X20DS1928

1 Allgemeines

Das Modul ist mit 1 EnDat Geber Schnittstelle ausgestattet. Das Modul unterscheidet selbständig, ob Geber mit EnDat 2.1 oder EnDat 2.2 angeschlossen wurden. Mit dem Modul können sowohl Geber, die in B&R Servomotoren eingebaut sind, als auch Fremdachsengeber (Geber, die eine beliebige Maschinenbewegung abtasten) ausgewertet werden. Die Eingangssignale werden überwacht. Damit können Drahtbruch, Leitungsschluss und Ausfall der Geberversorgung erkannt werden.

- EnDat 2.1 und EnDat 2.2 Geber Schnittstelle
- Überwachung der Gebereingänge
- 5 VDC und GND für Geberversorgung
- NetTime-Zeitstempel: Positionszeit

EnDat Geber

EnDat ist ein von der Johannes Heidenhain GmbH (www.heidenhain.de) entwickelter Standard, der die Vorteile von absoluter und inkrementeller Positionsmessung in sich vereint und darüber hinaus noch einen schreib- und lesbaren Parameterspeicher im Geber zur Verfügung stellt. Durch die absolute Positionsmessung entfällt gewöhnlich die Referenzfahrt. Gegebenenfalls ist ein Multi-Turn-Geber einzusetzen. Um Kosten zu sparen, kann aber auch ein Single-Turn-Geber zusammen mit einem Referenzschalter verwendet werden. In diesem Fall muss allerdings eine Referenzfahrt durchgeführt werden.

NetTime-Zeitstempel der Position

Für hochdynamische Positionieraufgaben ist nicht nur der Positionswert bedeutend, sondern auch der exakte Zeitpunkt der Positionserfassung. Das Modul verfügt dafür über eine NetTime-Funktion, die die aufgenommene Position mit einem Mikrosekunden genauen Zeitstempel versieht.

Die Zeitstempelfunktion basiert auf synchronisierten Timern. Tritt ein Zeitstempelereignis auf, so speichert das Modul unmittelbar die aktuelle NetTime. Nach der Übertragung der jeweiligen Daten inklusive dieses exakten Zeitpunktes in die CPU kann diese nun, gegebenenfalls mit Hilfe ihrer eigenen NetTime (bzw. Systemzeit), die Daten auswerten.

2 Bestelldaten

Bestellnummer	Kurzbeschreibung	Abbildung
	Digitale Signalverarbeitung und -aufbereitung	
X20DS1928	X20 Digitales Signalmodul, 1 EnDat 2.1/2.2 Schnittstelle, NetTime-Funktion	
	Erforderliches Zubehör	
	Busmodule	
X20BM11	X20 Busmodul, 24 VDC codiert, interne I/O-Versorgung durchverbunden	
X20BM15	X20 Busmodul, mit Knotennummernschalter, 24 VDC codiert, interne I/O-Versorgung durchverbunden	
	Feldklemmen	
X20TB12	X20 Feldklemme, 12-polig, 24 VDC codiert	

Tabelle 1: X20DS1928 - Bestelldaten

3 Technische Daten

Bestellnummer	X20DS1928
Kurzbeschreibung	
I/O-Modul	1x EnDat-Schnittstelle
Allgemeines	
B&R ID-Code	0xA912
Statusanzeigen	Zählrichtung, Betriebszustand, Modulstatus
Diagnose	
Modul Run/Error	Ja, per Status-LED und SW-Status
Zählrichtung	Ja, per Status-LED
Leistungsaufnahme	
Bus	0,01 W
I/O-intern	1,3 W
Zusätzliche Verlustleistung durch Aktoren (ohmsch) [W]	-
Ausführung der Signalleitungen	Für alle Signalleitungen sind geschirmte Leitungen zu verwenden
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
Gebereingänge	
Art	EnDat 2.1/2.2
Winkelpositionsauflösung	13 Bit, bei einem Signal von 1 V_{SS}
Geberüberwachung	Ja
max. Geberkabellänge	10 m, bei einem Leiterquerschnitt 4x 2x 0,14 mm^2 und 1x 2x 0,5 mm^2
Sinus-Cosinus-Eingänge	
Signalübertragung	Differenzsignale, symmetrisch
Signalfrequenz	DC bis 400 kHz
Differenzspannung	1 V_{SS}
Gleichtaktspannung	max. ±10 V
Abschlusswiderstand	120 Ω
Geberversorgung	
Ausgangsspannung	5 V (±5%)
Belastbarkeit	300 mA
Schutzmaßnahmen	
überlastfest	Ja
kurzschlussfest	Ja
Serielle EnDat Schnittstelle	
Signalübertragung	5 VDC Differenzsignal, EiA RS-485 Standard
Übertragungsstatus	Siehe EnDat-Spezifikation
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 50°C
Derating	Siehe Abschnitt "Derating"
Lagerung	-40 bis 85°C
Transport	-40 bis 85°C

Tabelle 2: X20DS1928 - Technische Daten

Bestellnummer	X20DS1928
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: X20DS1928 - Technische Daten

4 Status-LEDs

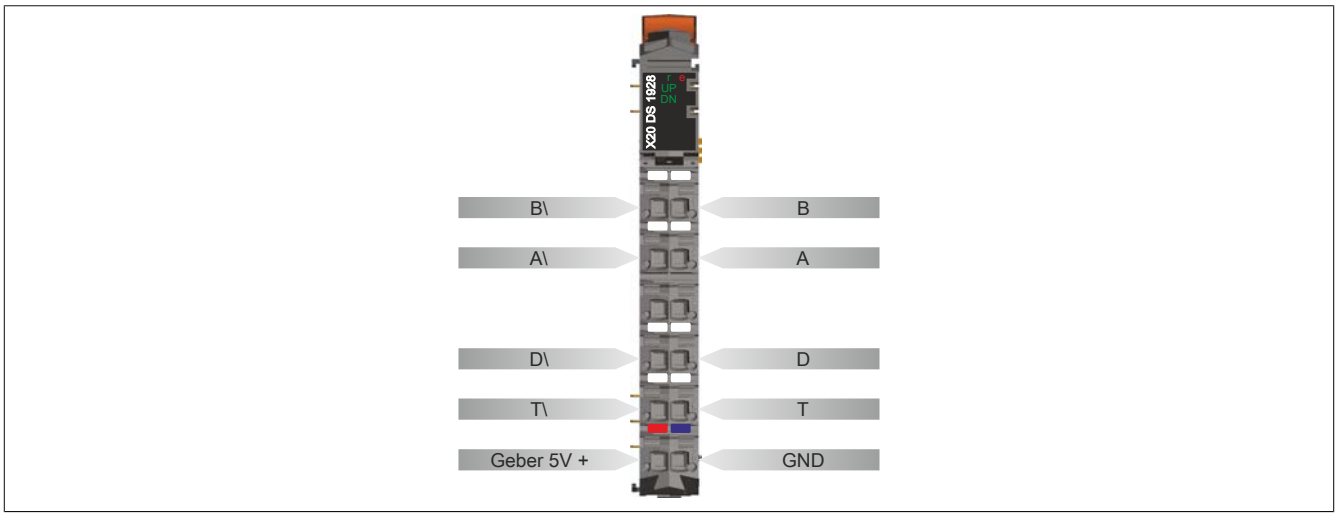
Für die Beschreibung der verschiedenen Betriebsmodi siehe X20 System Anwenderhandbuch, Abschnitt "Zusätzliche Informationen - Diagnose-LEDs".

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	Fehler- oder Resetzustand - mögliche Ursache: • Fehler der Geberversorgung
			Single Flash	I/O-Fehler - mögliche Ursachen: • Sinus/Cosinus relativer Positionsfehler (Drahtbruch) • Sinus/Cosinus absoluter Positionsfehler (Referenz)
			Double Flash	Systemfehler - mögliche Ursachen: • EnDat Kommunikationsfehler • EnDat Positionsfehler • EnDat Fehler beim Definieren der Parameter
			Triple Flash	I/O-Fehler und Systemfehler
			Single Flash invertiert	Fehler- oder Resetzustand und I/O-Fehler
			Double Flash invertiert	Fehler- oder Resetzustand und Systemfehler
			Triple Flash invertiert	Fehler- oder Resetzustand, I/O-Fehler und Systemfehler
	UP	Grün	Ein	Die LEDs "UP" und "DN" leuchten in Abhängigkeit von der Drehrichtung und der Drehzahl des angeschlossenen Gebers. Die LED "UP" zeigt eine Änderung der Geberposition in positiver Richtung an.
	DN	Grün	Ein	Die LED "DN" zeigt eine Änderung der Geberposition in negativer Richtung an.

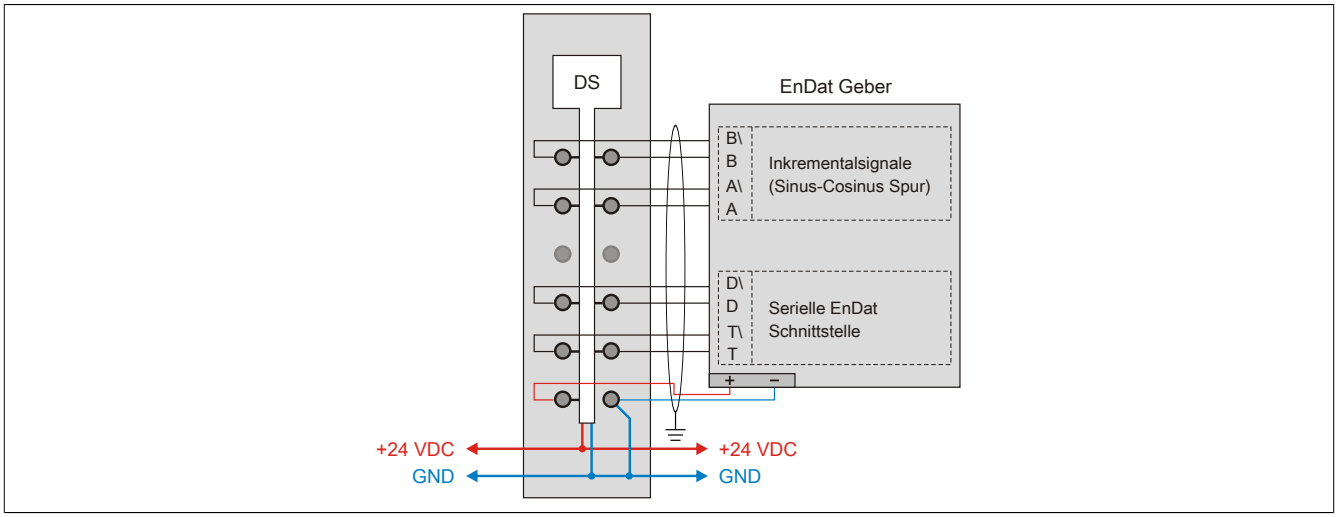
1) Je nach Konfiguration kann ein Firmware-Update bis zu mehreren Minuten benötigen.

5 Anschlussbelegung

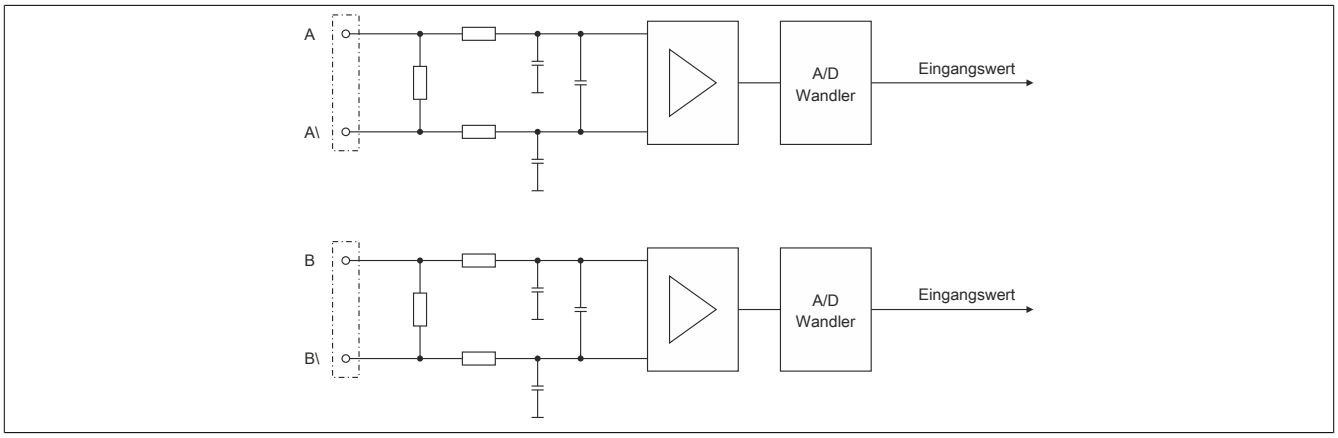
Für alle Signalleitungen sind geschirmte Leitungen zu verwenden.



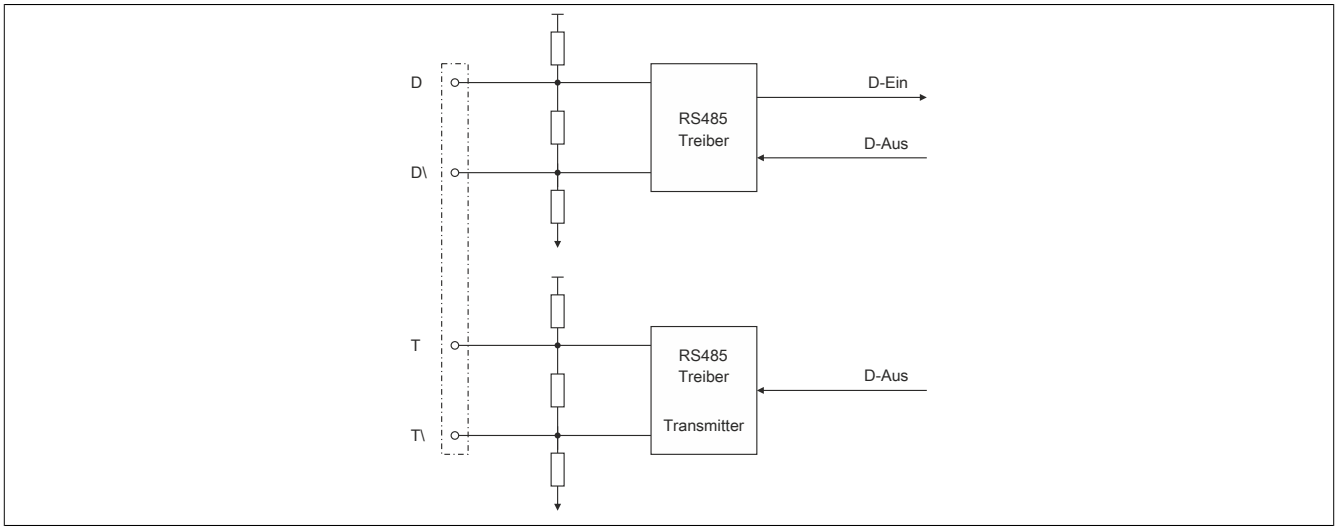
6 Anschlussbeispiel



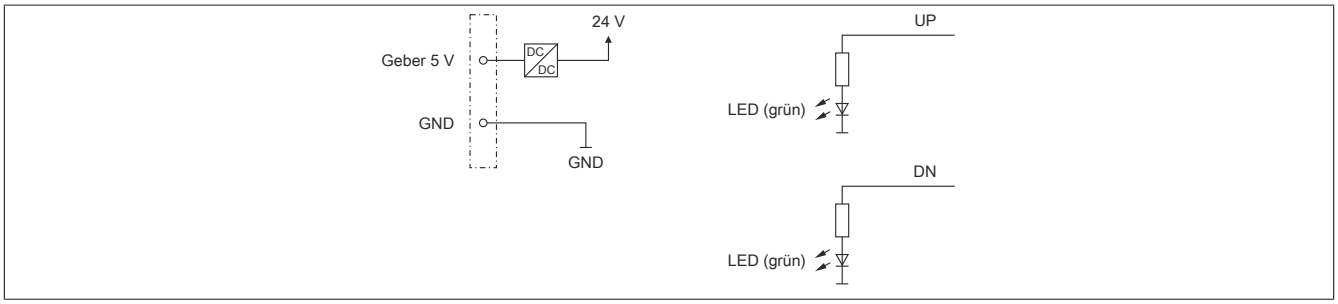
7 Eingangsschema der Inkrementalsignale (Sinus-Cosinus Spur)



8 Eingangsschema der seriellen EnDat Schnittstelle



9 Schema der Geberversorgung und LEDs

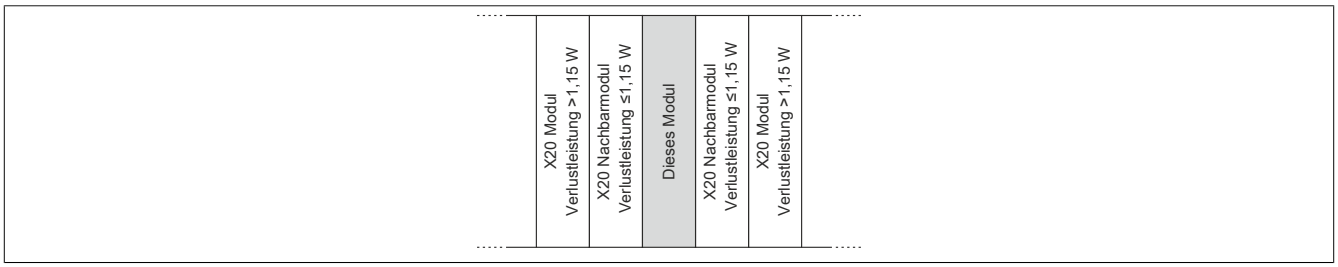


10 Derating

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.



11 Registerbeschreibung

11.1 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.

11.2 Registerübersicht - Funktionsmodell 0 (Standard)

Register	Bezeichnung	Datentyp	Read		Write	
			zyklisch	azyklisch	zyklisch	azyklisch
Modulkonfiguration						
513	CfO_SIframeGenID	USINT				•
654	CfO_SystemCyclePrescaler	UINT				•
Grundfunktionen						
683	SDCLifeCount	SINT	•			
4180	PositionHW	UDINT	•			
4188	PositionLW	UDINT	•			
	Position	DINT				
4172	PosTime (32-Bit)	DINT	•			
4174	PosTime (16-Bit)	INT	•			
4163	PosCycle	SINT	•			
Fehlermanagement						
389	ErrorEnableID_1710	USINT				•
261	ErrorInfo	USINT	•			
	EncoderSupplyError	Bit 0				
	VssCheckError	Bit 2				
	SinCosPosError	Bit 3				
	EnDatComError	Bit 4				
	EnDatPosError	Bit 5				
	EnDatParSetError	Bit 6				
	EnDatRefWarning	Bit 7				
325	AckErrorInfo	USINT			•	
	AckEncoderSupplyError	Bit 0				
	AckVssCheckError	Bit 2				
	AckSinCosPosError	Bit 3				
	AckEnDatComError	Bit 4				
	AckEnDatPosError	Bit 5				
	AckEnDatParSetError	Bit 6				
	AckEnDatRefWarning	Bit 7				
4352	EnDatError	UINT	•			
4353	EnDatWarning	UINT	•			
4099	EnDat-Fehler quittieren	USINT			•	
	EnDatAck	Bit 0				
Sin/Cos - Konfiguration						
1025	SinCosEnable	USINT				•
1027	SinCosRefSource	USINT				•
1034	SinCosVssMin	UINT				•
1038	SinCosVssMax	UINT				•
1044	SinCosQuitTime	UDINT				•
EnDat - Identifikation lesen						
4097	EnDatMode	USINT				•
4400 + N	OperatingParam_N (Index N = 00 bis 15)	UINT		•		
4352 + N	OperatingStatus_0N (Index N = 0 bis 3)	UINT		•		
4352 + N	ParamManuf_N (Index N = 04 bis 47)	UINT		•		
4416 + N	ParamManufEnDat22_N (Index N = 01 bis 63)	UINT		•		
EnDat - zusätzliche Informationen lesen						
4860 + N*8	EnDatInfoCmd0N (Index N = 1 bis 4)	UDINT				•
4935	Gültigkeit der Infodaten	USINT	•			
	EnDatInfoOK01	Bit 0				
	...	...				
	EnDatInfoOK04	Bit 3				
4978 + N*16	EnDatInfo0N (Index N = 1 bis 4)	UINT	•			
		INT				
Flatstream-Modus						
4609	OutputMTU	USINT				•
4611	InputMTU	USINT				•
4613	FlatStreamMode	USINT				•
4615	Forward	USINT				•

Register	Bezeichnung	Datentyp	Read		Write	
			zyklisch	azyklisch	zyklisch	azyklisch
4620	ForwardDelay	UINT				•
4672	InputSequence	USINT	•			
4672 + N	RxByteN (Index N = 1 bis 15)	USINT	•			
4704	OutputSequence	USINT			•	
4704 + N	TxByteN (Index N = 1 bis 15)	USINT			•	

11.3 Registerübersicht - Bus Controller Funktionsmodell 254

Register	Offset[1)]	Bezeichnung	Datentyp	Read		Write	
				zyklisch	azyklisch	zyklisch	azyklisch
Modulkonfiguration							
513	-	CfO_SIframeGenID	USINT				•
654	-	CfO_SystemCyclePrescaler	UINT				•
Grundfunktionen							
4180	0	PositionHW	UDINT	•			
4188	4	PositionLW	UDINT	•			
4163	15	PosCycle	SINT	•			
Fehlermanagement							
389	-	ErrorEnableID_1710	USINT				•
261	14	ErrorInfo	USINT	•			
		EncoderSupplyError	Bit 0				
		VssCheckError	Bit 2				
		SinCosPosError	Bit 3				
		EnDatComError	Bit 4				
		EnDatPosError	Bit 5				
		EnDatParSetError	Bit 6				
		EnDatRefWarning	Bit 7				
325	6	AckErrorInfo	USINT			•	
		AckEncoderSupplyError	Bit 0				
		AckVssCheckError	Bit 2				
		AckSinCosPosError	Bit 3				
		AckEnDatComError	Bit 4				
		AckEnDatPosError	Bit 5				
		AckEnDatParSetError	Bit 6				
		AckEnDatRefWarning	Bit 7				
4352	-	EnDatError	UINT		•		
4353	-	EnDatWarning	UINT		•		
4099	-	EnDat-Fehler quittieren	USINT				•
		EnDatAck	Bit 0				
Sin/Cos - Konfiguration							
1025	-	SinCosEnable	USINT				•
1027	-	SinCosRefSource	USINT				•
1034	-	SinCosVssMin	UINT				•
1038	-	SinCosVssMax	UINT				•
1044	-	SinCosQuitTime	UDINT				•
EnDat - Identifikation lesen							
4097	-	EnDatMode	USINT				•
4400 + N	-	OperatingParam_N (Index N = 00 bis 15)	UINT		•		
4352 + N	-	OperatingStatus_0N (Index N = 0 bis 3)	UINT		•		
4352 + N	-	ParamManuf_N (Index N = 04 bis 47)	UINT		•		
4416 + N	-	ParamManufEnDat22_N (Index N = 01 bis 63)	UINT		•		
EnDat - zusätzliche Informationen lesen							
4860 + N*8	-	EnDatInfoCmd0N (Index N = 1 bis 4)	UDINT				•
4935	-	Gültigkeit der Infodaten	USINT		•		
		EnDatInfoOK01	Bit 0				
		...	...				
		EnDatInfoOK04	Bit 3				
4978 + N*16	-	EnDatInfo0N (Index N = 1 bis 4)	UINT		•		
			INT				
Flatstream-Modus							
4609	-	OutputMTU	USINT				•
4611	-	InputMTU	USINT				•
4613	-	FlatStreamMode	USINT				•
4615	-	Forward	USINT				•
4620	-	ForwardDelay	UINT				•
4672	8	InputSequence	USINT	•			
4672 + N	9 - 13	RxByteN (Index N = 1 bis 5)	USINT	•			
4704	0	OutputSequence	USINT			•	
4704 + N	1 - 5	TxByteN (Index N = 1 bis 5)	USINT			•	

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

11.3.1 Verwendung des Moduls am Bus Controller

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

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

11.3.2 CAN-I/O Bus Controller

Das Modul belegt an CAN-I/O 2 analoge logische Steckplätze.

11.4 Modulkonfiguration

Mit Hilfe der folgenden Konfigurationsregister kann der Anwender verschiedene Moduleinstellungen festlegen. Auf diese Weise wird z. B. das Verhalten am X2X-Link beeinflusst. Hier stehen dem Nutzer 2 Optionsregister zur Verfügung.

11.4.1 Datenabfrage

Name:
CfO_SIframeGenID

Mit diesem Register legt der Anwender den Zeitpunkt fest, zu dem die synchronen/zyklischen Eingangsdaten generiert werden. Für eine jitterfreie Datenbeschaffung ist X2X-Zyklus optimiert einzustellen, für die beste Performance reaktionsschnell.

Datentyp	Werte	Information
USINT	9	Reaktionsschnell
	14	X2X-Zyklus optimiert; Bus Controller Default

11.4.2 Vorteiler

Name:
CfO_SystemCyclePrescaler

Damit das Modul sowohl mit der CPU als auch mit dem EnDat-Geber kommunizieren kann, muss die EnDat-Zykluszeit mindestens das Zweifache der Modulzykluszeit betragen. Die tatsächliche EnDat-Zykluszeit ergibt sich durch Multiplikation der Modulzykluszeit mit dem Wert im diesem Register.

Datentyp	Werte	Information
UINT	2	EnDat-Zyklus: 200 bis 400 µs; Bus Controller Default
	4	EnDat-Zyklus: 400 bis 800 µs
	8	EnDat-Zyklus: 800 bis 1600 µs

11.5 Grundfunktionen

Das Modul kann in Zusammenarbeit mit einem EnDat-Geber eine Position einlesen. Die empfangenen Daten werden in 2 unterschiedlichen Formaten aufbereitet und mit einem Zeitstempel versehen. Es stehen 6 Register für die Weiterverarbeitung zur Verfügung. Auf diese Weise kann der Anwender frei wählen, welches Format für seine individuelle Anwendung am besten geeignet ist.

11.5.1 SDC-Zählerregister

Name:
SDCLifeCount

Das 8-Bit-Zählregister wird für das SDC-Softwarepaket benötigt. Es wird entsprechend dem Systemtakt inkrementiert, damit der SDC die Gültigkeit des Datenframes prüfen kann.

Datentyp	Werte
SINT	-128 bis 127

11.5.2 Absolute Positionswerte

Name:
PositionHW
PositionLW

Die Absolutposition des Gebers wird mit 64-Bit aufgelöst. Der Positionswert wird in den Registern PositionHW und PositionLW abgelegt. Die oberen 32-Bit stehen im Register PositionHW und die unteren 32-Bit im Register PositionLW.

Bei SinCos-Signalauswertung siehe "Format des SinCos-Signals" auf Seite 15 für Information über das Datenformat.

Datentyp	Werte
2x UDINT	0 bis 4.294.967.295

11.5.3 SDC-Positionswert

Name:
Position

Die SDC-Library verlangt die Position als vorzeichenbehafteten 32-Bit Wert. Zu diesem Zweck kann das Low Word der Position separat angesprochen werden. Der Wert kann aber auch als Standardpositionswert verwendet werden.

Bei SinCos-Signalauswertung siehe "Format des SinCos-Signals" auf Seite 15 für Information über das Datenformat.

Datentyp	Werte
DINT	-2.147.483.648 bis 2.147.483.647

11.5.4 NetTime der Positionswerte

Name:
PosTime

In diesem Register wird jeder ermittelten Position der aktuelle Wert der NetTime zugeordnet. Die NetTime wird dabei µs-genau erfasst.
Die Verwendung der SDC-Library erfordert einen 16-Bit Wert. Der Wert der NetTime wird deshalb auch in diesem Format aufbereitet.

Für weitere Informationen zu NetTime und Zeitstempel siehe "NetTime Technology" auf Seite 54.

Datentyp	Werte	Informaton
DINT	-2.147.483.648 bis 2.147.483.647	NetTime in µs
INT	-32768 bis 32767	

11.5.5 Zähler für Positionswerte

Name:
PosCycle

Der PosCycle ist ein rundlaufender Zähler und wird inkrementiert, sobald das Modul einen neuen gültigen Positionswert ermittelt hat.

Datentyp	Werte
SINT	-128 bis 127

11.6 Fehlermanagement

Das Modul ist in der Lage Fehlerzustände zu diagnostizieren. Hier werden die Fehlerdiagnosen auf folgenden Wegen durchgeführt:

- "Modulbezogene Diagnose" auf Seite 10
- "EnDat-bezogene Diagnose" auf Seite 13

11.6.1 Modulbezogene Diagnose

Das Modul diagnostiziert 7 verschiedene Fehler bzw. Warnungen. Je nach Einstellung können die Fehlerbits entweder einzeln oder gepackt abgerufen werden.

11.6.1.1 Fehler und Warnungen konfigurieren

Name:
ErrorEnableID_1710

In diesem Register können die implementierten Diagnosealgorithmen an- bzw. abgeschaltet werden.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	255

Bitstruktur :

Bit	Bezeichnung	Wert	Information
0	Geberversorgung:	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert (Bus Controller Default)
1	Reserviert	-	
2	Vss Sin/Cos:	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert (Bus Controller Default)
3	Positionsfehler:	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert (Bus Controller Default)
4	EnDat - Kommunikation:	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert (Bus Controller Default)
5	EnDat - Position:	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert (Bus Controller Default)
6	EnDat - Parameter:	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert (Bus Controller Default)
7	EnDat - Referenzwarnung:	0	Warnung deaktiviert
		1	Warnung aktiviert (Bus Controller Default)

Geberversorgung:

Spannungsversorgung des Gebers ist unzulässig niedrig.

Vss Sin/Cos:

Spannungswert für Sin/Cos-Spur verstößt gegen konfigurierte Grenzwerte
→ siehe Register "SinCosVssMin" auf Seite 16 bzw. "SinCosVssMax" auf Seite 16

Positionsfehler:

Ermittelter Positionswert verstößt gegen Anforderungen der Applikation.

EnDat - Kommunikation:

Kommunikationsfehler auf der EnDat-Schnittstelle (z. B. unkorrekte Prüfsumme).

EnDat - Position:

Geber bewertet ermittelten Positionswert als ungültig.

EnDat - Parameter:

Registerwerte zur Geber-Identifikation sind inkonsistent
→ Gegenmaßnahmen: Verdrahtung prüfen oder Rescan durchführen (siehe "EnDatAck" auf Seite 14)

EnDat - Referenzwarnung:

Die digitale Schnittstelle liefert einen absoluten Positionswert, mit dem die Lage der Achse genau beschrieben werden kann. Zu Beginn einer Messung wird deshalb der Positionswert auf diesen absoluten Wert referenziert. Über das analoge Interface können sehr schnelle Änderungen inkrementell erfasst werden. Auf diese Weise kann das Modul den Positionswert hochauflösend weiterzählen. Sowohl das analoge als auch das digitale Signal werden zyklisch eingelesen. Weicht während des Betriebes der inkrementell ermittelte Wert vom absoluten Wert ab, muss die Position erneut referenziert werden und die Referenzwarnung wird angezeigt.

11.6.1.2 Status der Fehler und Warnungen

Name:
ErrorInfo

EncoderSupplyError
VssCheckError
PositionError
EnDatComError
EnDatPosError
EnDatParSetError
EnDatRefWarning

Dieses Register zeigt an, welcher Fehler bzw. welche Warnung gerade auftritt. Für eine Beschreibung der Fehler siehe "Fehler und Warnungen konfigurieren" auf Seite 10

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	EncoderSupplyError	0	Fehler nicht vorhanden
		1	Fehler der Geberversorgung
1	Reserviert	-	
2	VssCheckError	0	Fehler nicht vorhanden
		1	Vss-Fehler Sin/Cos-Spur
3	PositionError	0	Fehler nicht vorhanden
		1	Positionsfehler
4	EnDatComError	0	Fehler nicht vorhanden
		1	Kommunikationsfehler EnDat
5	EnDatPosError	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert
6	EnDatParSetError	0	Fehlererkennung deaktiviert
		1	Fehlererkennung aktiviert
7	EnDatRefWarning	0	Warnung deaktiviert
		1	Warnung aktiviert

11.6.1.3 Fehler und Warnungen quittieren

Name:
AckErrorInfo

AckEncoderSupplyError
AckVssCheckError
AckPositionError
AckEnDatComError
AckEnDatPosError
AckEnDatParSetError
AckEnDatRefWarning

Dieses Register dient der Quittierung einer im Register "Status der Fehler und Warnungen" auf Seite 11 aufgetretenen Fehlermeldung. Für eine Beschreibung der Fehler siehe "Fehler und Warnungen konfigurieren" auf Seite 10

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	AckEncoderSupplyError	0	Keine Fehlerquittierung
		1	Fehlerquittierung
1	Reserviert	-	
2	AckVssCheckError	0	Keine Fehlerquittierung
		1	Fehlerquittierung
3	AckPositionError	0	Keine Fehlerquittierung
		1	Fehlerquittierung
4	AckEnDatComError	0	Keine Fehlerquittierung
		1	Fehlerquittierung
5	AckEnDatPosError	0	Keine Fehlerquittierung
		1	Fehlerquittierung
6	AckEnDatParSetError	0	Keine Fehlerquittierung
		1	Fehlerquittierung
7	AckEnDatRefWarning	0	Keine Quittierung
		1	Quittierung

11.6.2 EnDat-bezogene Diagnose

Im EnDat-Standard sind bereits Speicherbereiche für Fehlerbehandlungen vorgesehen. Um die Fehlererkennung laut EnDat-Standard zu nutzen, wurde das Fehlermanagement angepasst. Es wurden zusätzliche Register im Modul implementiert, die diese Bereiche des Geberspeichers aufbereiten.
Das Modul ermöglicht den Zugriff auf alle bisher definierten Speicherbereiche zur Fehlerbehandlung. Die Speicherbereiche werden auf die Register im Modul gespiegelt und können vom Anwender interpretiert werden.
Genaue Informationen, welche Fehler auf diese Weise erkannt werden, müssen im Handbuch des Gebers nachgeschlagen werden.

11.6.2.1 Endat-Fehler

Name:
EnDatError

Mit Hilfe dieses Registers werden kritische Zustände des EnDat-Gebers angezeigt. Das System arbeitet in der Regel nicht mehr und muss gewartet werden.

Datentyp	Werte
UINT	siehe Bitstruktur

Die im Folgenden beschriebene Bitstruktur orientiert sich an der allgemeinen Empfehlung laut EnDat-Standard. Die Spezifikation lässt offen, welche Auslösealgorithmen genutzt und welche der aufgelisteten Meldungen unterstützt werden müssen. Einzelheiten sind dem Handbuch des Gebers zu entnehmen.

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	Beleuchtung	0	Ok
		1	Ausgefallen
1	Signalamplitude	0	Ok
		1	Fehlerhaft
2	Positionswert	0	Ok
		1	Fehlerhaft
3	Überspannung	0	Nein
		1	Ja
4	Unterspannung	0	Nein
		1	Ja
5	Überstrom	0	Nein
		1	Ja
6	Batterie	0	Ok
		1	Wechsel nötig
7 - 15	Reserviert	-	

11.6.2.2 EnDat-Warnungen

Name:
EnDatWarning

Mit Hilfe dieses Registers werden kritische Zustände des EnDat-Gebers angezeigt. Der Geber ist zwar noch einsatzfähig, sollte aber umgehend überprüft werden. In der Regel liegt eine Überschreitung von vorgegebenen Toleranzen vor.

Datentyp	Werte
UINT	siehe Bitstruktur

Die im Folgenden beschriebene Bitstruktur orientiert sich an der allgemeinen Empfehlung laut EnDat-Standard. Die Spezifikation lässt offen, welche Auslösealgorithmen genutzt und welche der aufgelisteten Meldungen unterstützt werden müssen. Einzelheiten sind dem Handbuch des Gebers zu entnehmen.

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	Frequenzkollision	0	Nein
		1	Ja
1	Temperaturüberschreitung	0	Nein
		1	Ja
2	Regelreserve - Beleuchtung	0	Nicht benötigt
		1	benötigt
3	Ladung - Batterie	0	Ok
		1	Niedrig
4	Referenzpunkt	0	Erreicht
		1	Nicht erreicht
5 - 15	Reserviert	-	

11.6.2.3 EnDat-Fehler quittieren

Name:
EnDatAck

Das "EnDatAck" quittiert alle Fehler bzw. Warnungen aus den Registern "EnDatError" auf Seite 13 und "EnDat-Warning" auf Seite 14. Außerdem kann es das Modul anweisen die Parameter zur Identifikation neu einzulesen.

Wird eines der Bits in diesem Register gesetzt, wird das Bit vom System automatisch zurückgesetzt und der dazugehörige Algorithmus ausgeführt.

Datentyp	Werte
USINT	siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	EnDatAck	0	Keine Quittierung
		1	Quittieren
1	Rescan - Identifikationsregister	0	Eingelesene Parameter beibehalten
		1	Parameter neu einlesen
2 - 7	Reserviert	-	

11.7 Sin/Cos - Konfiguration der analogen Schnittstelle

Das Modul verfügt neben dem digitalen EnDat auch über eine analoge Schnittstelle zur Erfassung eines differenziellen Sinus-Cosinus-Signals. Zur Steigerung der Auflösung sieht der EnDat-Standard eine Kooperation der analogen und der digitalen Information vor. Auf diese Weise kann die Position sehr dynamisch und gleichzeitig mit hoher Auflösung abgebildet werden.

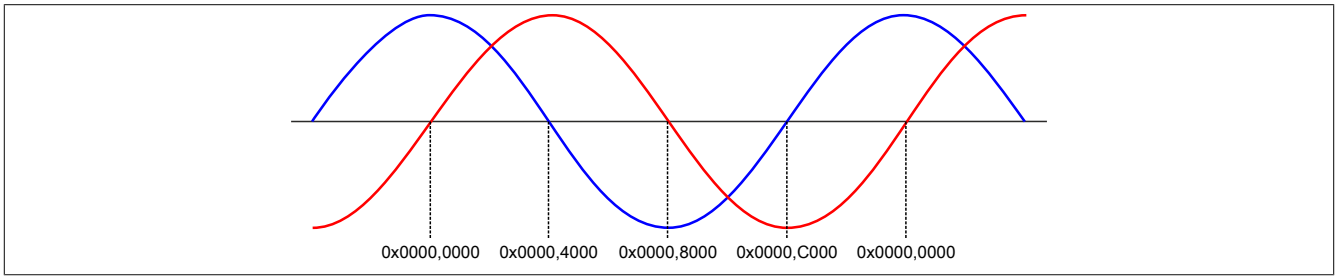
11.7.1 Format des SinCos-Signals

In den Registern "Absolute Positionswerte" auf Seite 9 und "SDC-Positionswert" auf Seite 9 wird das SinCos-Signal als Positionswert dargestellt. Dabei gilt folgender Zusammenhang:

- PositionLW und Position sind in der Funktion identisch.
- PositionHW erweitert den Ganzzahlenbereich von PositionLW um zusätzliche Multiturn-Funktionalität.

<table>
<tr><td>64-Bit Register</td><td>PositionHW
(ohne Vorzeichen)</td><td colspan="2">PositionLW
(ohne Vorzeichen)</td></tr>
<tr><td>32-Bit Register</td><td>-</td><td colspan="2">Position (vorzeichenbehaftet)</td></tr>
<tr><td>Format</td><td>Ganzzahlerweiterung
(auf 48-Bit)</td><td>Ganzzahl
(16-Bit)</td><td>Nachkommastellen
(mit 13-Bit Auflösung)</td></tr>
<tr><td>Information</td><td></td><td>Eine volle Sinuswelle entspricht einem Inkrement der Ganzzahl.</td><td>
<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>0</td><td>0</td><td>0</td></tr>
</table>
Achtung: Die unteren 3 Bit enthalten immer den Wert 0.
</td></tr>
<tr><td>Word/DWord</td><td>DWord</td><td>Word 1</td><td>Word 0</td></tr>
</table>

Zusammenhang zwischen Sinuskurve (rot) und Nachkommastellen:



11.7.2 SinCos aktivieren

Name:
SinCosEnable

Dieses Register muss aus Konfigurationsgründen immer mit dem Wert 1 belegt werden.

Datentyp	Werte	Information
USINT	1	Bus Controller Default: 1

11.7.3 SinCos Referenzquelle aktivieren

Name:
SinCosRefSource

Dieses Register muss aus Konfigurationsgründen immer mit dem Wert 1 belegt werden.

Datentyp	Werte	Information
USINT	1	Bus Controller Default: 1

11.7.4 Unteren Vss-Wert konfigurieren

Name:
SinCosVssMin

Dieses Register gibt den zulässigen unteren Grenzwert für die Spitzen-Spitzen-Spannung der Sinus/Cosinus-Spur vor. Auf diese Weise wird das anstehende Signal überwacht. Unterschreitet der eingehende Wert diese Vorgabe, meldet das Modul den entsprechenden Fehler.

Datentyp	Werte	Information
UINT	0 bis 1500	Werte in mV Bus Controller Default: 800

11.7.5 Oberen Vss-Wert konfigurieren

Name:
SinCosVssMax

Dieses Register gibt den zulässigen oberen Grenzwert für die Spitzen-Spitzen-Spannung der Sinus/Cosinus-Spur vor. Auf diese Weise wird das anstehende Signal überwacht. Überschreitet der eingehende Wert diese Vorgabe, meldet das Modul den entsprechenden Fehler.

Datentyp	Werte	Information
UINT	0 bis 1500	Werte in mV Bus Controller Default: 1200

11.7.6 Wartezeit nach Fehler konfigurieren

Name:
SinCosQuitTime

Wenn ein Fehler auf der analogen Schnittstelle erkannt wird, bleiben die letzten korrekt ermittelten Werte weiterhin gültig. In diesem Register kann eine Zeitspanne eingestellt werden, in der das Modul nach dem Fehlerzustand wieder korrekte Werte empfängt, ohne sie intern weiterzuverarbeiten. Erst danach werden neu eingelesene korrekte Analogwerte als gültig anerkannt.

Datentyp	Werte	Information
UDINT	0 bis 20.000.000	Werte in µs Bus Controller Default: 100000

11.8 EnDat

11.8.1 EnDat - Konfiguration der digitalen Schnittstelle

Mittels der EnDat-Schnittstelle ist es möglich eine Punkt-zu-Punkt-Verbindung zu genau einem EnDat-Geber aufzubauen.

Um die Geber-Daten im SPS-Programm zu verwenden, stehen 2 verschiedene Methoden zur Auswahl. Zum einen können die wesentlichen Geberwerte im Modul zwischengespeichert werden und stehen im Anschluss der CPU zur Verfügung. Zum anderen bietet das Modul den sogenannten Flatstream-Modus an, der den gesamten Kommandoumfang gemäß EnDat-Spezifikation unterstützt.

Detaillierte Informationen zur EnDat-Spezifikation sind im Dokument "Technische Information – EnDat 2.2" enthalten.

11.8.1.1 EnDat-Moduleigenschaften konfigurieren

Name:
EnDatMode

Mit Hilfe dieses Registers werden verschiedene Moduleigenschaften vordefiniert.

Datentyp	Werte	Bus Controller Default
USINT	siehe Bitstruktur	0

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	EnDat-Schnittstelle	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
1	Format der eingelesenen Positionsdaten	0	Nicht vorzeichenbehaftet (Bus Controller Default)
		1	Vorzeichenbehaftet
2	schneller EnDat-Takt (6 MHz)	0	Aktiviert, wenn Geber kompatibel (Bus Controller Default)
		1	Deaktiviert
3	Sin/Cos-Spur	0	Aktiviert (Bus Controller Default)
		1	Deaktiviert
4 - 7	Reserviert	-	

11.8.2 EnDat - Identifikation lesen

Mit Hilfe der EnDat-Schnittstelle ist der Anwender nicht nur in der Lage die Achsposition zu bestimmen. Er kann auch bestimmte Daten auslesen, die im Speicher des Gebers hinterlegt sind.
In der EnDat-Spezifikation wird der Geberspeicher in logische Gruppen unterteilt. Es werden u. a. Speicherbereiche für die Betriebsparameter (operating parameter), den Betriebszustand (operating status), die Herstellerparameter (parameter of manufacturer), und die Herstellerparameter laut EnDat 2.2 (parameter of manufacturer according to EnDat 2.2) unterschieden.
Die 4 wichtigsten Speicherbereiche werden auf Modulregister gespiegelt. Im Anschluss können die Informationen in der Applikation abgerufen und zur Identifikation eines speziellen Gebers genutzt werden.

Information:

EnDat existiert in verschiedenen Ausprägungen. Dies ist unbedingt zu beachten. Es wurde stets versucht EnDat durch neue technische Möglichkeiten zu erweitern und trotzdem abwärtskompatibel zu gestalten. Der Standard wurde mehrfach weiterentwickelt und ist deshalb nicht einheitlich strukturiert.

Grundsätzlich erfolgt die Abfrage der Speicherdaten zur Identifikation beim Start des Moduls. Zusätzlich können die Daten über das Register "EnDatAck" auf Seite 14 erneut eingelesen werden. Das Modul liest die Daten aus dem Geber ein und stellt ein Abbild davon für die SPS bereit.

11.8.2.1 Betriebsparameter

Name:
OperatingParam_00 bis OperatingParam_15

Mit Hilfe dieser Register können die aktuellen Betriebsparameter ausgelesen werden. Die Daten in diesen Registern entsprechen exakt den Werten im Geber. Genauere Informationen sind dem Handbuch des Gebers oder der aktuellen EnDat-Spezifikation zu entnehmen.

Datentyp	Werte
16x UINT	siehe Handbuch des Gebers

11.8.2.2 Betriebszustand

Name:
OperatingStatus_00 bis OperatingStatus_03

Mit Hilfe dieser Register kann der aktuelle Betriebszustand des Gebers ausgelesen werden. Die ersten 2 Register aus dieser Gruppe sind identisch zu den Registern "EnDatError" auf Seite 13 und "EnDatWarning" auf Seite 14. Sie besitzen aus diesem Grund eine Sonderstellung, da sie zyklisch aktualisiert werden.
In den Registern 02 und 03 werden Informationen zum Schreibschutz und zur sonstigen Konfiguration verwaltet. Die Daten in diesen Registern entsprechen exakt den Werten im Geber.
Genauere Informationen sind dem Handbuch des Gebers oder der aktuellen EnDat-Spezifikation zu entnehmen.

Datentyp	Werte
4x UINT	siehe Handbuch des Gebers

11.8.2.3 Herstellerparameter

Name:
ParamManuf_04 bis ParamManuf_47

Diese Register sind dafür bestimmt, die Herstellerparameter so aufzubereiten, wie es im EnDat-Standard 2.1 vorgesehen ist. Die genaue Anordnung der Informationen ist z. B. in der Dokumentation "Technische Information - EnDat 2.2" beschrieben.

Datentyp	Werte
44x UINT	siehe "Technische Information - EnDat 2.2" bzw. Herstellerdaten des Gebers

11.8.2.4 Zusätzliche Herstellerparameter laut EnDat 2.2

Name:
ParamManufEnDat22_00 bis ParamManufEnDat22_63

Diese Register sind dafür bestimmt, die Herstellerparameter so aufzubereiten, wie es im EnDat-Standard 2.2 vorgesehen ist. Die genaue Anordnung der Informationen ist in der Dokumentation "Technische Information - EnDat 2.2" beschrieben.

Datentyp	Werte
64x UINT	siehe "Technische Information - EnDat 2.2" bzw. Herstellerdaten des Gebers

11.8.3 EnDat - zusätzliche Informationen lesen

Neben den Identifikationsdaten können noch weitere Informationen aus dem Geber abgerufen werden. Der nachfolgend beschriebene Algorithmus benötigt allerdings genaue Kenntnisse über den Speicheraufbau des Gebers bzw. die EnDat-Spezifikation.

Konfiguration

Es stehen 4 verschiedene Kanäle zur Verfügung, die während eines Zyklus bedient werden können. Jeweils ein Register je Kanal dient zur Konfiguration, d. h. es steuert, welche Daten aus dem Geber ausgelesen und auf das dazugehörige Info-Byte gespiegelt werden.

11.8.3.1 EnDat-Kommandos senden

Name:
EnDatInfoCmd01 bis EnDatInfoCmd04

Über diese Register wird kanalweise gesteuert, welche Daten auf dem dazugehörigen Infobyte aufbereitet werden. Das Register besteht aus bis zu 4 Einzelinformationen mit jeweils 8-Bit.

Datentyp	Werte	Bus Controller Default
4x UDINT	siehe Bitstruktur	0

Bitstruktur:

Bit	Bezeichnung	Information	
0 - 7	Kommando	Auswahl des Antwortabschnitts	
8 - 15	Codes des Speicherbereichs	MRS-Code	
		ungeblockter Parameter	**blockweise angeordneter Parameter**
16 - 23	Speicher-ID	Parameternummer	Blocknummer
24 - 31	Speicher-ID	-	Parameternummer

Bei der Abfrage von Daten aus dem Geber ist grundsätzlich zu unterscheiden, ob ein Befehl aus der EnDat 2.1 oder der EnDat 2.2 Spezifikation angewendet wird.
Bei der Abfrage von Geberdaten mit Hilfe eines EnDat 2.1 Befehls (0x04 und 0x06) muss zusätzlich zum MRS-Code die Parameter- und optional die Blocknummer angegeben werden.
Bei einer Abfrage des Speichers mit Hilfe eines EnDat 2.2 Befehls wird auf die Parameter- und die Blocknummer verzichtet. Das Modul sendet nacheinander alle 4 Worte des Speicherbereichs, der mit Hilfe des MRS-Codes ausgewählt wurde. Je nachdem welches der 4 Antwortbytes benötigt wird, muss der passende Befehl genutzt werden.

Codes des Speicherbereichs

Der einzustellende Code entspricht exakt dem MRS-Code für den Geberspeicher. Die EnDat-Spezifikation lässt bisher einige Speicherbereiche im Geber undefiniert, um spätere Weiterentwicklungen einpflegen zu können. Aus diesem Grund ist an dieser Stelle keine zuverlässige detaillierte Erklärung möglich.
Genauere Informationen sind dem Handbuch des Gebers oder der aktuellen EnDat-Spezifikation zu entnehmen.

Parameternummer

Um den gewünschten Parameter im Geberspeicher exakt anzusprechen, muss laut EnDat 2.1 die entsprechende Parameternummer angegeben werden. In älteren EnDat-Versionen wurde der Geberspeicher nicht in Blöcke unterteilt. Aus diesem Grund gibt es Speicherbereiche, die ohne die Angabe einer Blocknummer selektiert werden können. Die Parameternummer muss in diesem Fall auf dem dritten Byte eingetragen werden.
Genauere Informationen sind dem Handbuch des Gebers oder der aktuellen EnDat-Spezifikation zu entnehmen.

Blocknummer

Um den Adressbereich des Geberspeichers zu erweitern, wurden ab der zweiten Sektion des Geberspeichers zusätzlich Blocknummern eingeführt. Befindet sich der gewünschte Parameter in diesem "geblockten" Bereich, muss die Blocknummer auf dem dritten Byte angegeben werden. Die Parameternummer wird in diesem Fall auf dem vierten Byte mitgeteilt.
Genauere Informationen sind dem Handbuch des Gebers oder der aktuellen EnDat-Spezifikation zu entnehmen.

Abruf

Nach der korrekten Konfiguration wird der Positionswert zyklisch in das Modul übertragen. Als Zwischenspeicher stehen 2 Register je Kanal zur Verfügung. Das Modul bestätigt den erfolgreichen Empfang durch Setzen eines OK-Bits. Die EnDat-Spezifikation lässt offen, welches Format der empfangene Parameter besitzt. Das Modul stellt deshalb die Informationen in 2 Varianten zur Verfügung. Welches der beiden Register zur Weiterverarbeitung genutzt werden muss, hängt vom ausgelesenen Parameter ab.

11.8.3.2 Gültigkeit der Infodaten

Name:
EnDatInfoOK01 bis EnDatInfoOK04

Die Bits dieses Registers geben Auskunft über die Gültigkeit der aktuellen Infodaten im Zwischenspeicher.

Datentyp	Werte
USINT	siehe Bitstruktur

Bitstruktur:

Bit	Bezeichnung	Wert	Information
0	EnDatInfoOK01	0	Information 01 ungültig
		1	Information 01 gültig
...	...	...	
3	EnDatInfoOK04	0	Information 04 ungültig
		1	Information 04 gültig
4 - 7	Reserviert	-	

11.8.3.3 EnDat-Information lesen

Name:
EnDatInfo01 bis EnDatInfo04

Die Register liefern die jeweils geforderte Information als vorzeichenlosen oder -behafteten 2 Byte Wert.
Die EnDat-Spezifikation lässt offen, welches Format die empfangenen Parameter besitzen. Welches der beiden Datentypen zur Weiterverarbeitung genutzt werden muss, hängt deshalb vom ausgelesenen Parameter ab.

Datentyp	Werte
UINT	0 bis 65535
INT	-32768 bis 32767

11.9 Die Flatstream-Kommunikation

11.9.1 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.

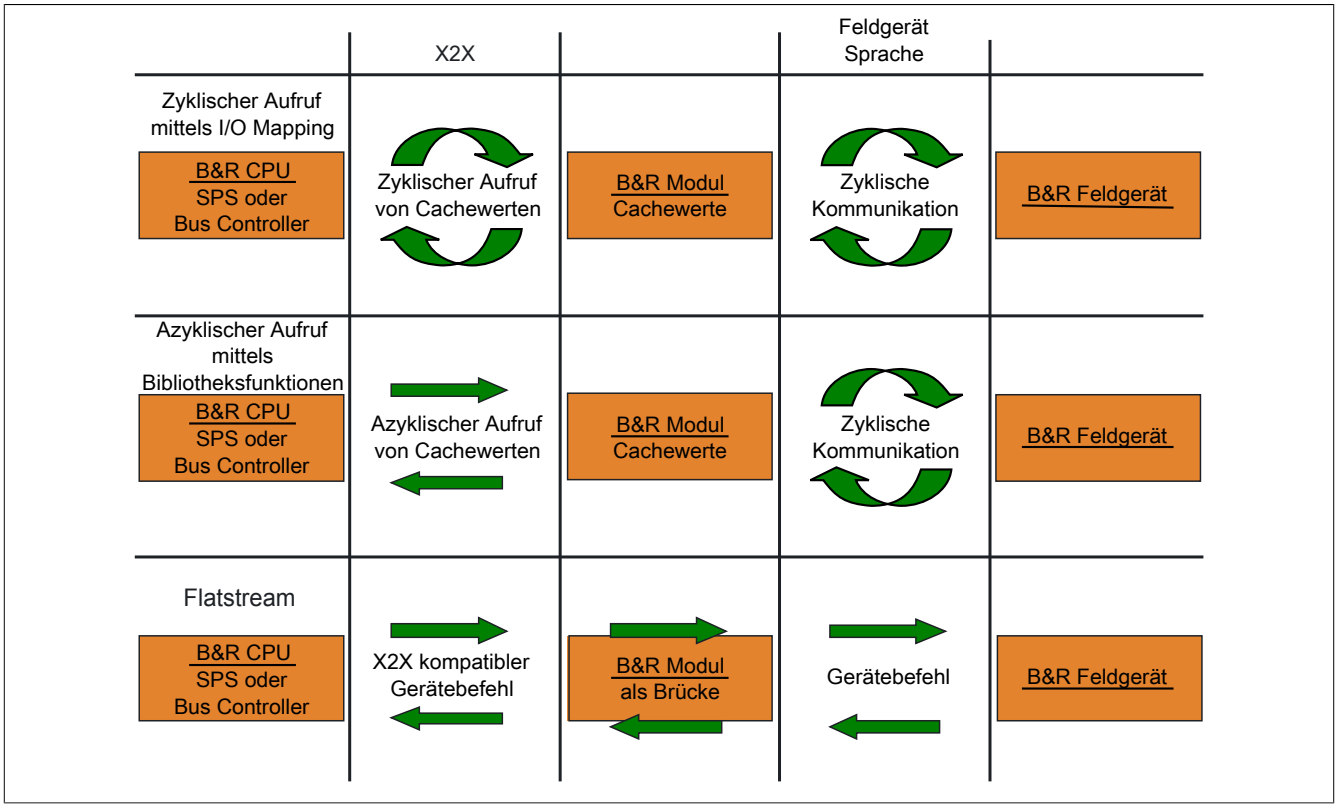


Abbildung 1: 3 Arten der Kommunikation

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.

11.9.2 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.

11.9.3 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.

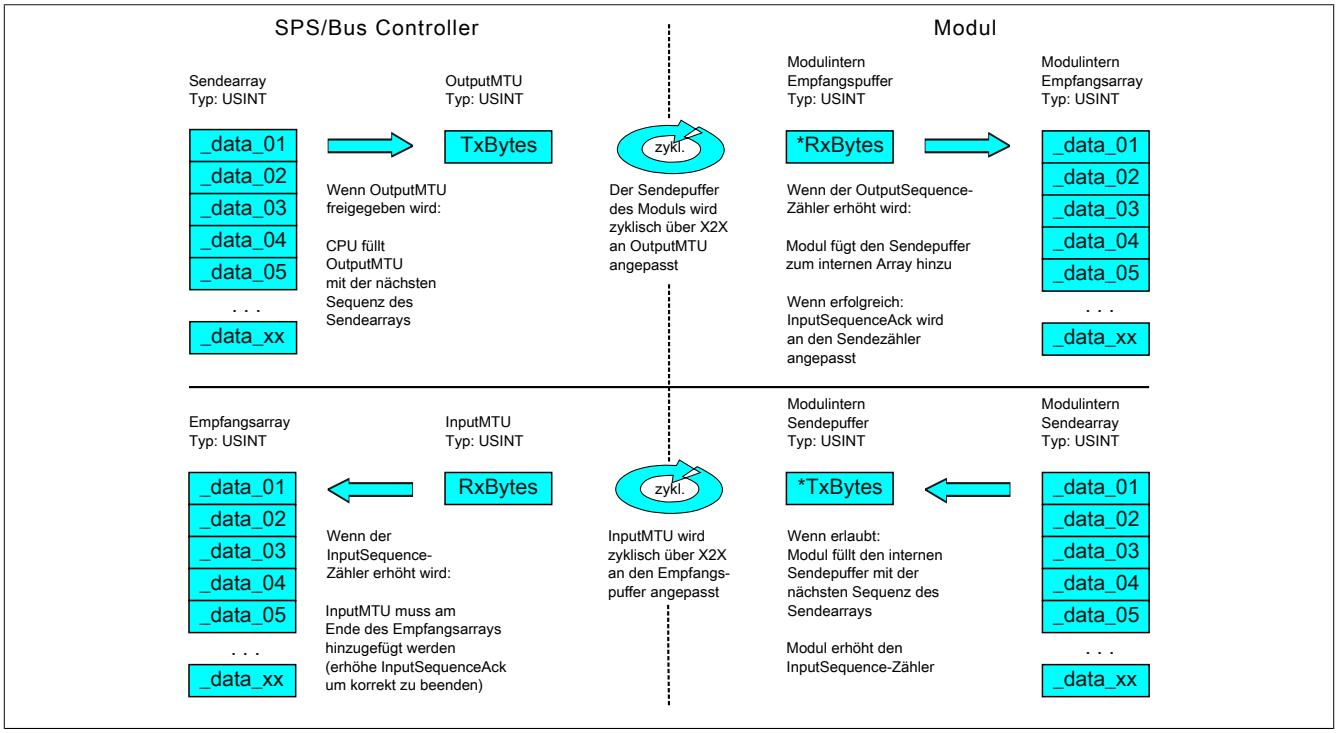


Abbildung 2: Kommunikation per Flatstream

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.

11.9.4 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.

11.9.4.1 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.

11.9.4.1.1 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)

11.9.4.2 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.

11.9.4.2.1 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

11.9.4.2.2 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

11.9.4.2.3 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

11.9.4.2.4 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.

11.9.4.2.5 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.

11.9.4.2.6 Beziehung zwischen Output- und InputSequence

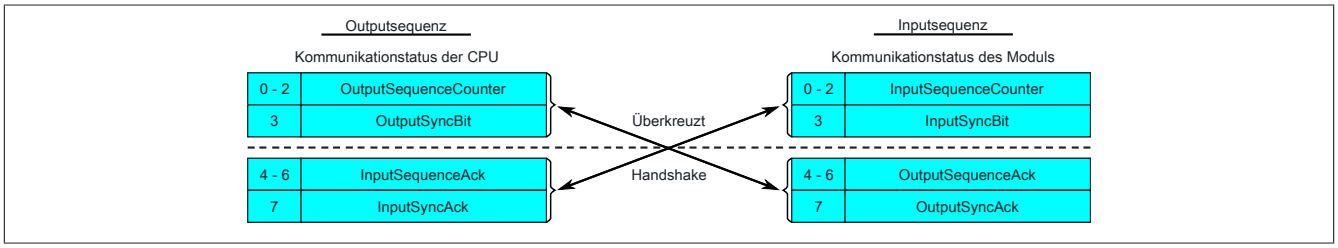


Abbildung 3: Zusammenhang zwischen Output- und InputSequence

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.

11.9.4.3 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.

11.9.4.4 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.

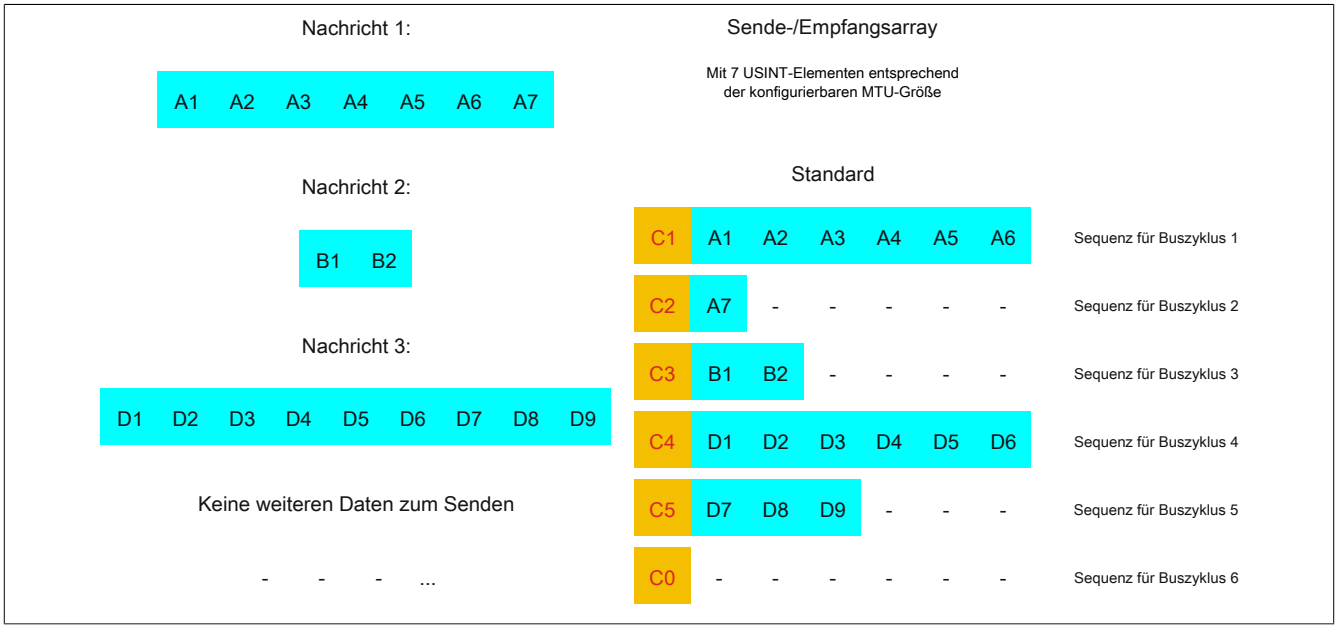


Abbildung 4: Sende-/Empfangsarray (Standard)

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)

11.9.4.5 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.

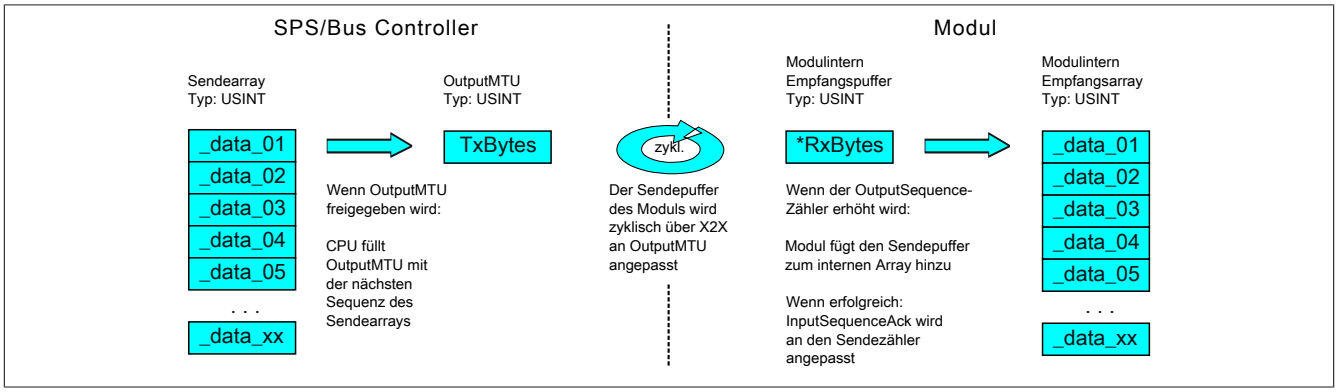


Abbildung 5: Kommunikation per Flatstream (Output)

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

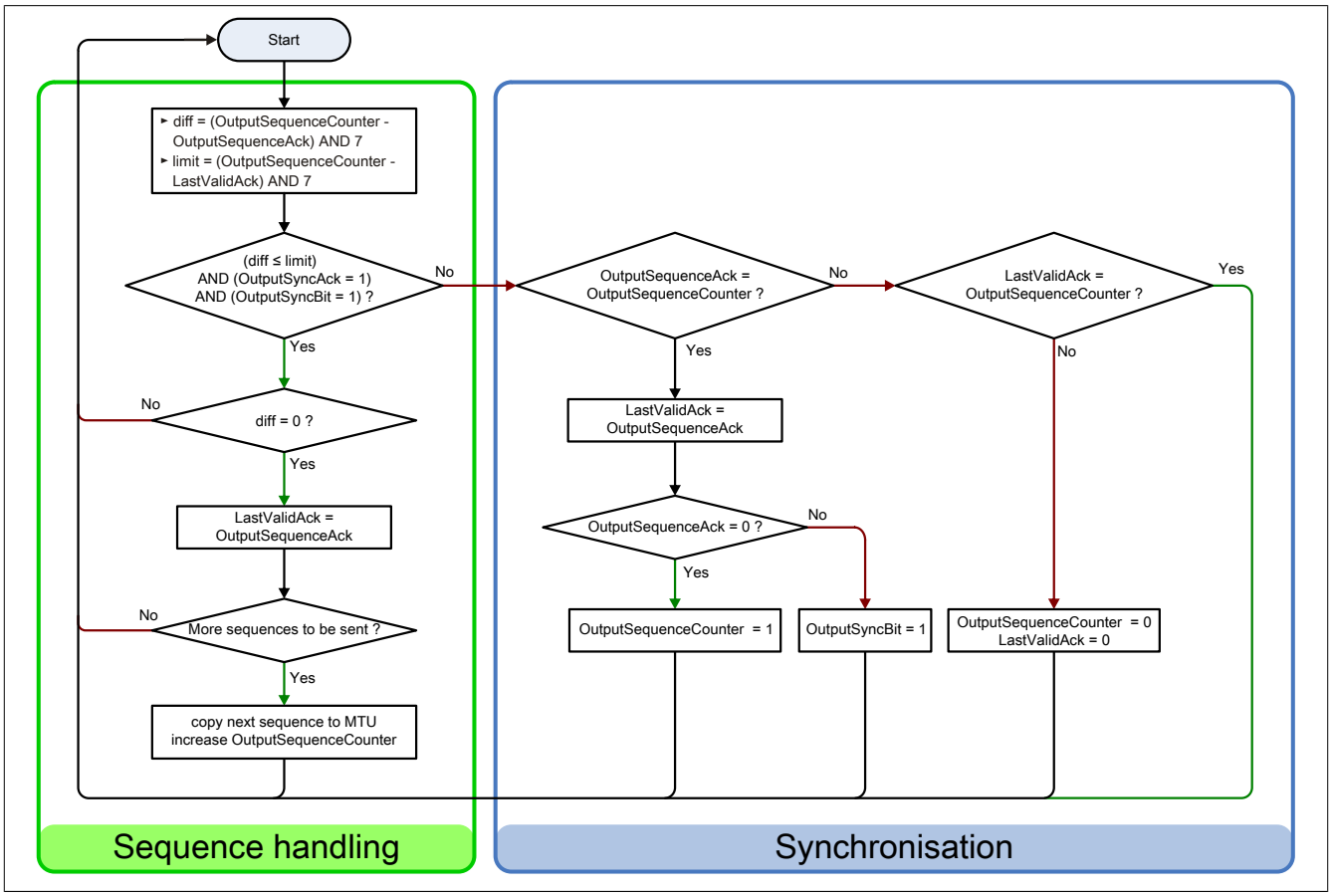


Abbildung 6: Ablaufdiagramm für Output-Richtung

11.9.4.6 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.

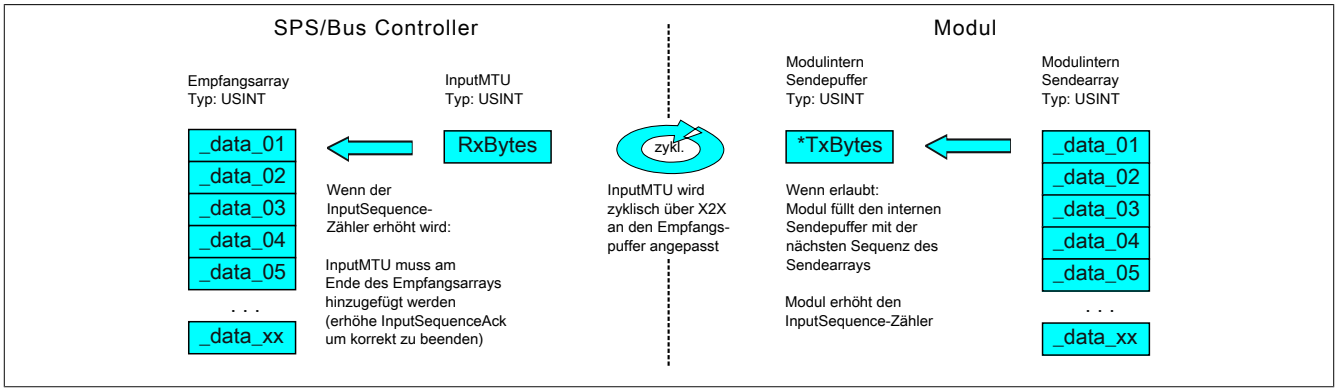


Abbildung 7: Kommunikation per Flatstream (Input)

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

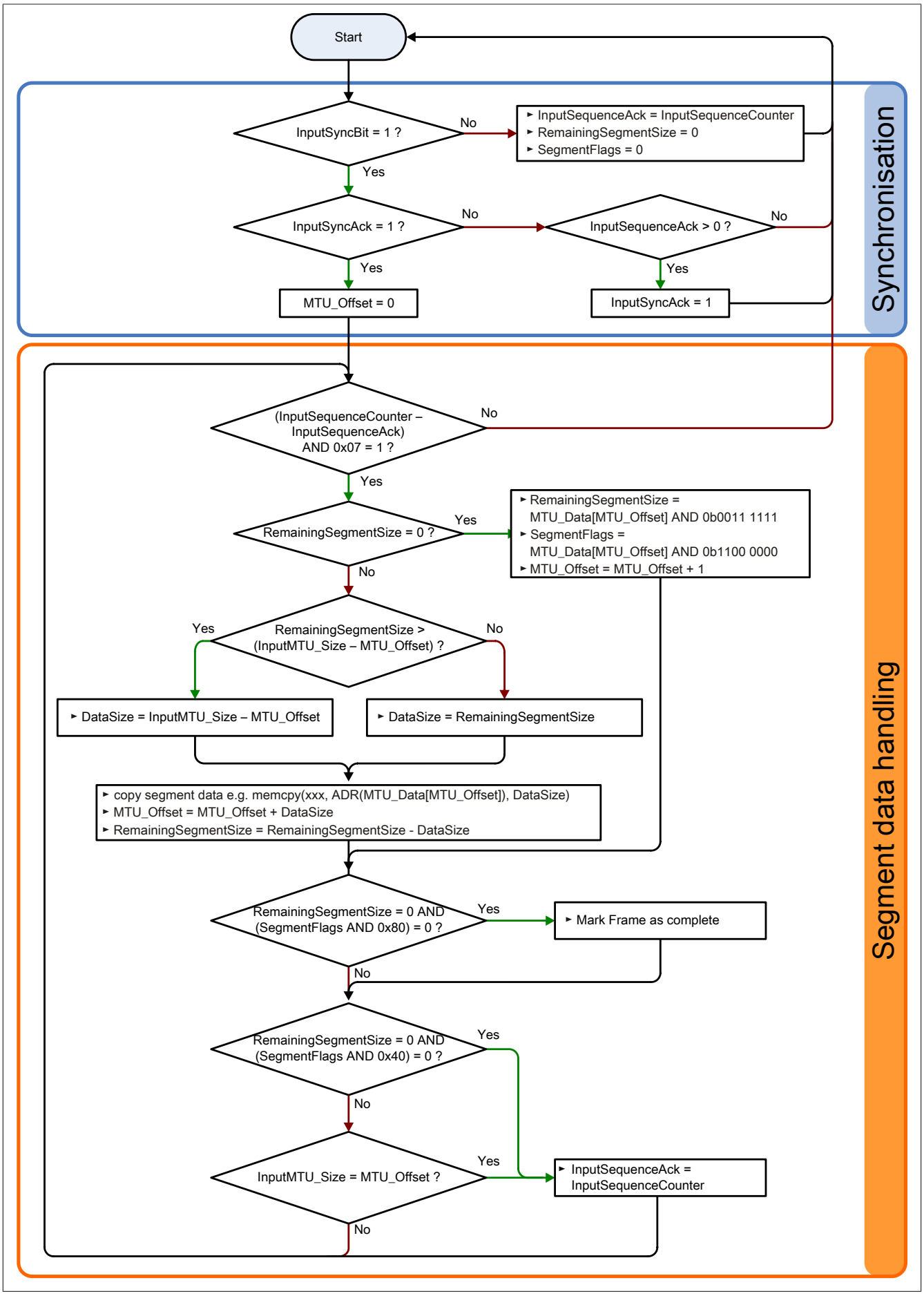


Abbildung 8: Ablaufdiagramm für Input-Richtung

11.9.4.7 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.

11.9.4.8 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.

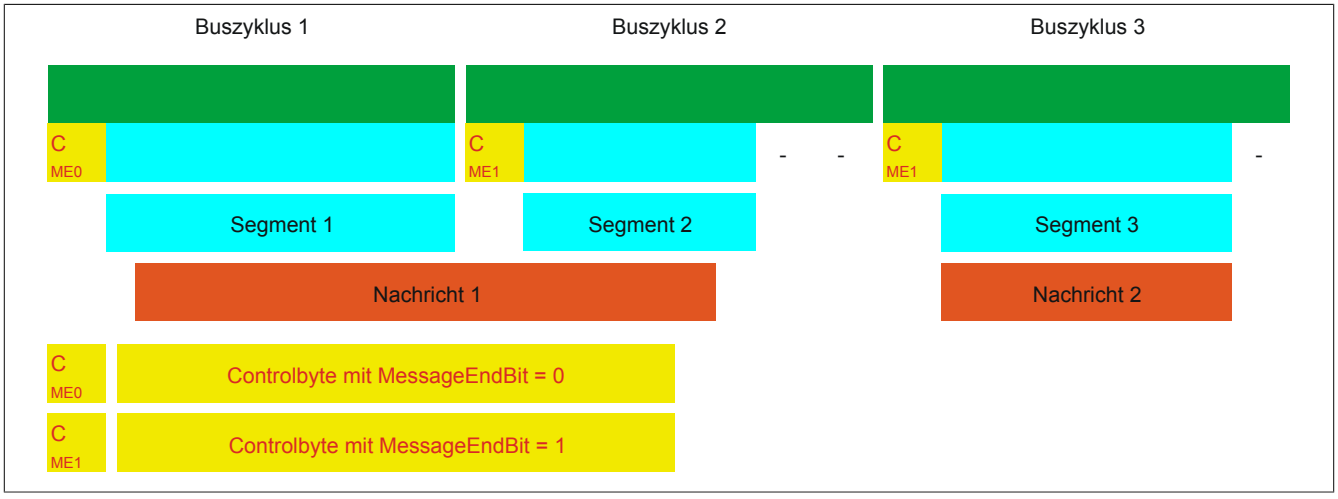


Abbildung 9: Anordnung von Nachrichten in der MTU (Standard)

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.

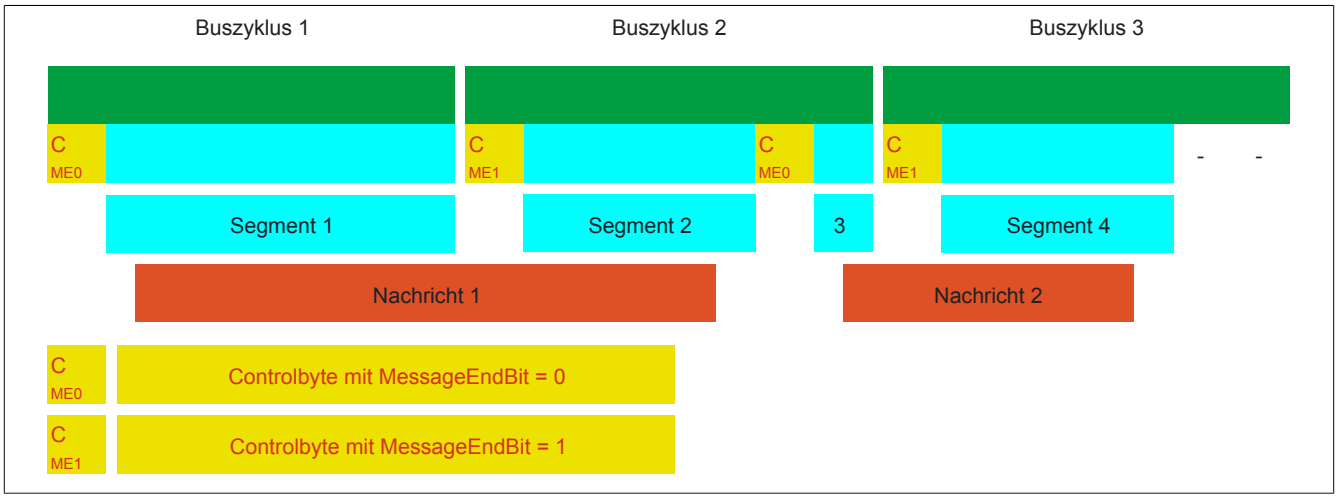


Abbildung 10: Anordnung von Nachrichten in der MTU (MultiSegmentMTU)

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.

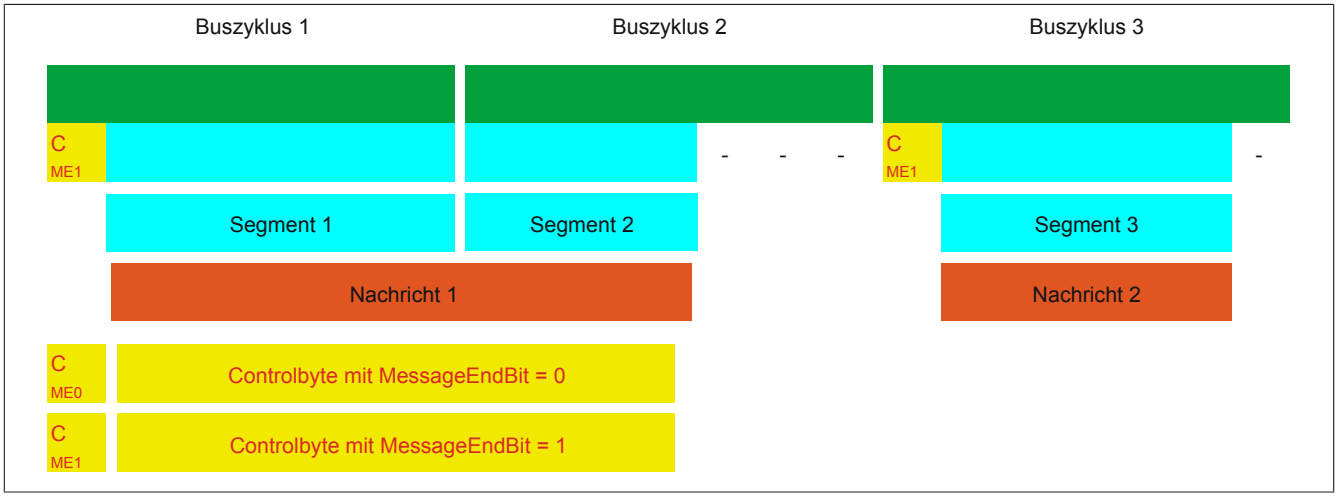


Abbildung 11: Anordnung von Nachrichten in der MTU (große Segmente)

Anwendung beider Optionen

Die beiden Optionen dürfen auch gleichzeitig angewendet werden.

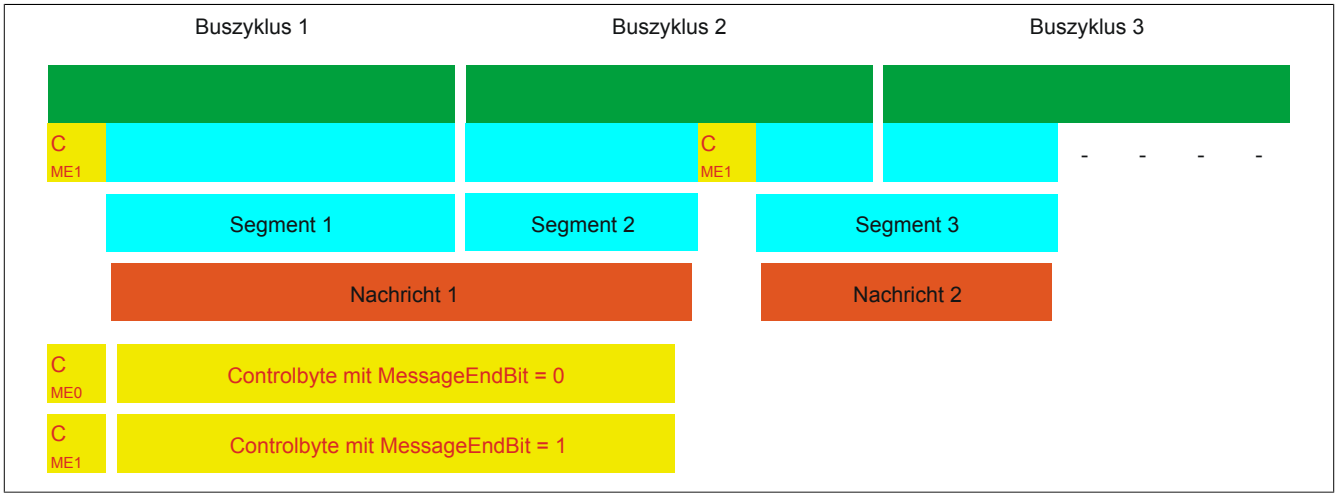


Abbildung 12: Anordnung von Nachrichten in der MTU (große Segmente und MultiSegmentMTU)

11.9.4.9 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.

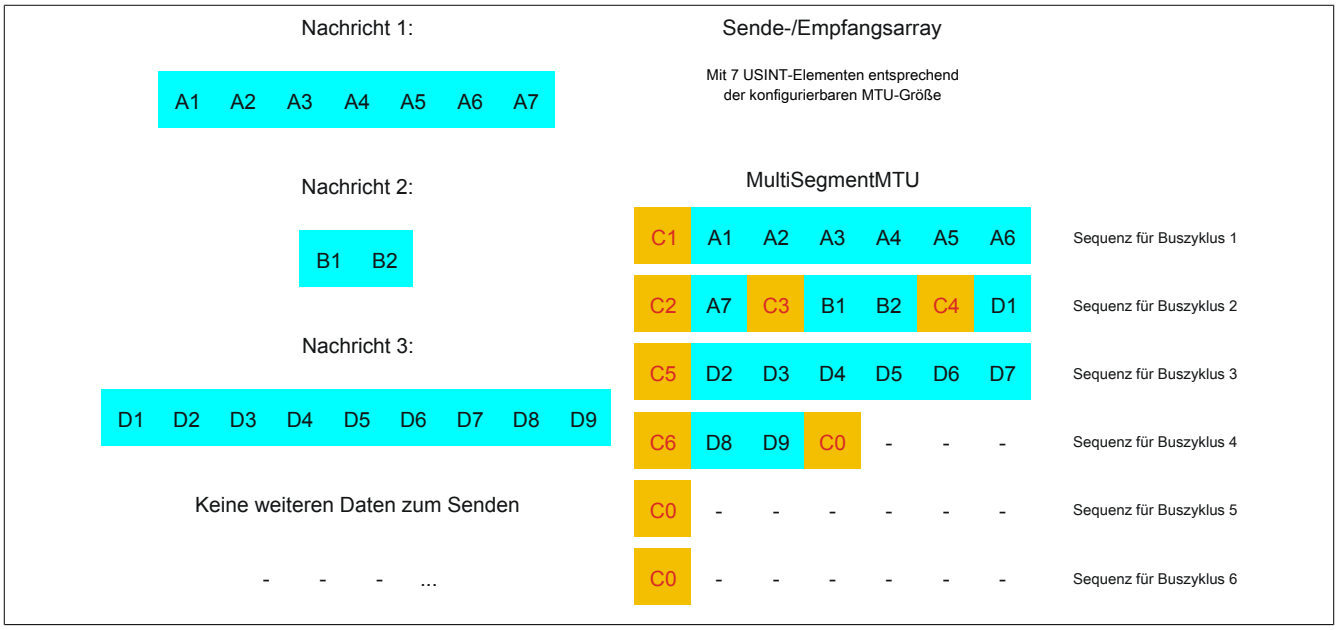


Abbildung 13: Sende-/Empfangsarray (MultiSegmentMTU)

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.

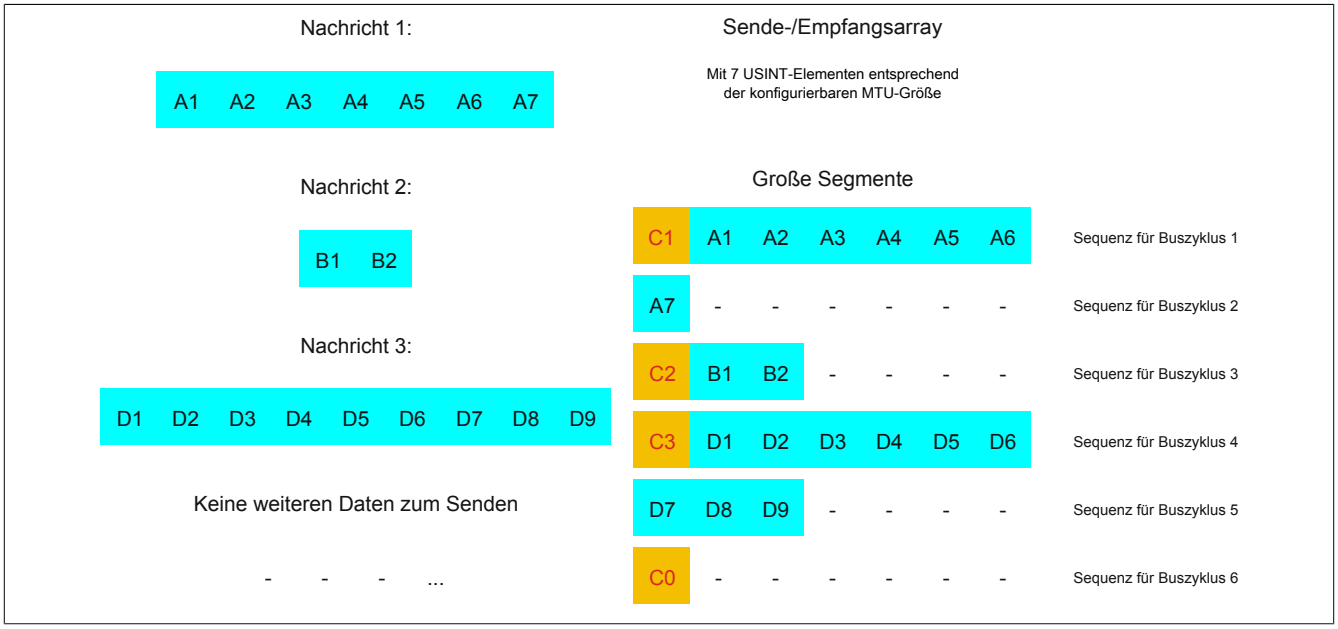


Abbildung 14: Sende-/Empfangsarray (große Segmente)

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.

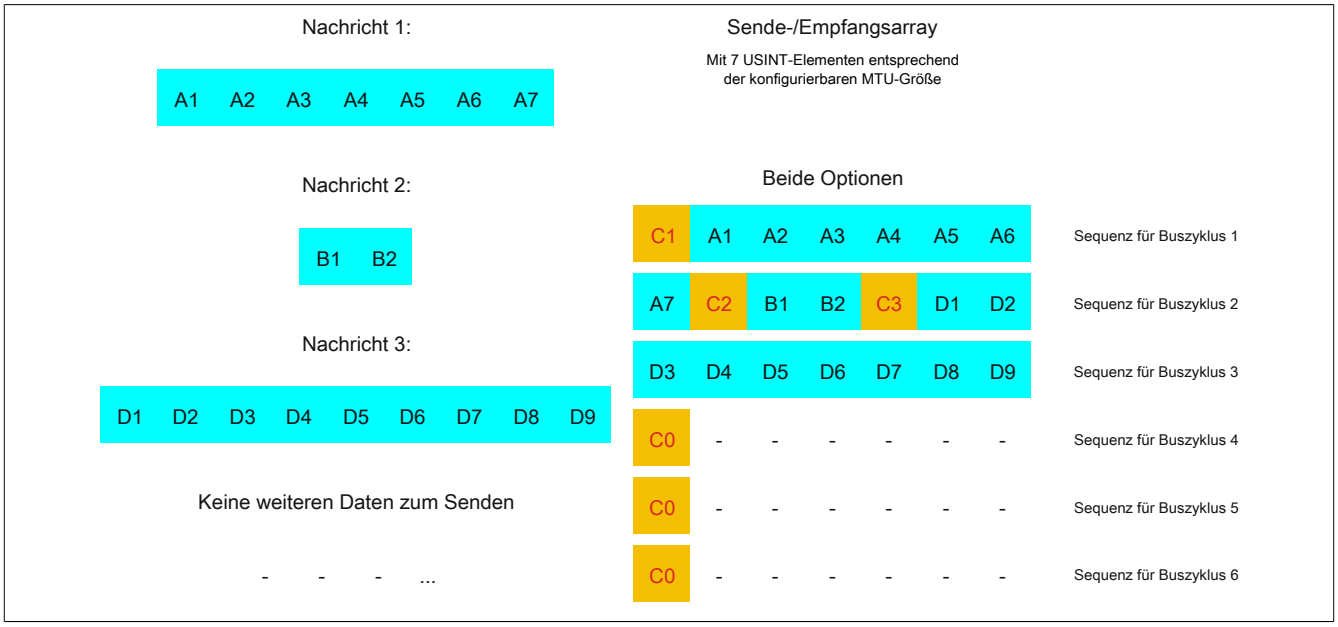


Abbildung 15: Sende-/Empfangsarray (große Segmente und MultiSegmentMTU)

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

11.9.5 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.

11.9.5.1 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)

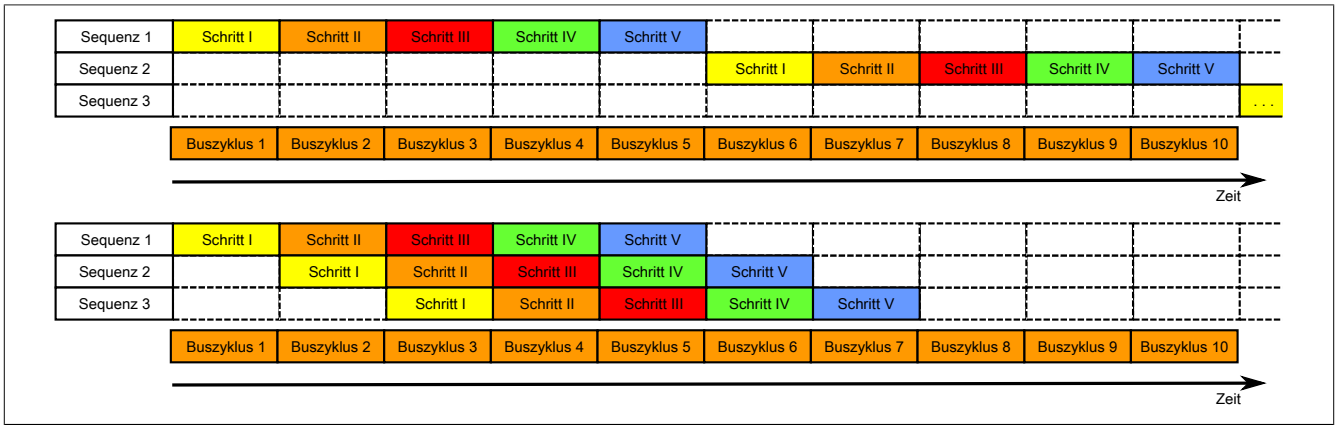


Abbildung 16: Vergleich Übertragung ohne bzw. mit Forward

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.

11.9.5.2 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.

11.9.5.2.1 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

11.9.5.2.2 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

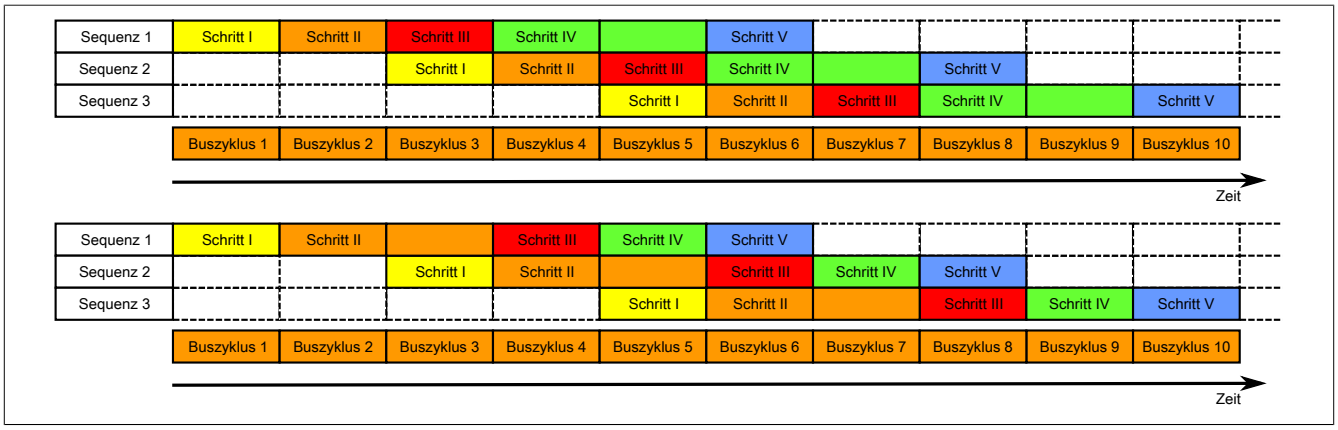


Abbildung 17: Auswirkung des ForwardDelay bei der Flatstream-Kommunikation mit Forward

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.

11.9.5.3 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.

11.9.5.4 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).

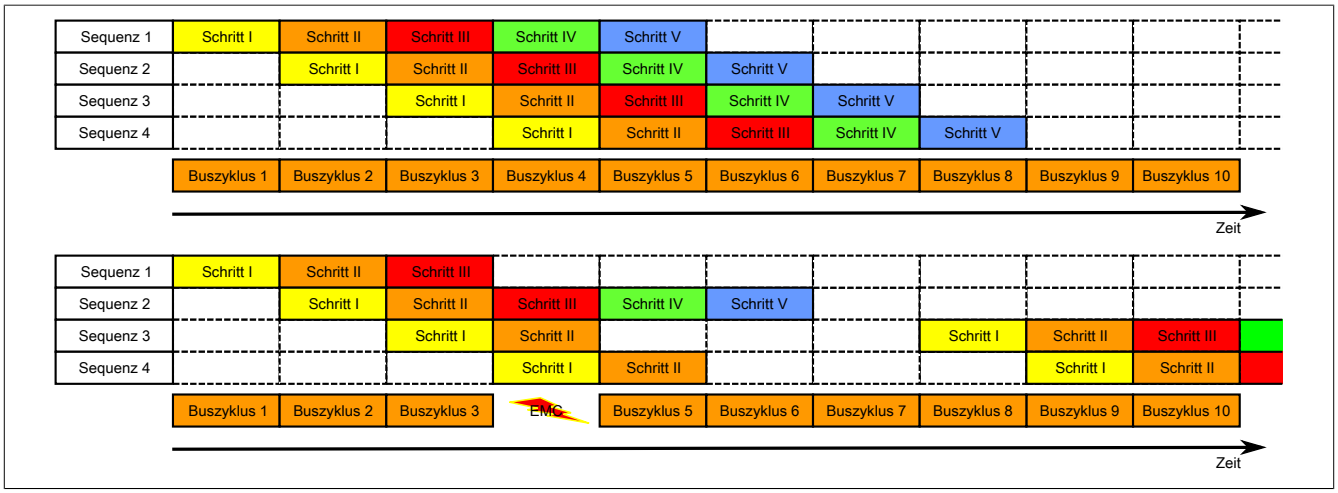


Abbildung 18: Auswirkung eines ausgefallenen Buszyklus

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.

11.10 EnDat mit Flatstream

EnDat ist eine synchrone Schnittstelle, die halbduplex kommunizieren kann. Um eine fehlerfreie Signalübertragung zu gewährleisten, werden verschiedene Sicherheitsvorkehrungen getroffen.

- Im Anschluss an ein Nutzsignal wird eine automatisch erzeugte Prüfsumme übermittelt, die der Empfänger auswertet.
- Zu Beginn einer Antwort wird der Befehl wiederholt, auf den der Geber reagiert.

Im Flatstream-Modus arbeitet das Modul als Bridge zwischen der CPU und dem EnDat-Slave. Es wurden EnDat-spezifische Algorithmen implementiert, um Zeitüberschreitungen zu überwachen und Prüfsummen zu handhaben. Dem Anwender sind diese Details im Normalbetrieb nicht zugänglich.

Weiterführende Informationen sind der Dokumentation "Technische Information - EnDat 2.2" bzw. den Herstellerdaten des Gebers zu entnehmen.

11.10.1 Übersicht der herkömmlichen EnDat-Kommandos für den Flatstream-Modus

Kommando-byte [hex]	Kommando	Nur EnDat 2.2
0x00	Zurücksetzen	
0x01	Fehler quittieren	
0x04	Parameter lesen	
0x05	Parameter schreiben	
0x06	Parameter aus Speicherblock lesen	•
0x07	Parameter in Speicherblock schreiben	•
0x08	Wort 1 der Zusatzinformation lesen	•
0x09	Wort 2 der Zusatzinformation lesen	•
0x0A	Wort 3 der Zusatzinformation lesen	•
0x0B	Wort 4 der Zusatzinformation lesen	•

11.10.2 Zurücksetzen (0x00)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x00	Kommando (Zurücksetzen)
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x00	Wiederholung (Sicherheit)
Master		

11.10.3 Fehler quittieren (0x01)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x01	Kommando (Fehler quittieren)
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x01	Wiederholung (Sicherheit)
Master		

11.10.4 Parameter lesen (0x04)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x04	Kommando (Parameter lesen)
2	MRS-Code	zu lesender Speicherbereich
3	Parameternr.	
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x04	Wiederholung (Sicherheit)
2	MRS-Code	
3	Parameternr.	
4	Value_L	ausgelesener Wert
5	Value_H	
Master		

11.10.5 Parameter schreiben (0x05)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x05	Kommando (Parameter schreiben)
2	MRS-Code	zu schreibender Speicherbereich
3	Parameternr.	
4	Value_L	zu schreibender Wert
5	Value_H	
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x05	Wiederholung (Sicherheit)
2	MRS-Code	
3	Parameternr.	
Master		

11.10.6 Parameter aus Speicherblock lesen (0x06)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x06	Kommando (Parameter aus Speicherblock lesen)
2	MRS-Code	zu lesender Speicherbereich
3	Blocknr.	
4	Parameternr.	
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x06	Wiederholung (Sicherheit)
2	MRS-Code	
3	Blocknr.	
4	Parameternr.	
5	Value_L	ausgelesener Wert
6	Value_H	
Master		

11.10.7 Parameter in Speicherblock schreiben (0x07)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x07	Kommando (Parameter in Speicherblock schreiben)
2	MRS-Code	zu schreibender Speicherbereich
3	Blocknr.	
4	Parameternr.	
5	Value_L	zu schreibender Wert
6	Value_H	
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x07	Wiederholung (Sicherheit)
2	MRS-Code	
3	Blocknr.	
4	Parameternr.	
Master		

11.10.8 Wort 1 der Zusatzinformation lesen (0x08)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x08	Kommando (Wort 1 der Zusatzinformation lesen)
2	MRS-Code	zu lesender Speicherbereich
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x08	Wiederholung (Sicherheit)
2	MRS-Code	
3	Value_L	Wort 1 der Zusatzinformation
4	Value_H	
Master		

11.10.9 Wort 2 der Zusatzinformation lesen (0x09)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x09	Kommando (Wort 2 der Zusatzinformation lesen)
2	MRS-Code	zu lesender Speicherbereich
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x09	Wiederholung (Sicherheit)
2	MRS-Code	
3	Value_L	Wort 1 der Zusatzinformation (Overhead)
4	Value_H	
5	Value_L	Wort 2 der Zusatzinformation
6	Value_H	
Master		

11.10.10 Wort 3 der Zusatzinformation lesen (0x0A)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x0A	Kommando (Wort 3 der Zusatzinformation lesen)
2	MRS-Code	zu lesender Speicherbereich
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x0A	Wiederholung (Sicherheit)
2	MRS-Code	
3	Value_L	Wort 1 der Zusatzinformation (Overhead)
4	Value_H	
5	Value_L	Wort 2 der Zusatzinformation (Overhead)
6	Value_H	
7	Value_L	Wort 3 der Zusatzinformation
8	Value_H	
Master		

11.10.11 Wort 4 der Zusatzinformation lesen (0x0B)

Master-Kommando

Protokollbytes		Information
Nr.	Name	
Master		
1	0x0B	Kommando (Wort 4 der Zusatzinformation lesen)
2	MRS-Code	zu lesender Speicherbereich
Slave		

Slave-Antwort

Protokollbytes		Information
Nr.	Name	
Slave		
1	0x0B	Wiederholung (Sicherheit)
2	MRS-Code	
3	Value_L	Wort 1 der Zusatzinformation (Overhead)
4	Value_H	
5	Value_L	Wort 2 der Zusatzinformation (Overhead)
6	Value_H	
7	Value_L	Wort 3 der Zusatzinformation (Overhead)
8	Value_H	
9	Value_L	Wort 4 der Zusatzinformation
10	Value_H	
Master		

11.11 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.

11.11.1 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.

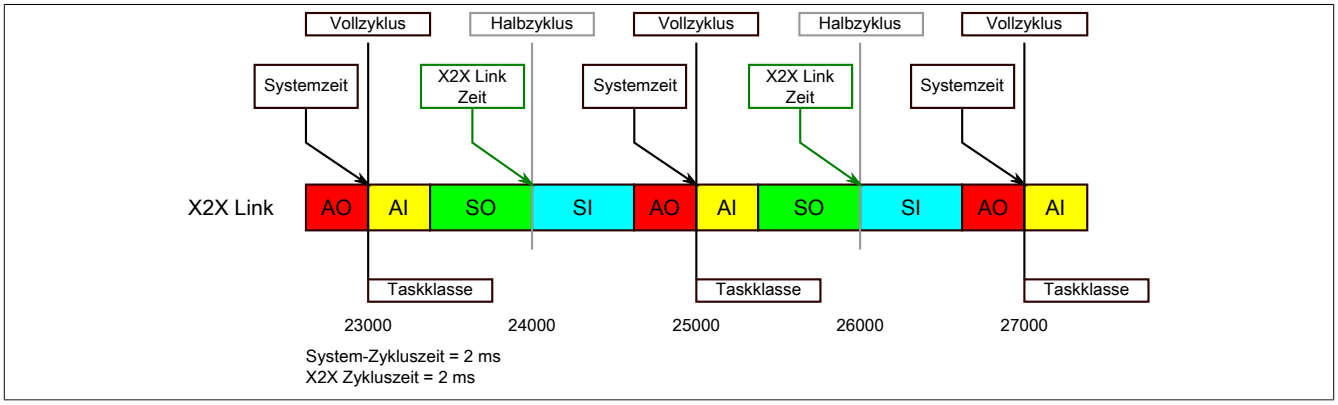
Die Initialisierung der Zeiten erfolgt auf Basis der Systemzeit während des Hochlaufs des X2X Links, der I/O-Module bzw. der POWERLINK-Schnittstelle.

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

11.11.1.1 SPS/Controller-Datenpunkte

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

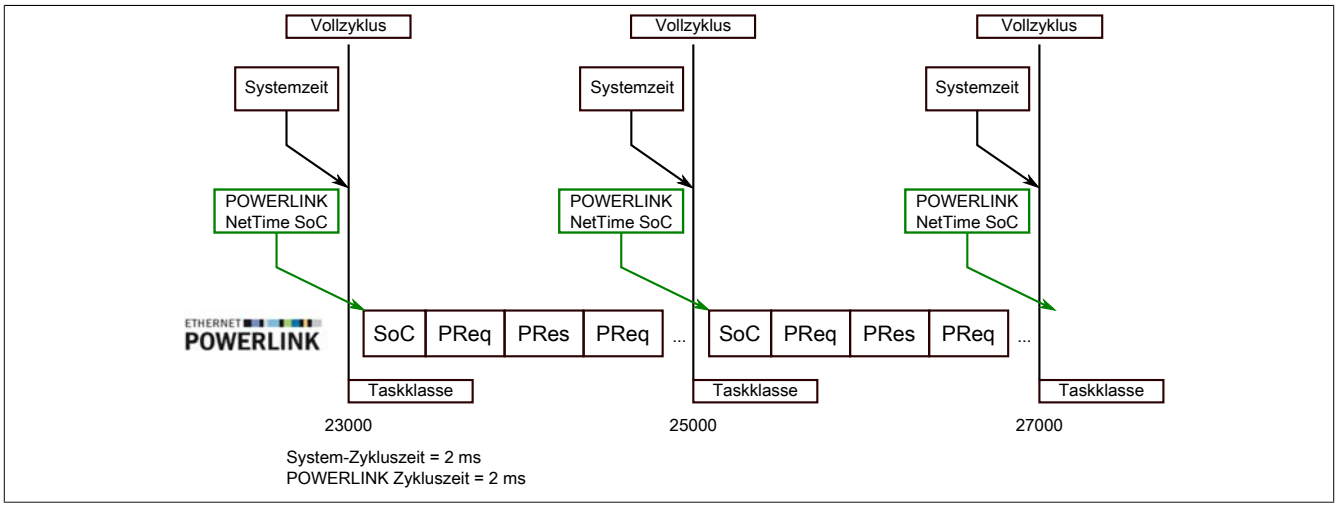
11.11.1.2 Refenzzeitpunkt X2X Link



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.

11.11.1.3 Referenzzeitpunkt POWERLINK

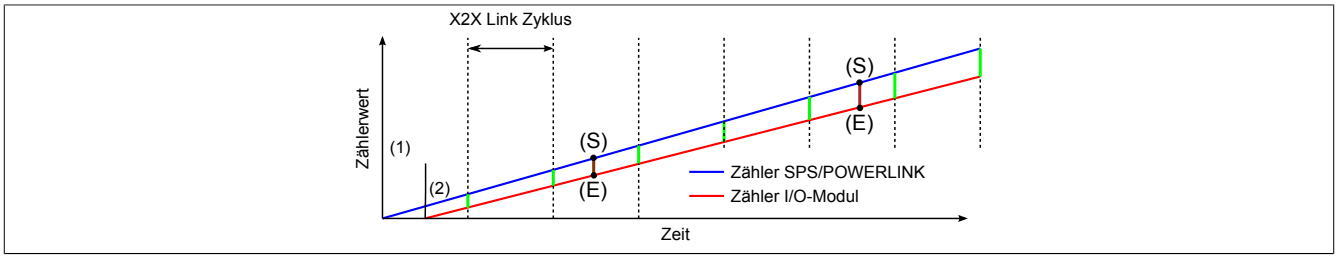


Der Referenzzeitpunkt am POWERLINK wird grundsätzlich beim SoC (Start of Cycle) des POWERLINK-Netzwerks gebildet. Der SoC startet systembedingt 20 µs nach dem Systemtakt. Dadurch ergibt sich folgende Differenz zwischen Systemzeit und POWERLINK-Referenzzeit:

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.

11.11.1.4 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.

11.11.2 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.

11.11.2.1 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.**

11.11.2.2 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.**

11.11.2.3 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.

11.12 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
100 µs

11.13 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
100 µs