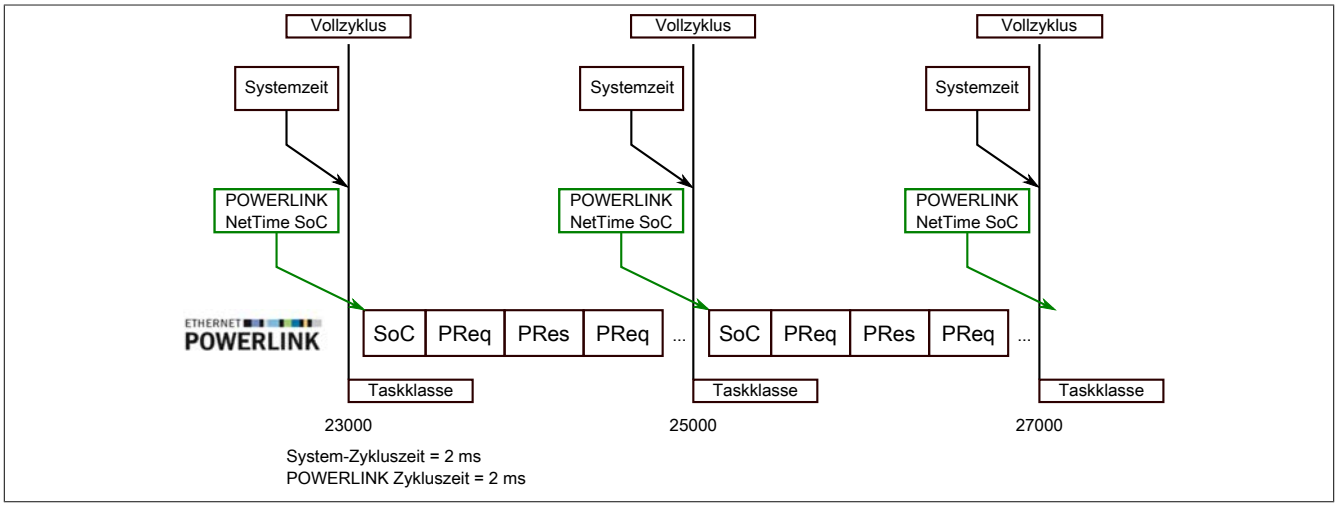
11.4.3.1.2 Refenzzeitpunkt X2X Link



Der Referenzzeitpunkt am X2X Link wird grundsätzlich zum Halbzyklus des X2X Link Zyklus gebildet. Dadurch ergibt sich beim Auslesen des Referenzzeitpunktes eine Differenz zwischen Systemzeit und X2X Link Referenzzeit.

Im Beispiel oben bedeutet dies einen Unterschied von 1 ms, das heißt, wenn zum Zeitpunkt 25000 im Task die Systemzeit und die X2X Link Referenzzeit miteinander verglichen werden, dann liefert die Systemzeit den Wert 25000 und die X2X Link Referenzzeit den Wert 24000.

11.4.3.1.3 Referenzzeitpunkt POWERLINK

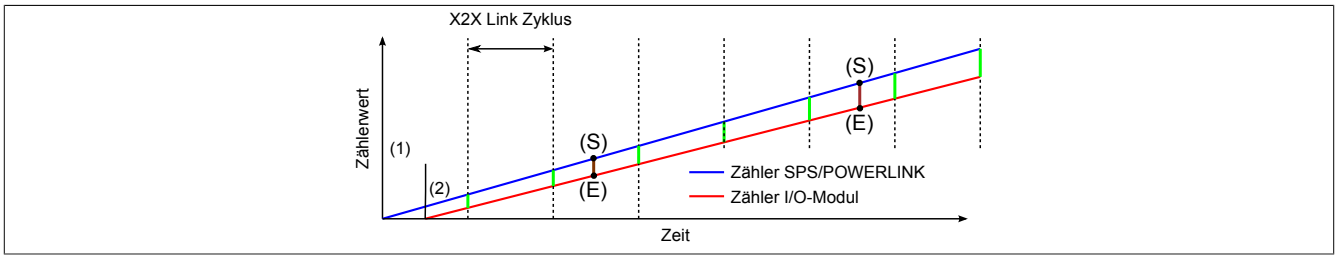


Der Referenzzeitpunkt am POWERLINK wird grundsätzlich beim SoC (Start of Cycle) des POWERLINK-Netzwerks gebildet. Der SoC startet systembedingt 20 µs nach dem Systemtakt. Dadurch ergibt sich folgende Differenz zwischen Systemzeit und POWERLINK-Referenzzeit:

POWERLINK-Referenzzeit = Systemzeit - POWERLINK-Zykluszeit + 20 µs.

Im Beispiel oben bedeutet dies einen Unterschied von 1980 µs, das heißt, wenn zum Zeitpunkt 25000 im Task die Systemzeit und die POWERINK-Referenzzeit miteinander betrachtet werden, dann liefert die Systemzeit den Wert 25000 und die POWERLINK-Referenzzeit den Wert 23020.

11.4.3.1.4 Synchronisierung von Systemzeit/POWERLINK-Zeit und I/O-Modul



Beim Hochfahren starten die internen Zähler für die SPS/POWERLINK (1) und dem I/O-Modul (2) zu unterschiedlichen Zeiten und erhöhen die Werte im µs-Takt.

Am Beginn jedes X2X Link Zyklus wird von der SPS bzw. vom POWERLINK-Netzwerk eine Zeitinformation an das I/O-Modul gesendet. Das I/O-Modul vergleicht diese Zeitinformation mit der modulinternen Zeit und bildet eine Differenz (grüne Linie) zwischen beiden Zeiten und speichert diese ab.

Bei Auftreten eines NetTime-Ereignisses (E) wird die modulinterne Zeit ausgelesen und mit dem gespeicherten Differenzwert korrigiert (braune Linie). Dadurch kann auch bei nicht absolut gleichlaufenden Zählern immer der exakte Systemzeitpunkt (S) eines Ereignisses ermittelt werden.

Anmerkung

Die Taktungenauigkeit ist im Bild als rote Linie stark überhöht dargestellt.

11.4.3.2 Zeitstempelfunktionen

NetTime-fähige Module stellen je nach Funktionsumfang verschiedene Zeitstempelfunktionen zur Verfügung. Tritt ein Zeitstempelereignis auf, so speichert das Modul unmittelbar die aktuelle NetTime. Nach der Übertragung der jeweiligen Daten inklusive dieses exakten Zeitpunkts an die CPU kann diese nun, gegebenenfalls mit Hilfe ihrer eigenen NetTime (bzw. Systemzeit), die Daten auswerten.

11.4.3.2.1 Zeitbasierte Eingänge

Über die NetTime Technology kann der exakte Zeitpunkt einer steigenden Flanke an einem Eingang ermittelt werden. Ebenso kann auch die steigende sowie fallende Flanke erkannt und daraus die Zeitdauer zwischen 2 Ereignissen ermittelt werden.

Information:

Der ermittelte Zeitpunkt liegt immer in der Vergangenheit.

11.4.3.2.2 Zeitbasierte Ausgänge

Über die NetTime Technology kann der exakte Zeitpunkt einer steigenden Flanke an einem Ausgang vorgegeben werden. Ebenso kann auch die steigende sowie fallende Flanke vorgegeben und daraus ein Pulsmuster generiert werden.

Information:

Die vorgegebene Zeit muss immer in der Zukunft liegen und die eingestellte X2X Link Zykluszeit für die Definition des Zeitpunkts berücksichtigt werden.

11.4.3.2.3 Zeitbasierte Messungen

Über die NetTime Technology kann der exakte Zeitpunkt einer stattgefundenen Messung ermittelt werden. Es kann dabei sowohl der Anfangs- und/oder der Endzeitpunkt der Messung übermittelt werden.

11.5 Die Flatstream-Kommunikation

11.5.1 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.

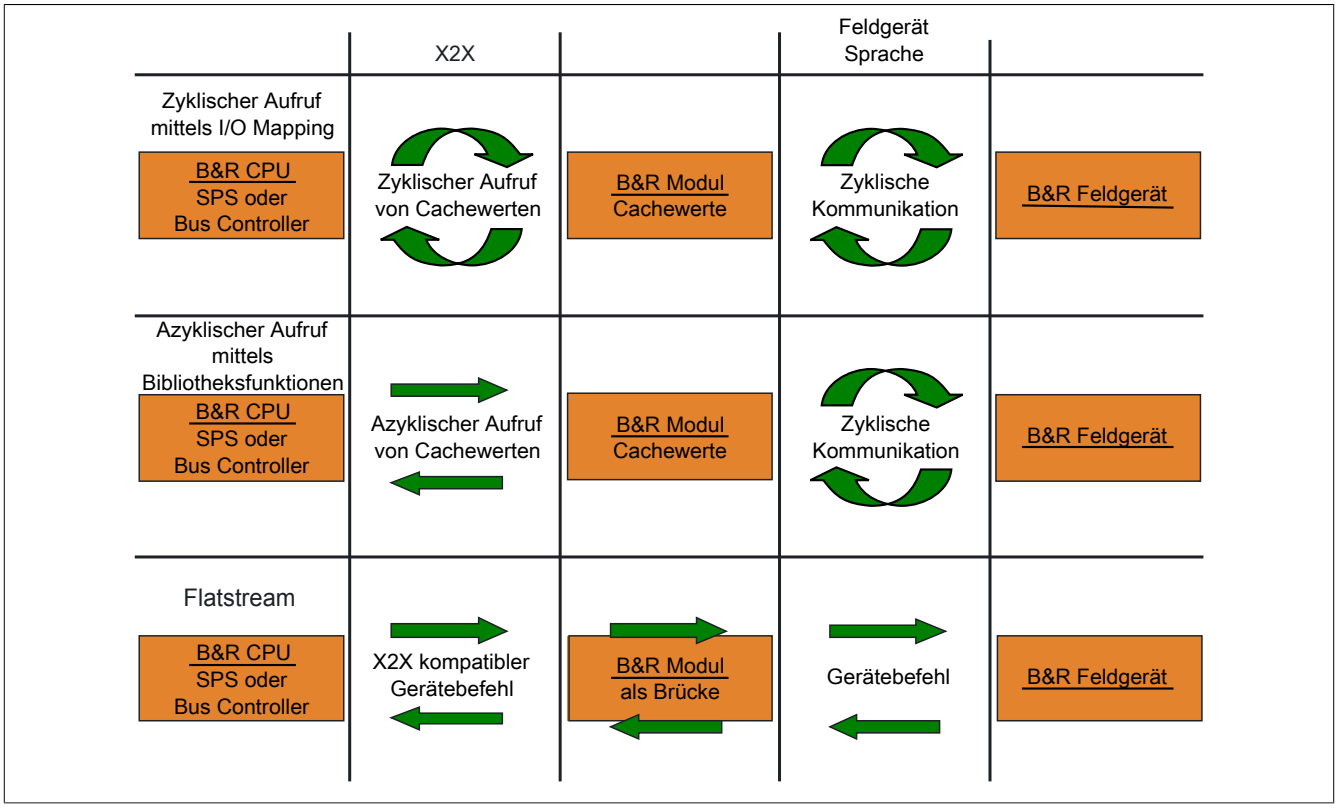


Abbildung 1: 3 Arten der Kommunikation

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.

11.5.2 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.

11.5.3 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.

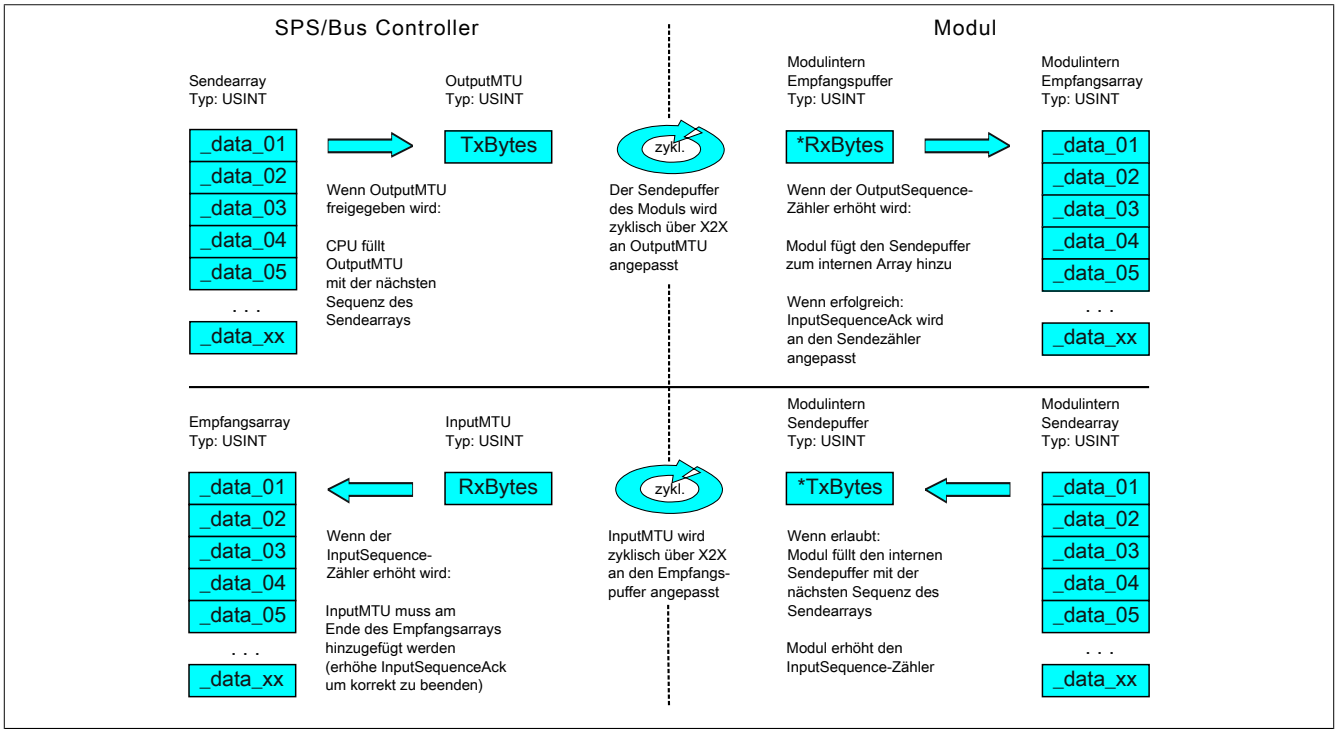


Abbildung 2: Kommunikation per Flatstream

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.

11.5.4 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.

11.5.4.1 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.

11.5.4.1.1 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)

11.5.4.2 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.

11.5.4.2.1 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

11.5.4.2.2 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

11.5.4.2.3 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

11.5.4.2.4 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.

11.5.4.2.5 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.

11.5.4.2.6 Beziehung zwischen Output- und InputSequence

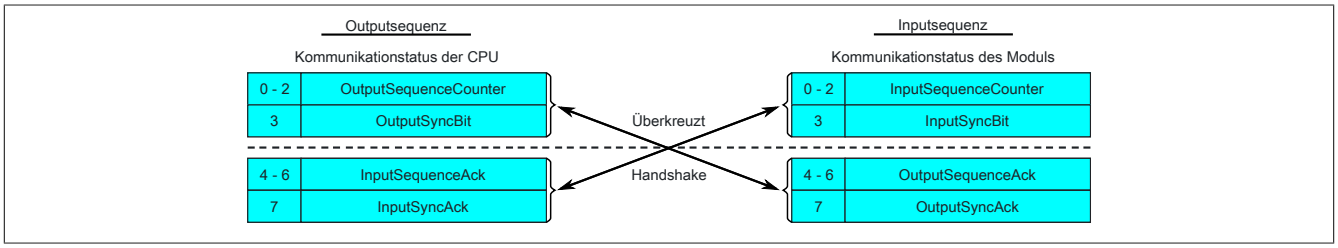


Abbildung 3: Zusammenhang zwischen Output- und InputSequence

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.

11.5.4.3 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.

11.5.4.4 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.

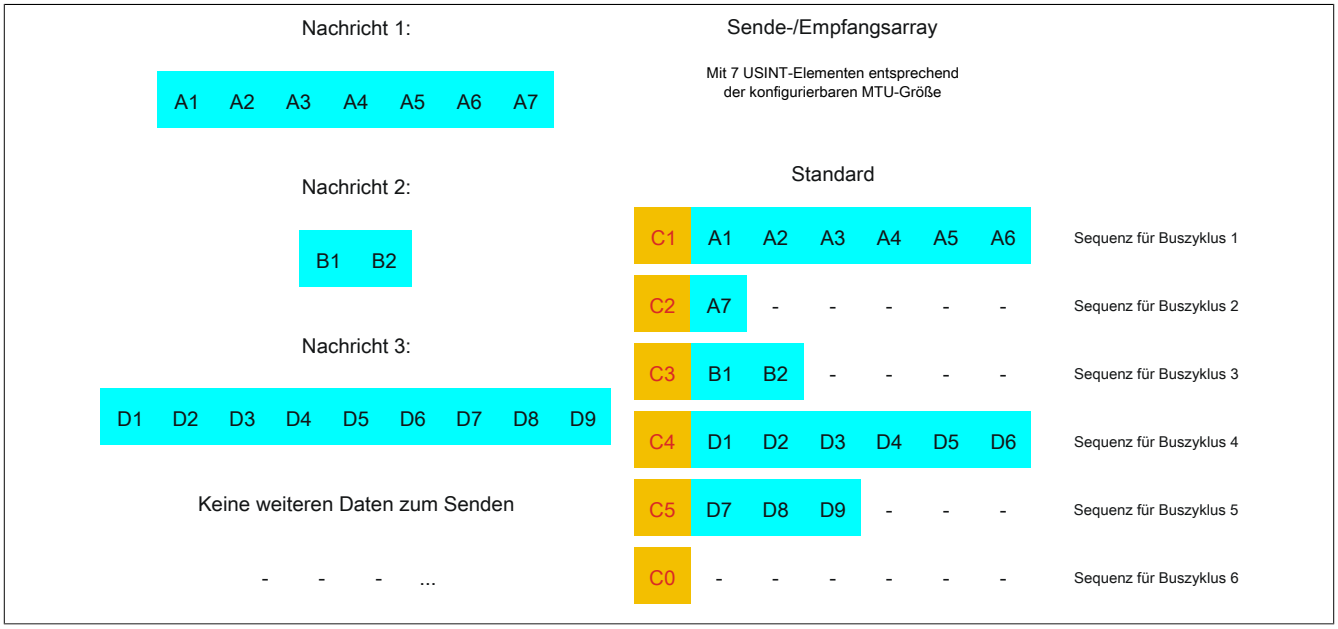


Abbildung 4: Sende-/Empfangsarray (Standard)

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)

11.5.4.5 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.

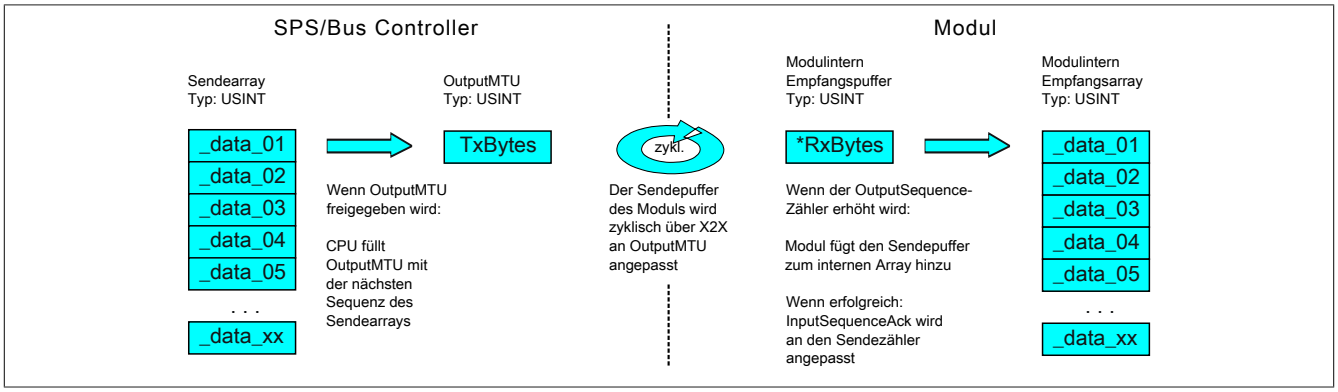


Abbildung 5: Kommunikation per Flatstream (Output)

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

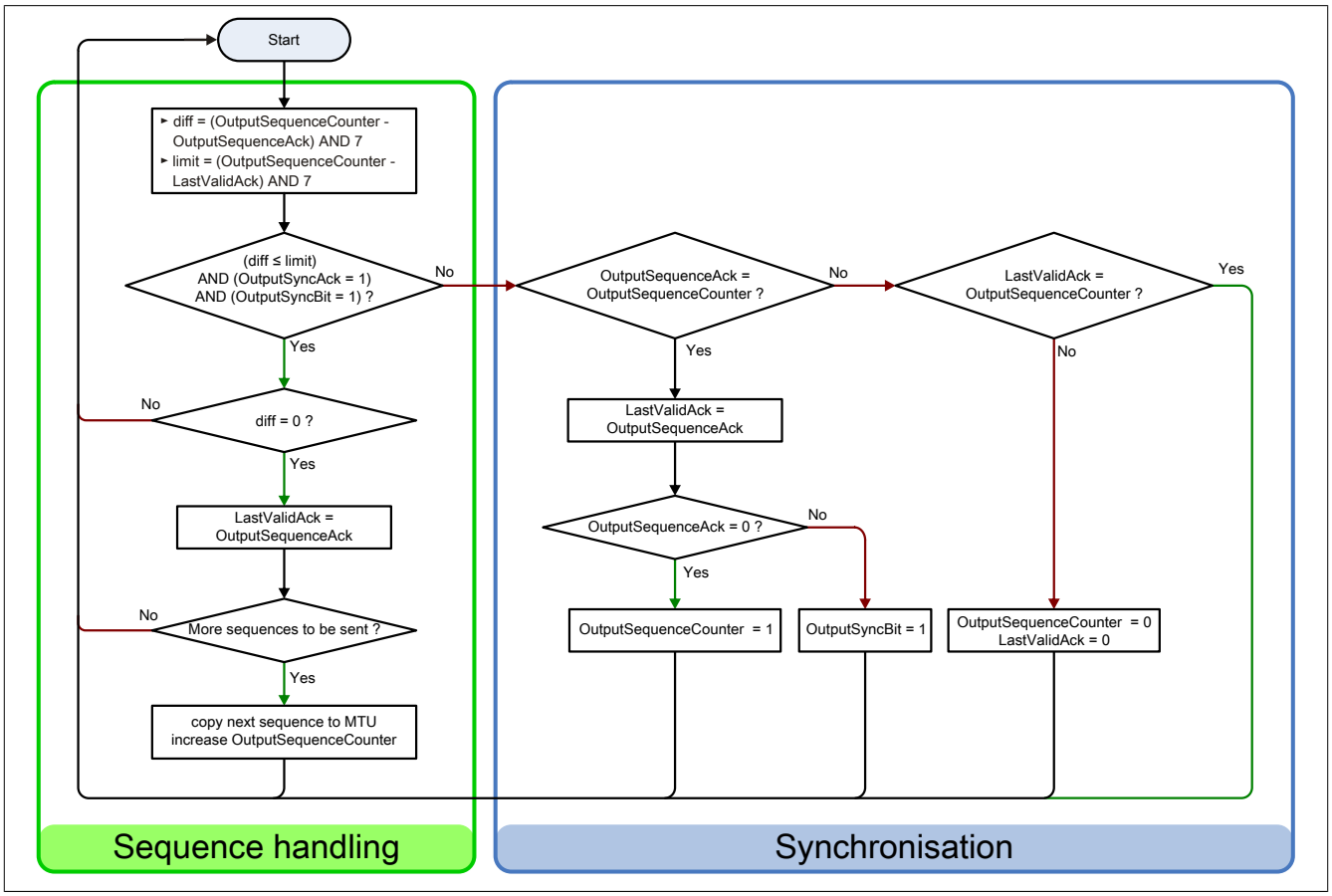


Abbildung 6: Ablaufdiagramm für Output-Richtung

11.5.4.6 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.

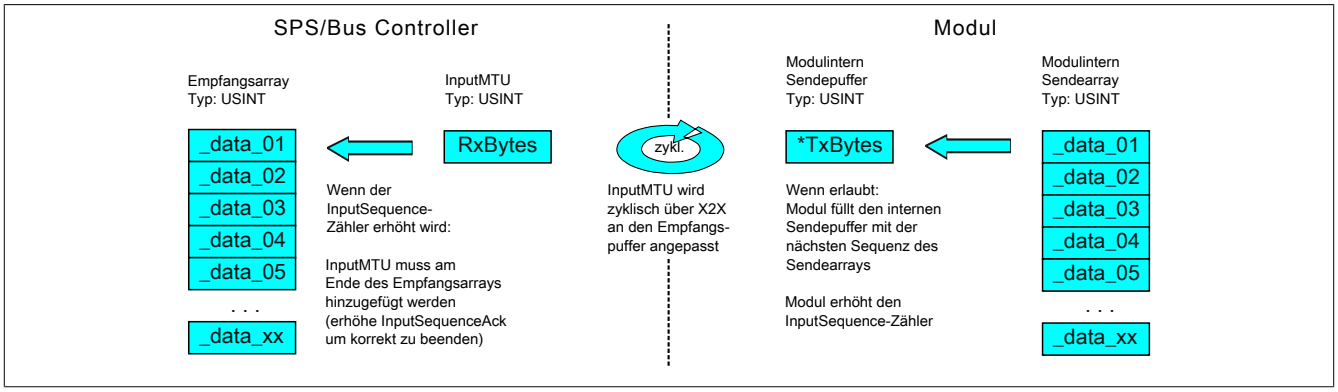


Abbildung 7: Kommunikation per Flatstream (Input)

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

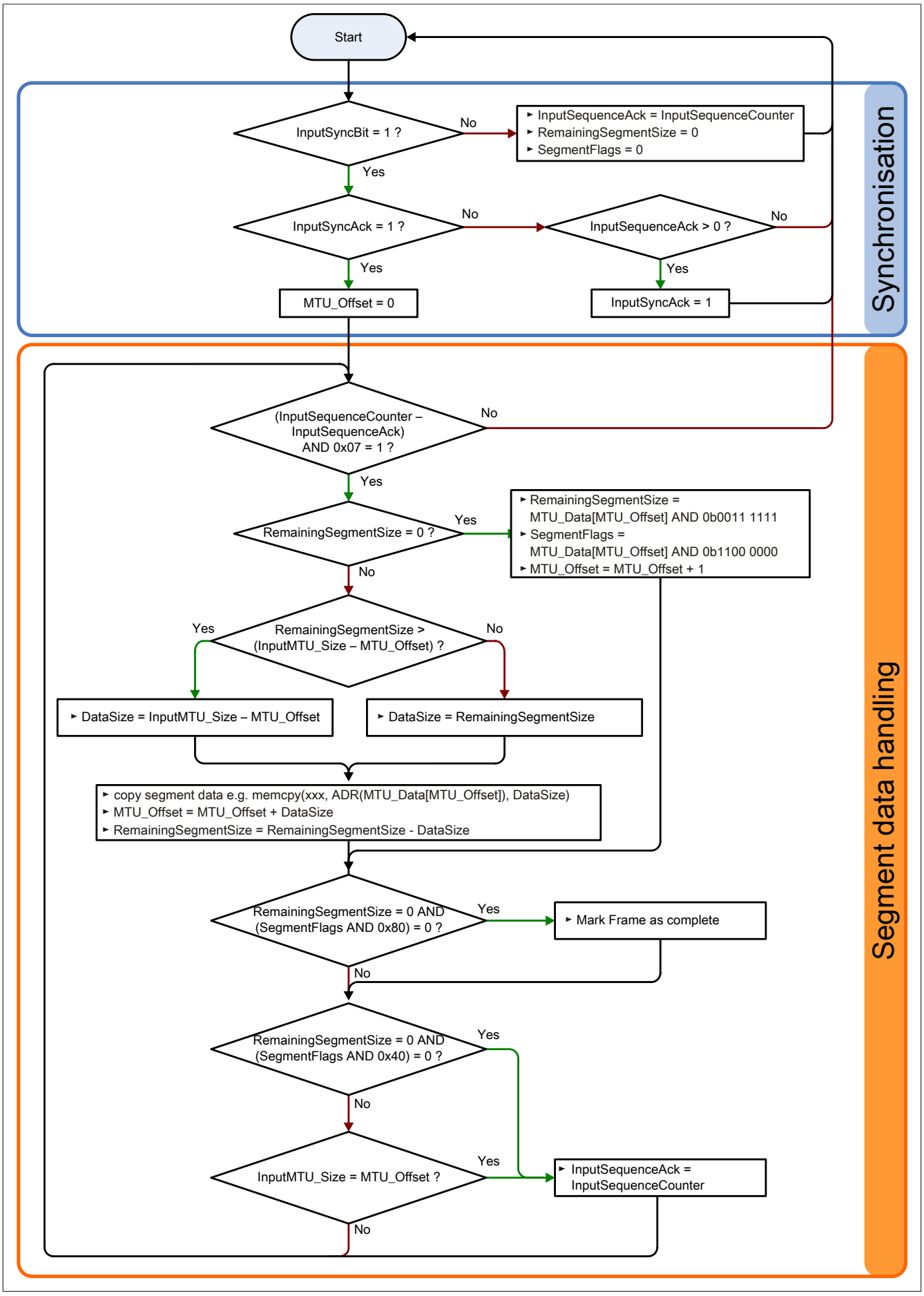


Abbildung 8: Ablaufdiagramm für Input-Richtung

11.5.4.7 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.

11.5.4.8 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.

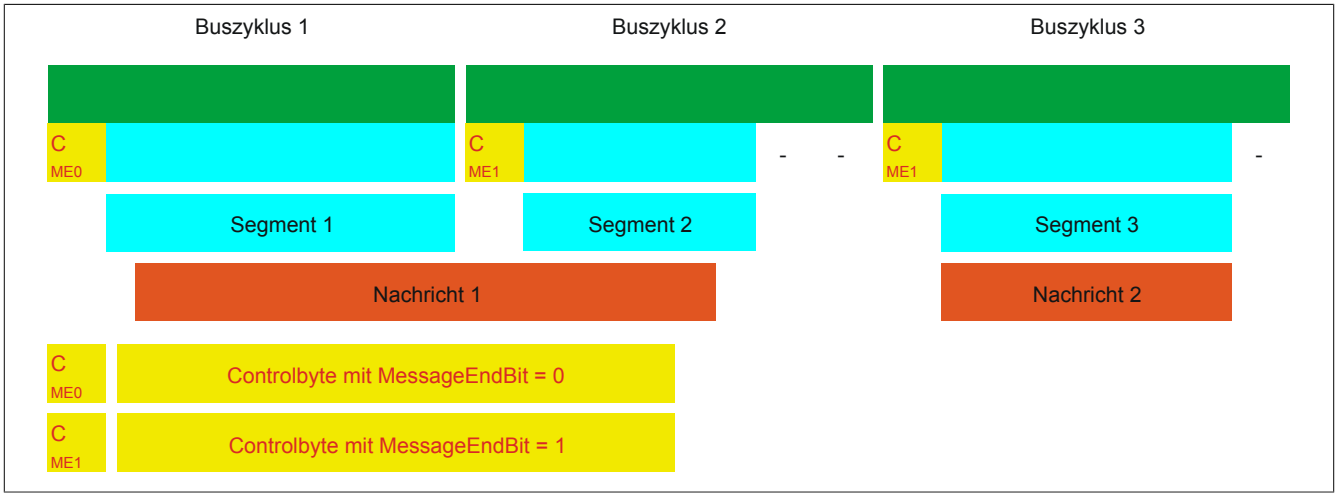


Abbildung 9: Anordnung von Nachrichten in der MTU (Standard)

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.

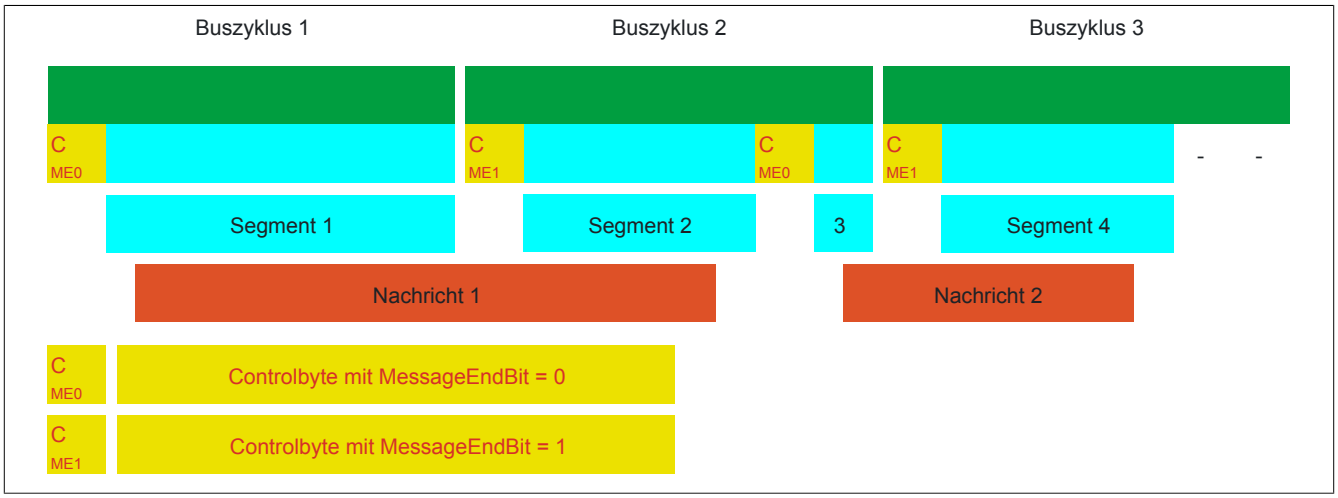


Abbildung 10: Anordnung von Nachrichten in der MTU (MultiSegmentMTU)

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.

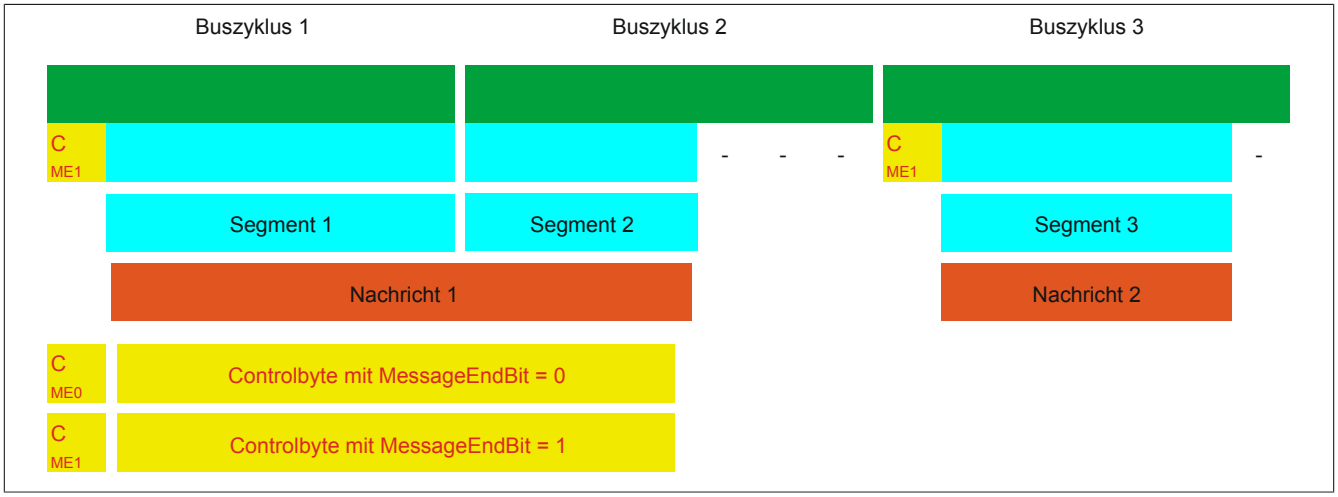


Abbildung 11: Anordnung von Nachrichten in der MTU (große Segmente)

Anwendung beider Optionen

Die beiden Optionen dürfen auch gleichzeitig angewendet werden.

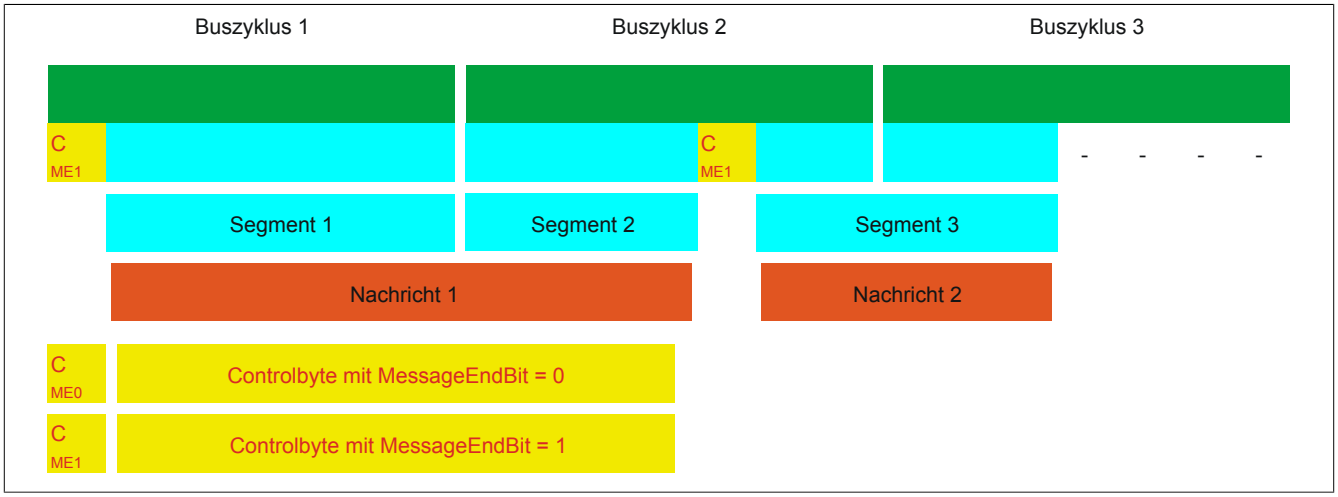


Abbildung 12: Anordnung von Nachrichten in der MTU (große Segmente und MultiSegmentMTU)

11.5.4.9 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.

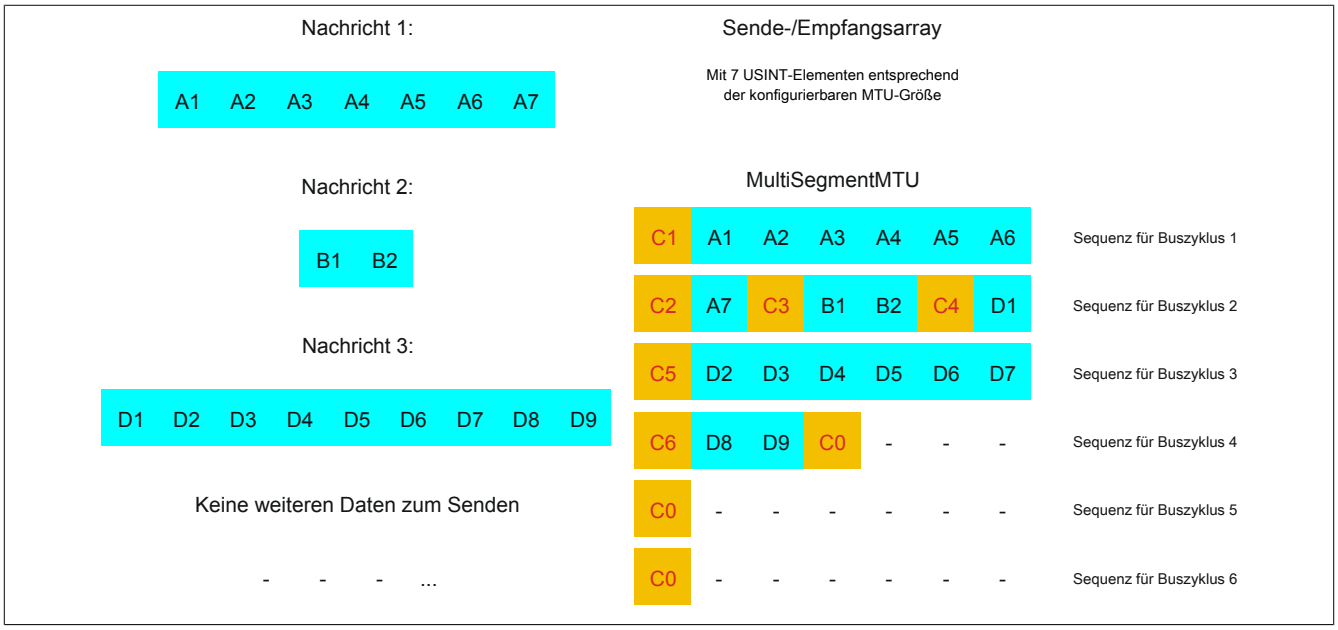


Abbildung 13: Sende-/Empfangsarray (MultiSegmentMTU)

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.

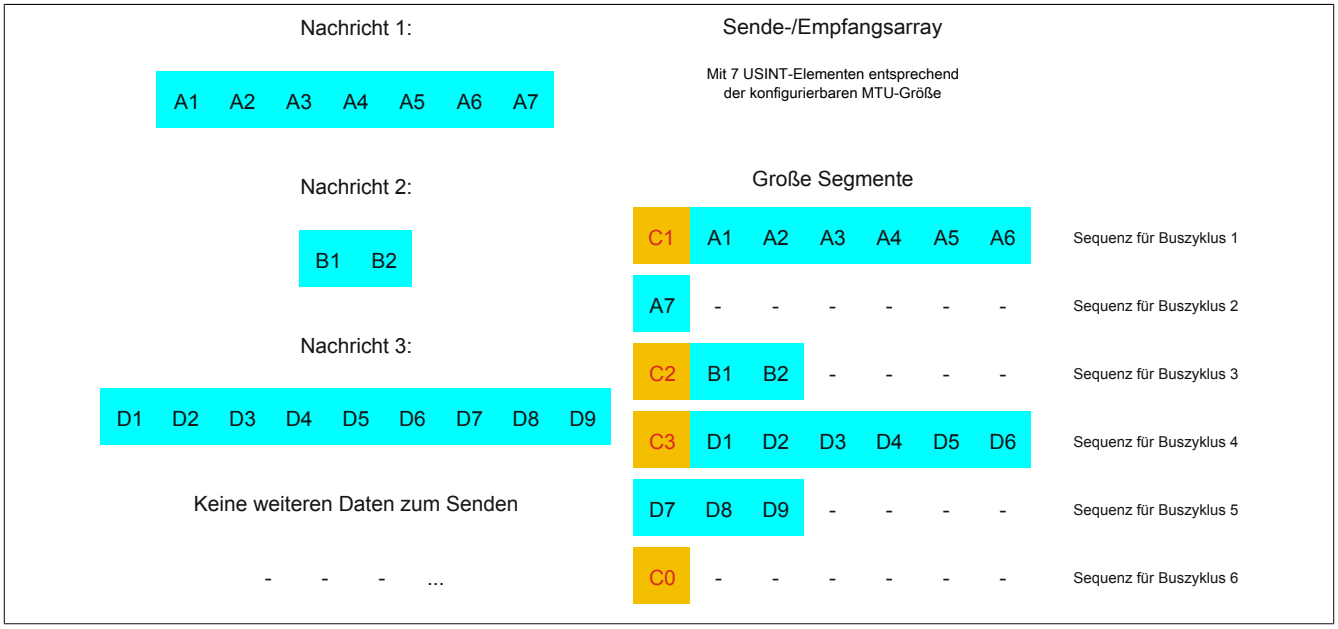


Abbildung 14: Sende-/Empfangsarray (große Segmente)

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.

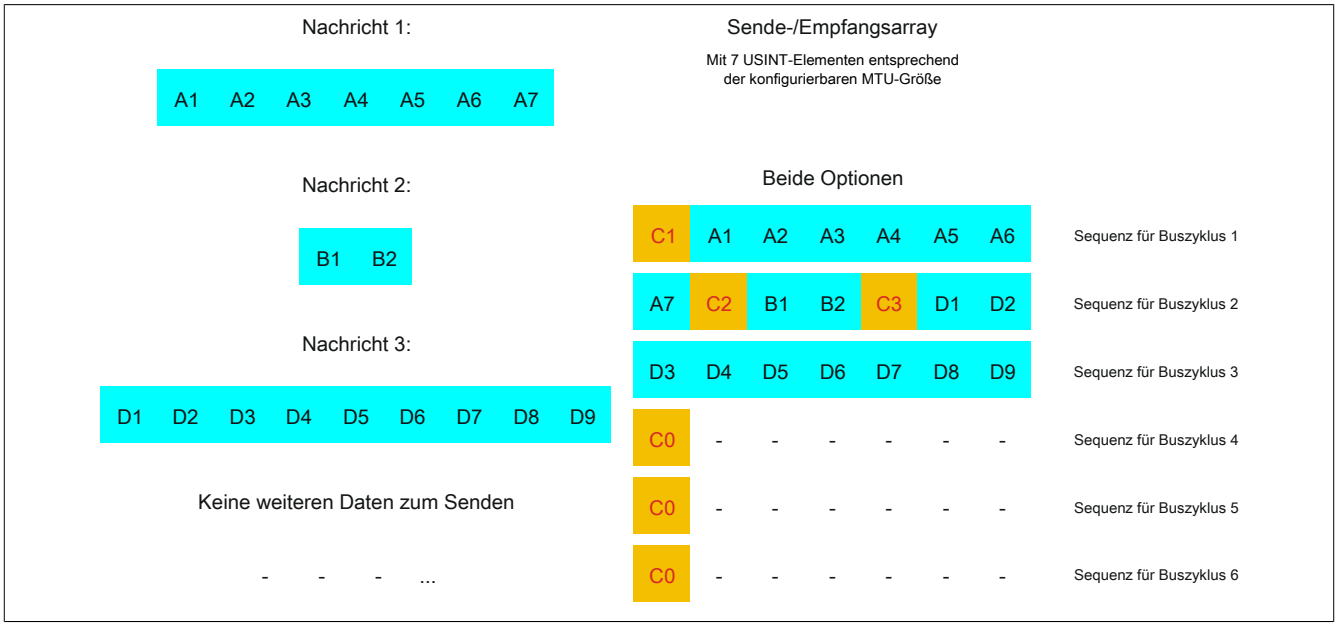


Abbildung 15: Sende-/Empfangsarray (große Segmente und MultiSegmentMTU)

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

11.5.5 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.

11.5.5.1 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)

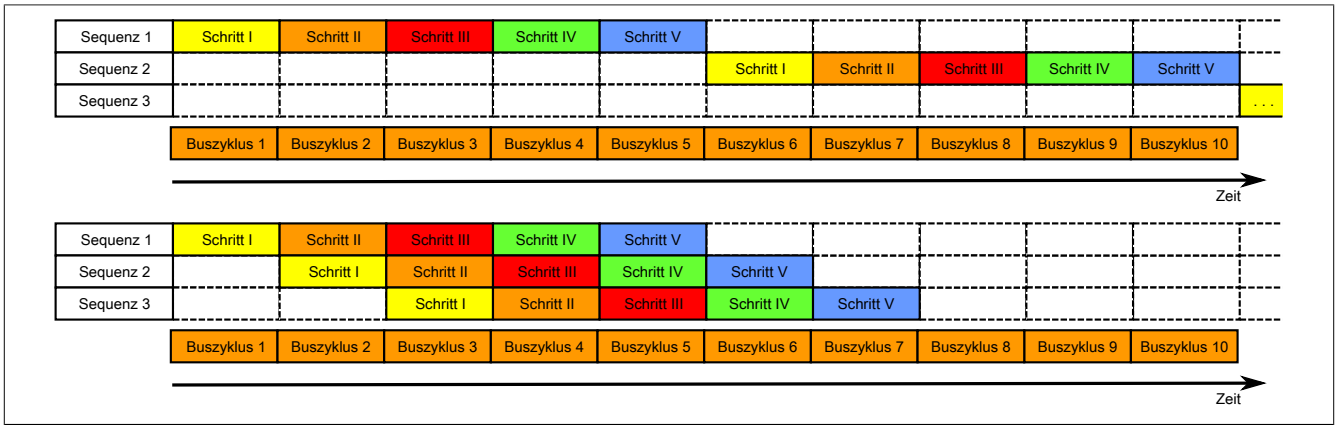


Abbildung 16: Vergleich Übertragung ohne bzw. mit Forward

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.

11.5.5.2 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.

11.5.5.2.1 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

11.5.5.2.2 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

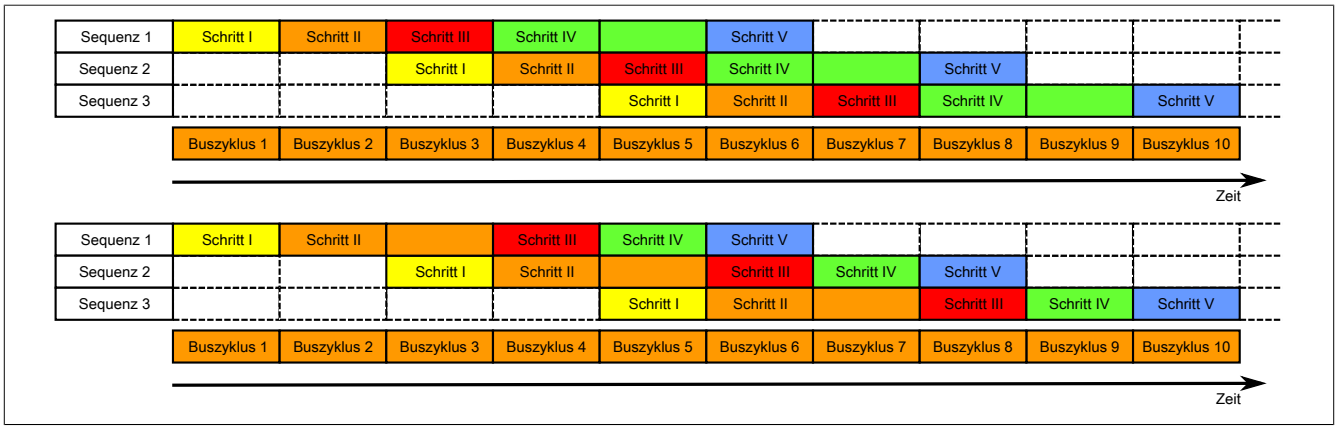


Abbildung 17: Auswirkung des ForwardDelay bei der Flatstream-Kommunikation mit Forward

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.

11.5.5.3 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.

11.5.5.4 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).

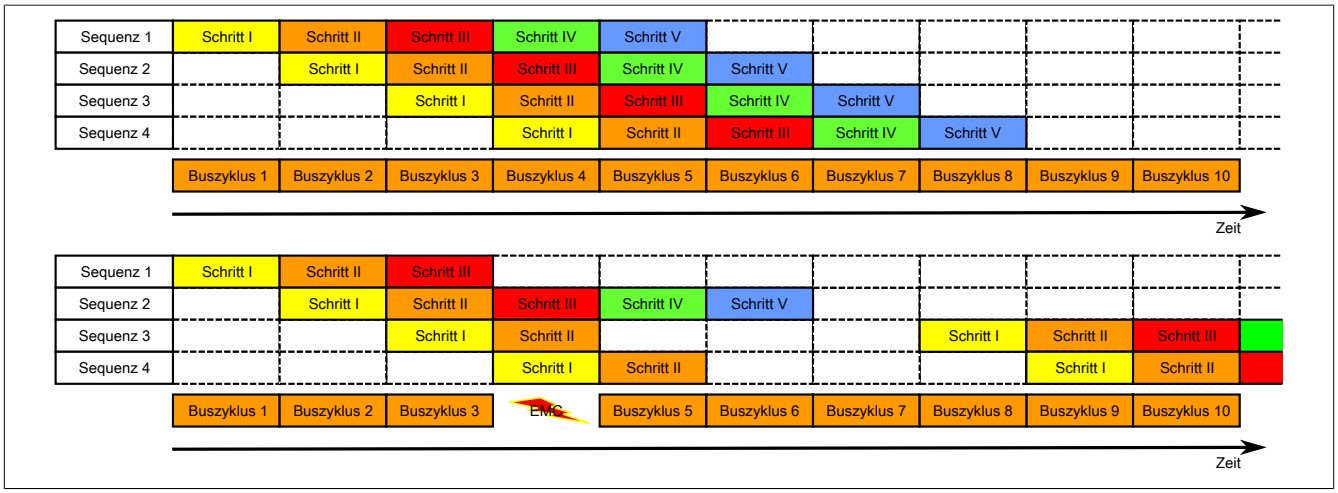


Abbildung 18: Auswirkung eines ausgefallenen Buszyklus

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.

11.6 Erforderliche Zykluszeit

Die Zykluszeit ist abhängig von der eingestellten Abtastrate, Datenauflösung und MTU-Größe. Sie ist so zu wählen, dass bei der Übertragung der Messwerte über den Flatstream kein Pufferüberlauf auftritt.