X20CS1070

1 Allgemeines

1.1 Mitgeltende Dokumente

Weiterführende und ergänzende Informationen sind den folgenden gelisteten Dokumenten zu entnehmen.

Mitgeltende Dokumente

Dokumentname	Titel
MAX20	X20 System Anwenderhandbuch
MAEMV	Installations- / EMV-Guide

1.2 Bestelldaten

Bestellnummer	Kurzbeschreibung	Abbildung
	Kommunikation im X20 Elektronikmodul	
X20CS1070	X20 Schnittstellenmodul, 1 CAN-Bus-Schnittstelle, max. 1 MBit/s, Objektpuffer in Sende- und Empfangsrichtung	
	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	
X20TB06	X20 Feldklemme, 6-polig, 24 VDC codiert	
X20TB12	X20 Feldklemme, 12-polig, 24 VDC codiert	

Tabelle 1: X20CS1070 - Bestelldaten

1.3 Modulbeschreibung

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

- CAN-Bus Schnittstelle für das serielle dezentrale Anbinden komplexer Geräte an das X20 System
- Integrierter Abschlusswiderstand

2 Technische Beschreibung

2.1 Technische Daten

Bestellnummer	X20CS1070
Kurzbeschreibung	
Kommunikationsmodul	1x CAN-Bus
Allgemeines	
B&R ID-Code	0x1FD1
Statusanzeigen	Datenübertragung, Abschlusswiderstand, Betriebszustand, Modulstatus
Diagnose	
Modul Run/Error	Ja, per Status-LED und SW-Status
Datenübertragung	Ja, per Status-LED
Abschlusswiderstand	Ja, per Status-LED
Leistungsaufnahme	
Bus	0,01 W
I/O-intern	1,44 W
Zusätzliche Verlustleistung durch Aktoren (ohmsch) [W]	-
Zulassungen	
CE	Ja
UKCA	Ja
ATEX	Zone 2, II 3G Ex nA nC IIA T5 Gc IP20, Ta (siehe X20 Anwenderhandbuch) FTZÚ 09 ATEX 0083X
UL	cULus E115267 Industrial Control Equipment
HazLoc	cCSAus 244665 Process Control Equipment for Hazardous Locations Class I, Division 2, Groups ABCD, T5
DNV	Temperature: **B** (0 to 55 °C) Humidity: **B** (up to 100%) Vibration: **B** (4 g) EMC: **B** (bridge and open deck)
LR	ENV1
KR	Ja
ABS	Ja
BV	**EC33B** Temperature: 5 - 55 °C Vibration: 4 g EMC: Bridge and open deck
EAC	Ja
KC	Ja
Schnittstellen	
Schnittstelle IF1	
Signal	CAN-Bus
Ausführung	Kontaktierung über 12-polige Feldklemme X20TB12
max. Reichweite	1000 m
Übertragungsrate	max. 1 MBit/s
Abschlusswiderstand	Im Modul integriert
Controller	SJA 1000
Elektrische Eigenschaften	
Potenzialtrennung	CAN (IF1) zu Bus und I/O-Versorgung 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: X20CS1070 - Technische Daten

Bestellnummer	X20CS1070
Luftfeuchtigkeit	
Betrieb	5 bis 95%, nicht kondensierend
Lagerung	5 bis 95%, nicht kondensierend
Transport	5 bis 95%, nicht kondensierend
Mechanische Eigenschaften	
Anmerkung	Feldklemme 1x X20TB06 oder X20TB12 gesondert bestellen Busmodul 1x X20BM11 gesondert bestellen
Rastermaß	12,5 $^{+0,2}$ mm

Tabelle 2: X20CS1070 - Technische Daten

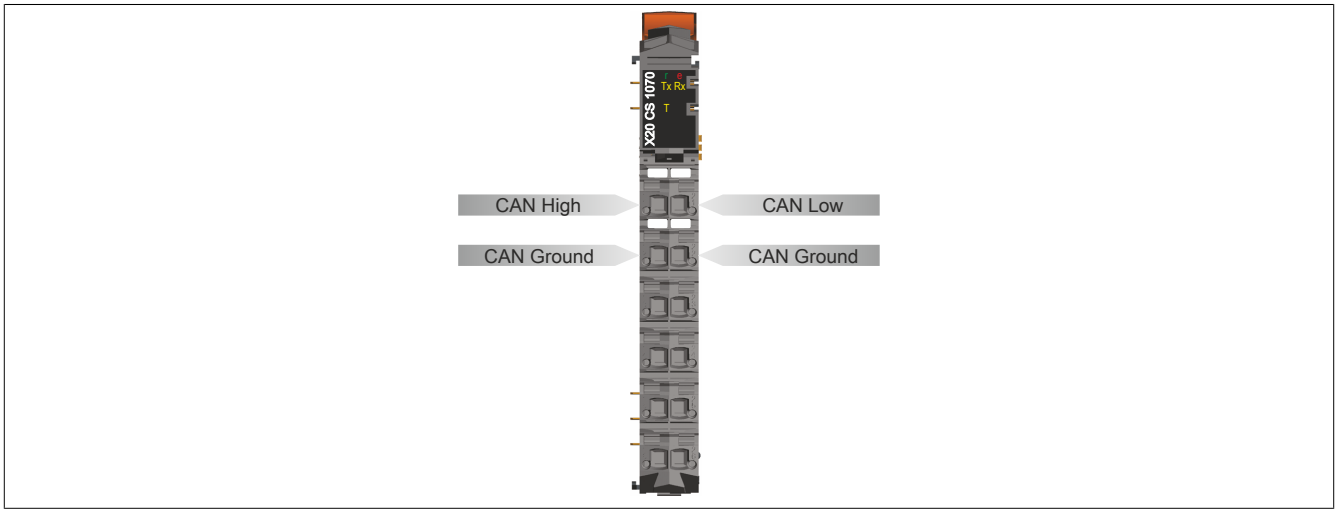
2.2 Status-LEDs

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

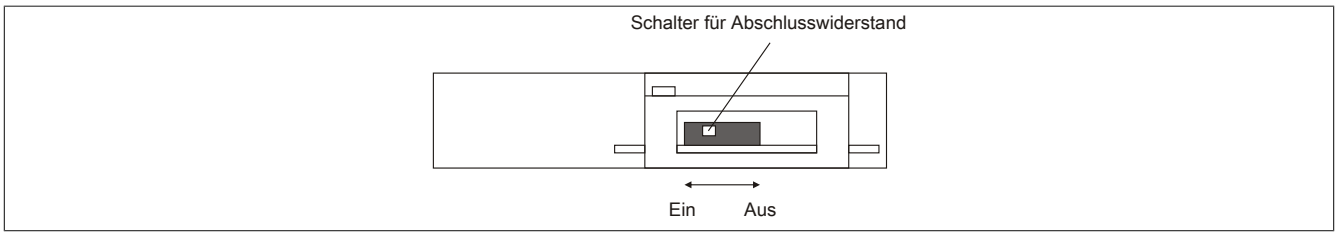
Abbildung	LED	Farbe	Status	Beschreibung
	r	Grün	Aus	Modul nicht versorgt
			Single Flash	Modus RESET
			Double Flash	Modus BOOT(während Firmware-Update)[1]
			Blinkend	Modus PREOPERATIONAL
			Ein	Modus RUN
	e	Rot	Aus	Modul nicht versorgt oder alles in Ordnung
			Single Flash	I/O Fehler ist aufgetreten • CAN-Bus: Warnung, Passiv oder Aus • Pufferüberlauf
			Ein	Fehler- oder Resetzustand
	e + r	Rot ein / grüner Single Flash		Firmware ist ungültig
	Tx	Gelb	Ein	Das Modul sendet Daten über die CAN-Bus Schnittstelle
	Rx	Gelb	Ein	Das Modul empfängt Daten über die CAN-Bus Schnittstelle
	T	Gelb	Ein	Der im Modul integrierte Abschlusswiderstand ist zugeschaltet

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

2.3 Anschlussbelegung



2.4 Abschlusswiderstand



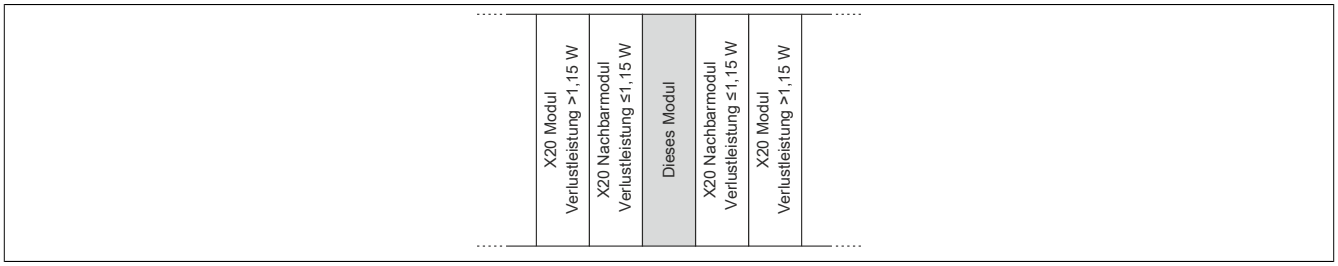
Am Kommunikationsmodul ist bereits ein Abschlusswiderstand integriert. Mit einem Schalter an der Gehäuseunterseite wird der Abschlusswiderstand zu- oder abgeschaltet. Ein aktivierter Abschlusswiderstand wird durch die LED "T" angezeigt.

2.5 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.



3 Funktionsbeschreibung

3.1 Das CAN-Objekt

Ein CAN-Objekt besteht immer aus 4 Bytes Identifier sowie maximal 8 folgenden Datenbytes. Dadurch ergibt sich auch der Zusammenhang zwischen CAN-Objektlänge und der Anzahl der CAN-Nutzdaten. Dies ist wichtig, da die Anzahl der CAN-Nutzdatenbytes bei der Kommunikation mittels "Flatstream" immer über die Framelänge bestimmt werden muss.

Zusammensetzung eines CAN-Objekts bzw. CAN-Frames

Byte	Bedeutung	Information
1	Identifier	ID Bit 0 bis 7
2		ID Bit 8 bis 15
3		ID Bit 16 bis 23
4		ID Bit 24 bis 31
5 - 12	CAN-Nutzdaten	0 bis 8 CAN-Nutzdatenbytes

Identifier

Die 32 Bits (4 Bytes) des CAN-Identifiers werden wie folgt verwendet:

Bit	Beschreibung	Wert	Information
0	Frameformat	0	Standard-Frameformat (SFF) mit 11 Bit Identifier
		1	Extended-Frameformat (EFF) mit 29 Bit Identifier
1	Frametyp	0	Datenframe
		1	Remoteframe (RTR)
2	Reserviert	-	
3 - 31	CAN-Identifier des zu sendenden Telegramms	x	Extended-Frameformat (EFF) mit 29 Bits Standard-Frameformat (SFF) mit 11 Bits[1)]

1) Nur Bits 21 bis 31 werden verwendet; Bits 3 bis 20 = 0

3.1.1 Datenstrom des CAN-Moduls

Im Funktionsmodell 254 werden die zu übertragenden Datenpakete eines Datenstroms als Frames bezeichnet.

Information:

Für das CAN-Modul gilt:

- **Ein Frame enthält immer ein CAN-Objekt und kann damit nicht länger als 12 Bytes sein.**
- **Das CAN-Objekt wird erst in den Sendepuffer übernommen, nachdem der Frame abgeschlossen wurde.**
- **Die CAN-Nutzdatenlänge steht in festem Zusammenhang mit der Framelänge bzw. der tatsächlichen Größe des CAN-Objekts. Dabei gilt:**
  - **CAN-Nutzdatenlänge = Framelänge - 4**
  - **Framelänge = CAN-Nutzdatenlänge + 4**

3.2 Die Flatstream-Kommunikation

3.2.1 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.

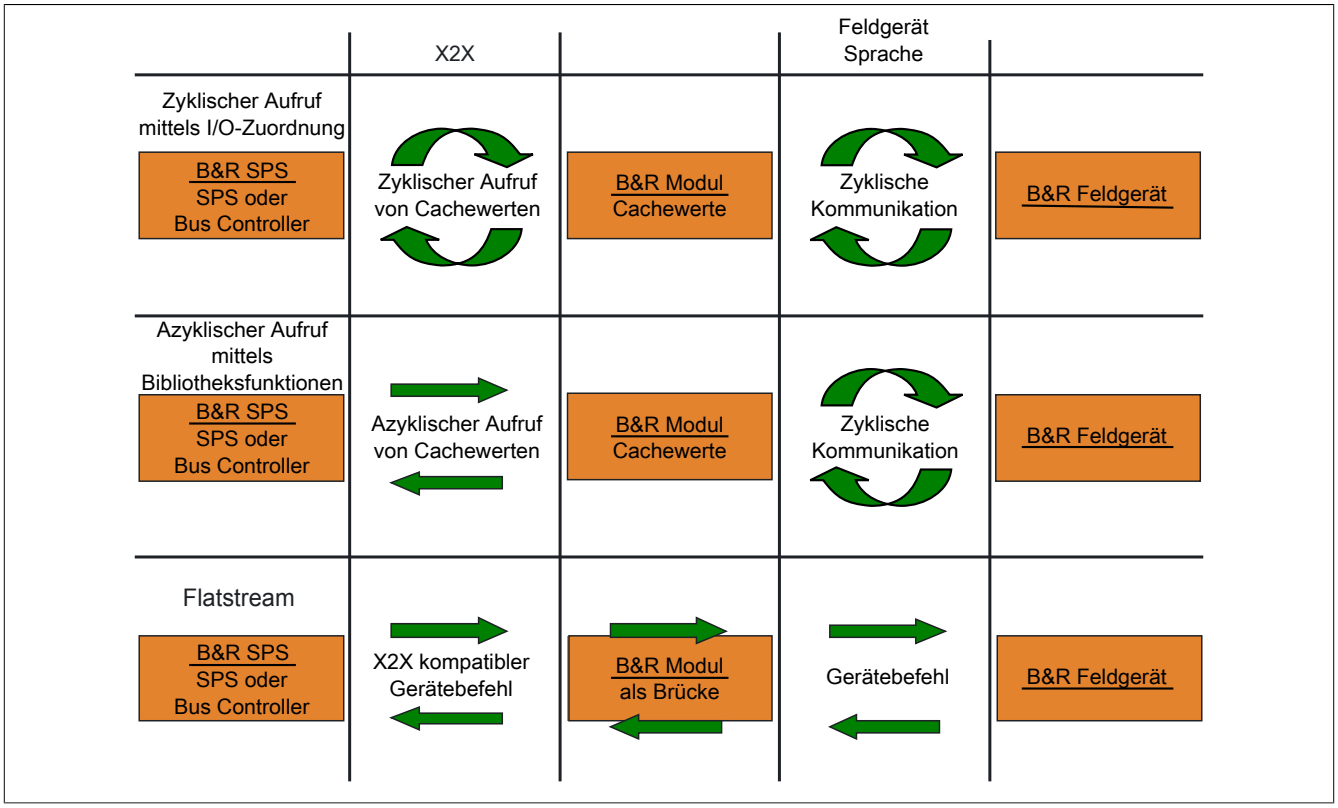


Abbildung 1: 3 Arten der Kommunikation

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

3.2.2 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 Forwards 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.

3.2.3 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.

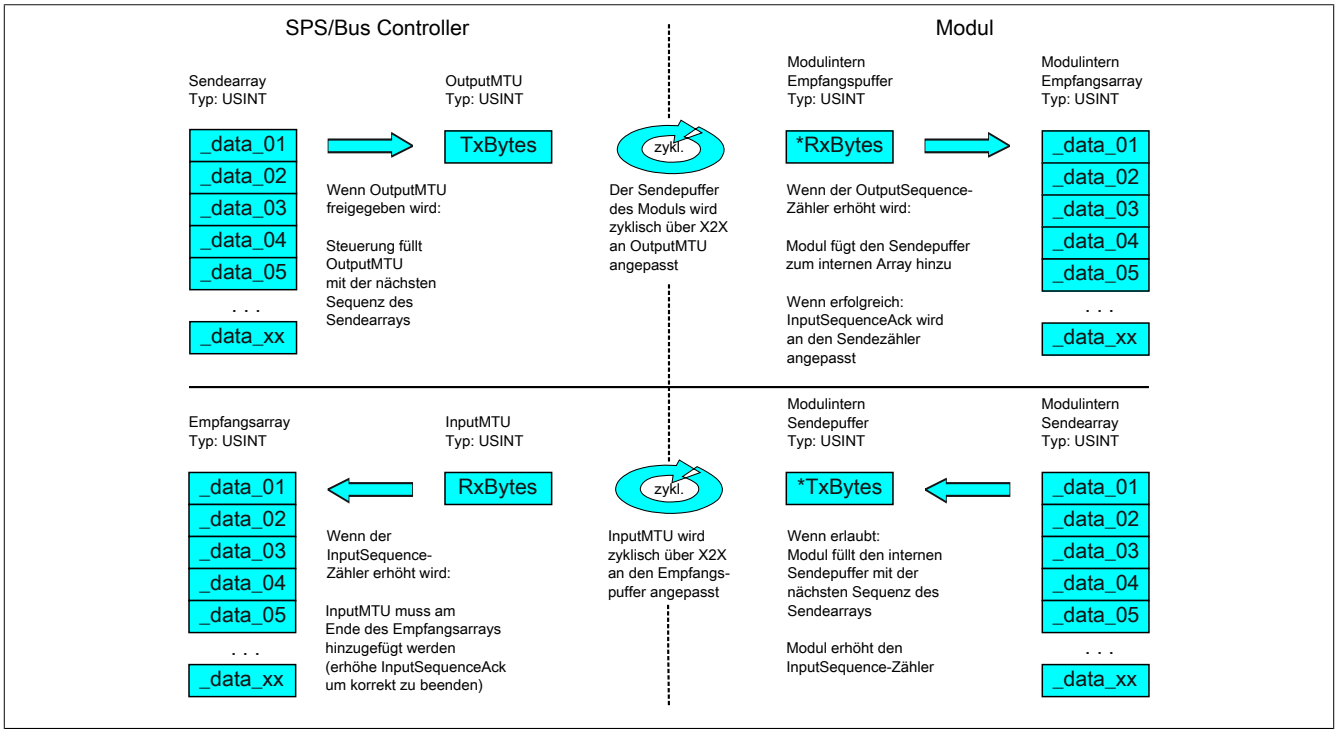


Abbildung 2: Kommunikation per Flatstream

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.

3.2.4 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 Steuerung 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.

3.2.4.1 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.

Information:

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

Information:

Die Register sind unter "Flatstream-Register" auf Seite 45 beschrieben.

3.2.4.2 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.

Information:

Die Register sind unter "Flatstream-Register" auf Seite 45 beschrieben.

3.2.4.2.1 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

3.2.4.2.2 Transport der Nutzdaten und der Controlbytes

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 Steuerung 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 Steuerung gewählt.

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

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

3.2.4.2.3 Kommunikationsstatus

Der Kommunikationsstatus wird über die Register "OutputSequence" und "InputSequence" ermittelt.

- OutputSequence enthält Informationen über den Kommunikationsstatus der Steuerung. Es wird von der Steuerung geschrieben und vom Modul gelesen.
- InputSequence enthält Informationen über den Kommunikationsstatus des Moduls. Es wird vom Modul geschrieben und sollte von der Steuerung nur gelesen werden.

Beziehung zwischen Output- und InputSequence

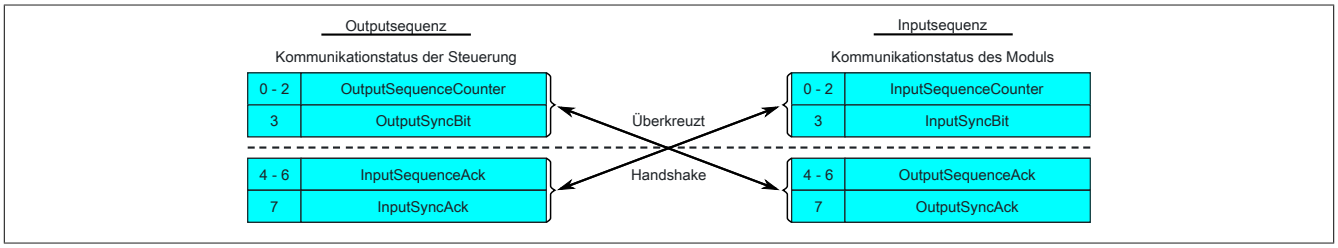


Abbildung 3: Zusammenhang zwischen Output- und InputSequence

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.

3.2.4.3 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 (Steuerung als Sender)

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

Algorithmus

1) Steuerung muss 000 in OutputSequenceCounter schreiben und OutputSyncBit zurücksetzen. Steuerung 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 Steuerung die erwarteten Werte in OutputSequenceAck und OutputSyncAck registriert, darf sie den OutputSequenceCounter inkrementieren. Die Steuerung 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 Steuerung die erwarteten Werte in OutputSequenceAck und OutputSyncAck registriert, darf sie das OutputSyncBit setzen. Die Steuerung 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, Steuerung kann Daten an Modul senden.

Synchronisation der Input-Richtung (Steuerung als Empfänger)

Die korrespondierenden Synchronisationsbits (InputSyncBit und InputSyncAck) sind zurückgesetzt. Aus diesem Grund können momentan keine Nachrichten vom Modul an die Steuerung 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) Steuerung darf den aktuellen Inhalt der InputMTU nicht übernehmen, weil der Kanal noch nicht synchronisiert ist. Steuerung 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) Steuerung darf den aktuellen Inhalt der InputMTU nicht übernehmen, weil der Kanal noch nicht synchronisiert ist. Steuerung 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) Steuerung 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 Steuerung senden.

3.2.4.4 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.

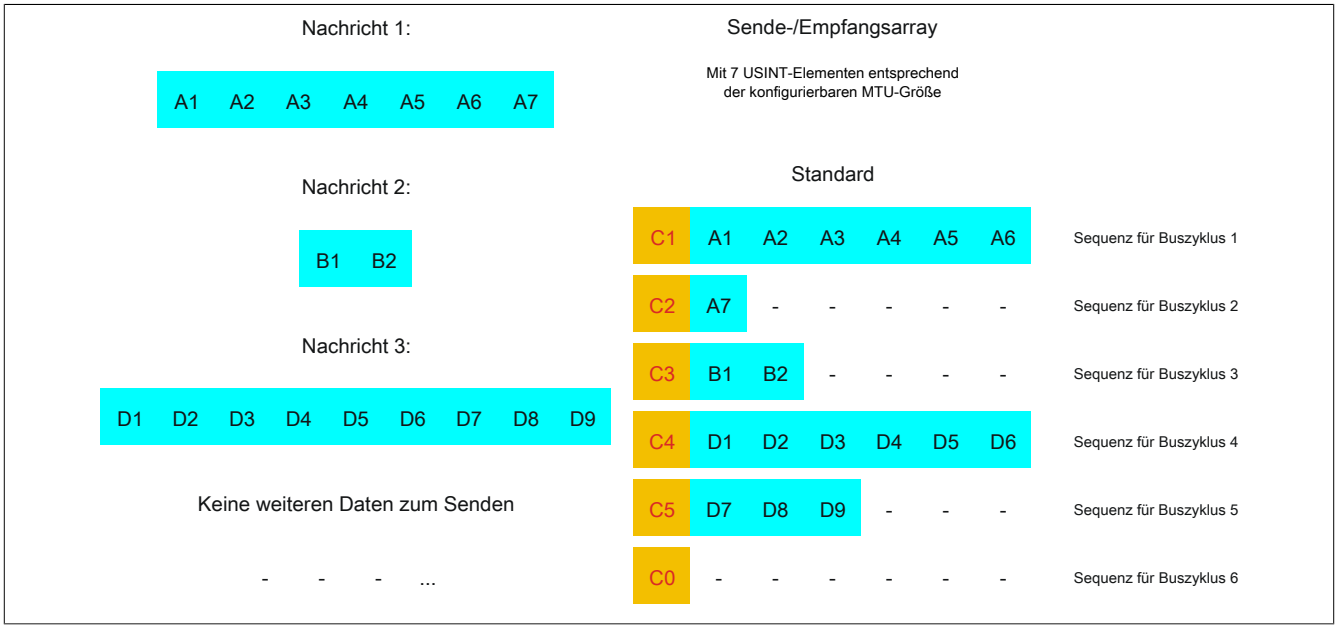


Abbildung 4: Sende-/Empfangsarray (Standard)

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)

3.2.4.4.1 Senden von Daten an ein Modul (Output)

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

Information:

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

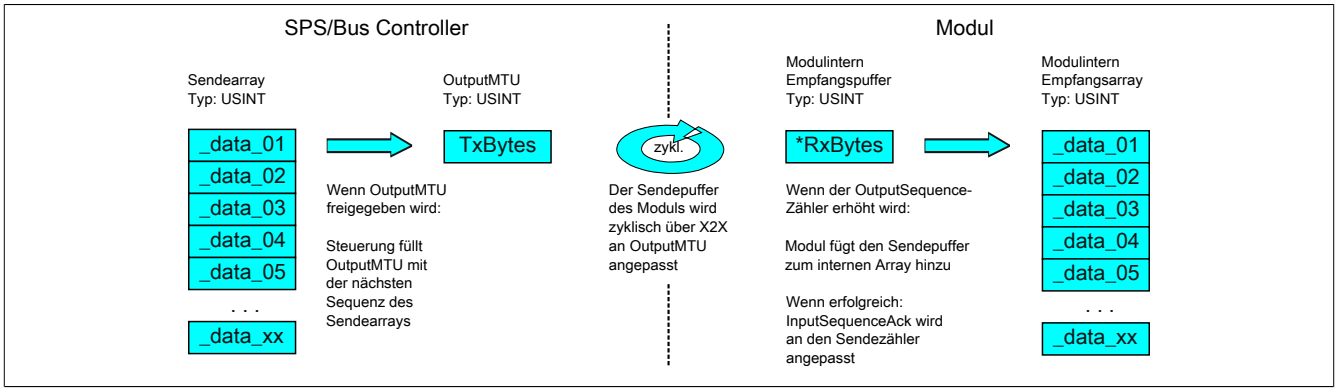


Abbildung 5: Kommunikation per Flatstream (Output)

Nachricht kleiner als OutputMTU

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

Algorithmus

Zyklische Statusabfrage: *- Modul überwacht OutputSequenceCounter*
0) Zyklische Prüfungen: - Steuerung muss OutputSyncAck prüfen → falls OutputSyncAck = 0; OutputSyncBit zurücksetzen und Kanal resynchronisieren - Steuerung muss Freigabe der OutputMTU prüfen → falls OutputSequenceCounter > InputSequenceAck; MTU nicht freigegeben, weil letzte Sequenz noch nicht bestätigt
1) Vorbereitung (Sendearray anlegen): - Steuerung muss Nachricht auf zulässige Segmente aufteilen und entsprechende Controlbytes bilden - Steuerung muss Segmente und Controlbytes zu Sendearray zusammenfügen
2) Senden: - Steuerung überträgt das aktuelle Element des Sendearrays in die OutputMTU → OutputMTU wird zyklisch in den Sendepuffer des Moduls übertragen, aber noch nicht weiterverarbeitet - Steuerung 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: - Steuerung 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

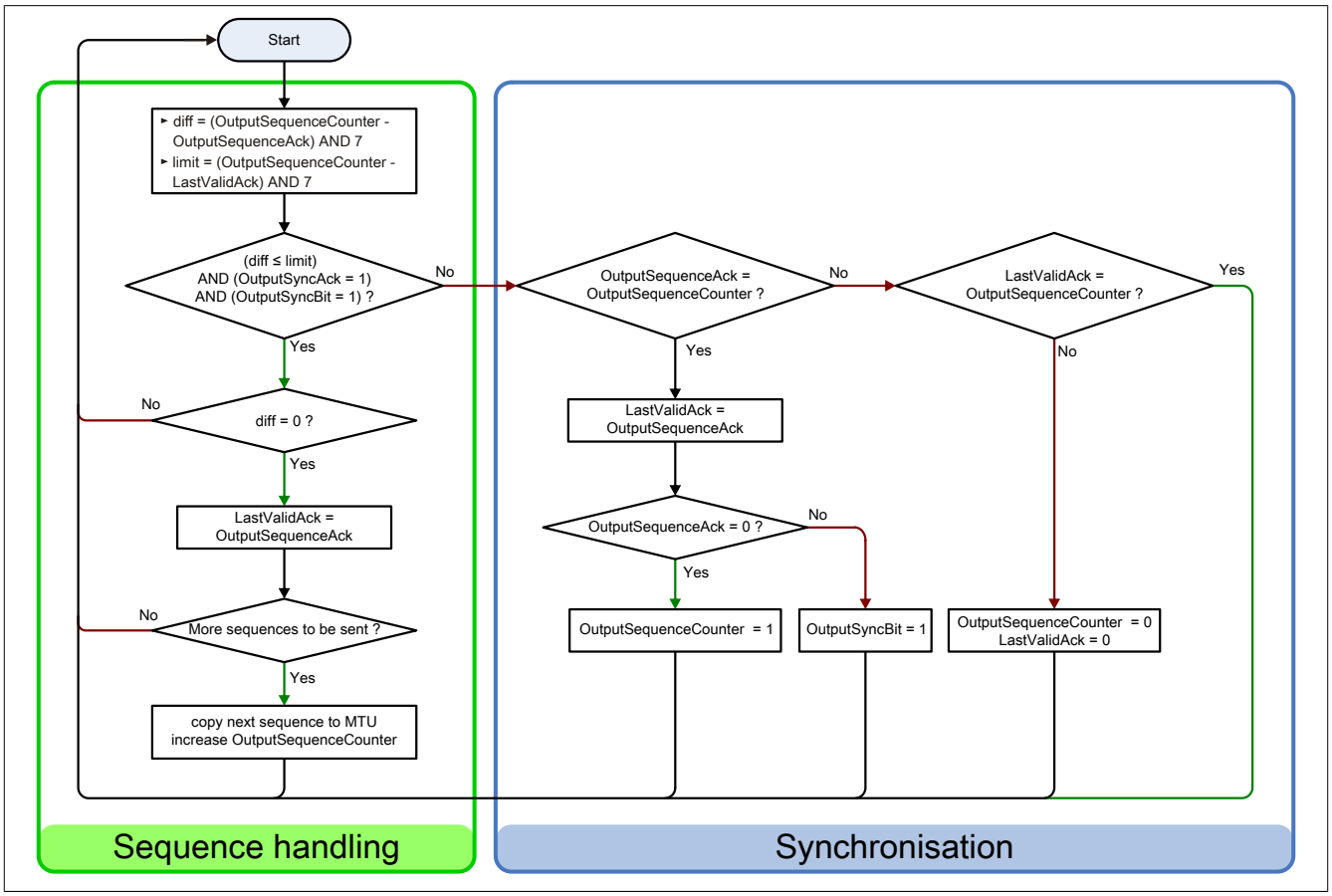


Abbildung 6: Ablaufdiagramm für Output-Richtung

3.2.4.4.2 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.

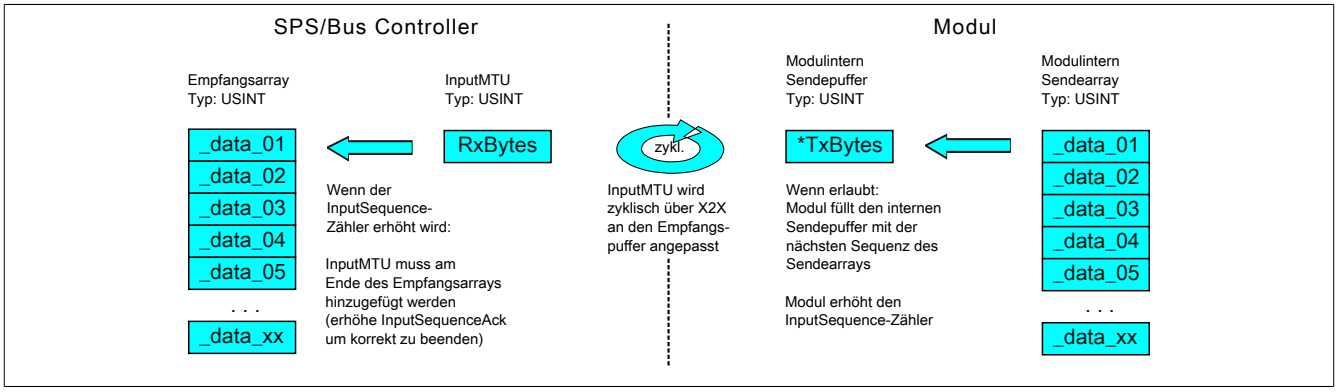


Abbildung 7: Kommunikation per Flatstream (Input)

Algorithmus

0) Zyklische Statusabfrage: - Steuerung 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): - Steuerung muss Daten aus InputMTU übernehmen und an das Ende des Empfangsarrays anfügen - Steuerung 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

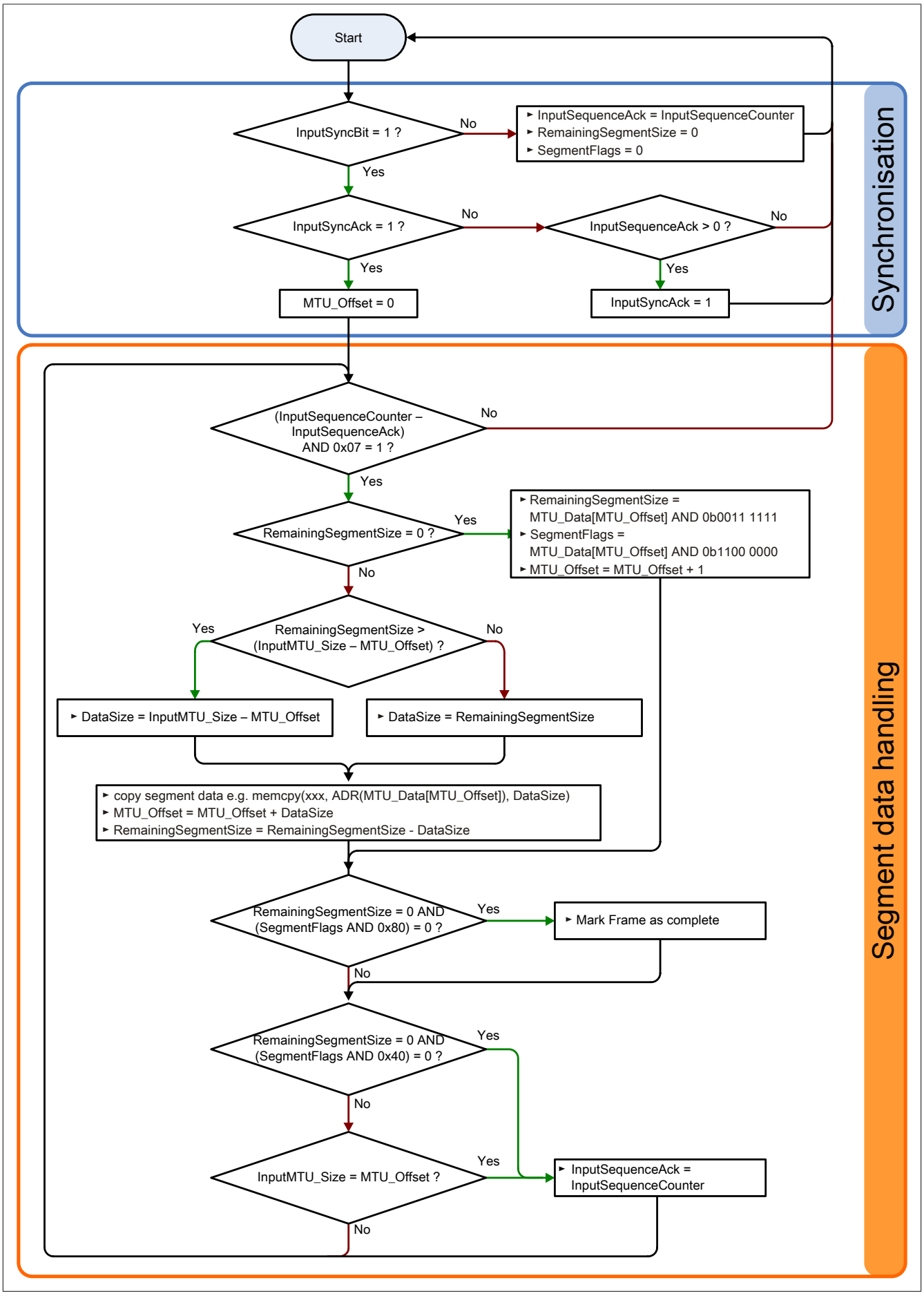


Abbildung 8: Ablaufdiagramm für Input-Richtung

3.2.4.4.3 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

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.

Information:

Beim Betrieb ohne Forward ist diese Situation sehr unwahrscheinlich.

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.

3.2.4.5 Flatstream-Modus

In Input-Richtung wird das Sende-Array automatisch generiert. Dem Anwender werden über den Flatstream-Modus mehrere Optionen zur Verfügung gestellt, um eine kompaktere Anordnung beim eintreffenden Datenstrom zu erlauben. Diese sind:

- Standard
- MultiSegmentMTU erlaubt
- Große Segmente erlaubt

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.

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.

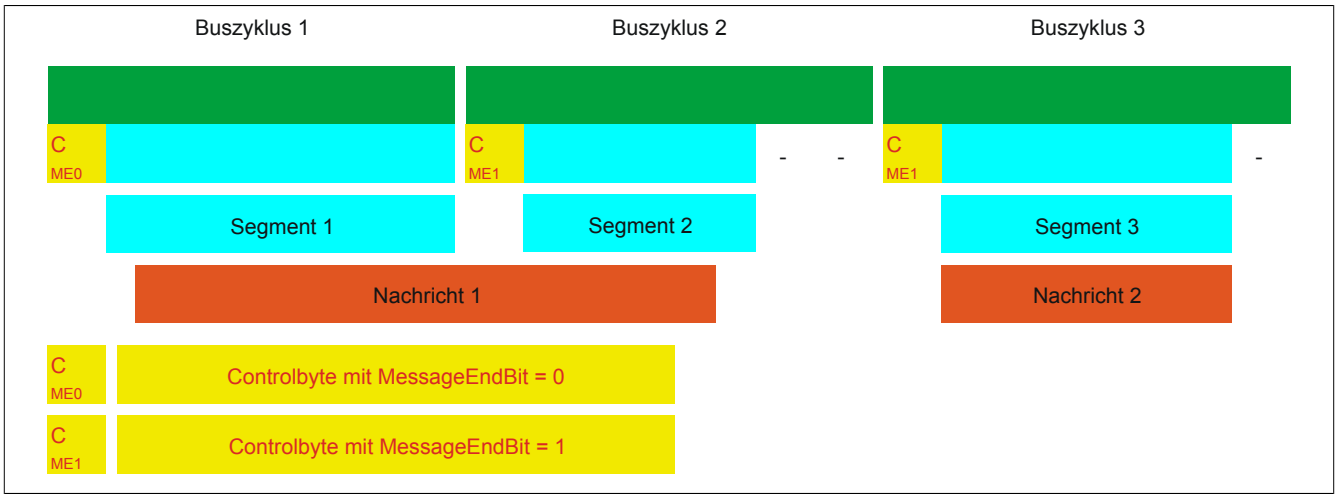


Abbildung 9: Anordnung von Nachrichten in der MTU (Standard)

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.

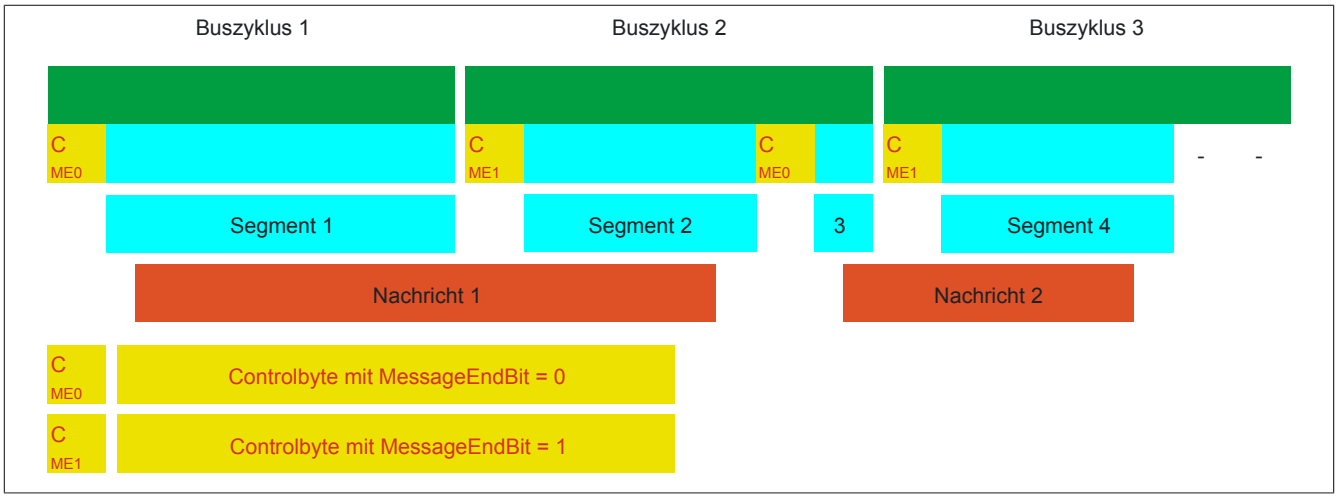


Abbildung 10: Anordnung von Nachrichten in der MTU (MultiSegmentMTU)

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.

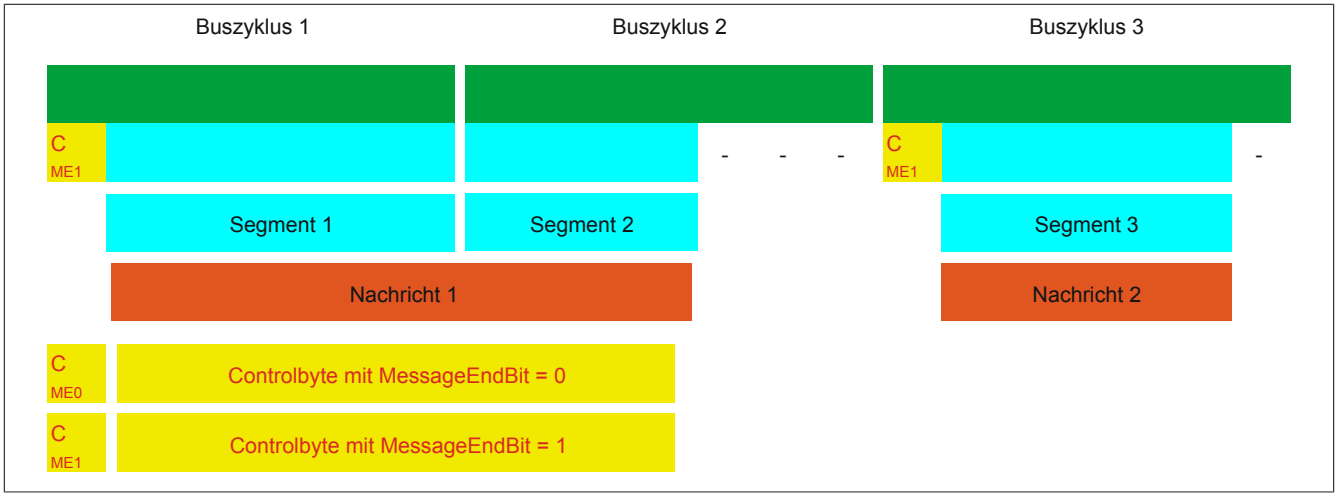


Abbildung 11: Anordnung von Nachrichten in der MTU (große Segmente)

Anwendung beider Optionen

Die beiden Optionen dürfen auch gleichzeitig angewendet werden.

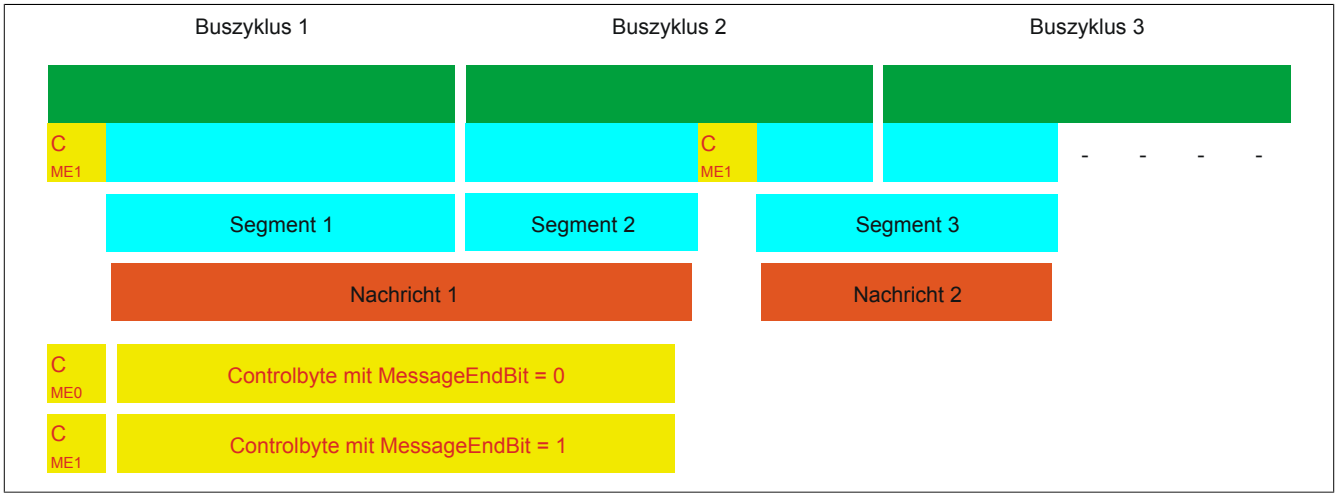


Abbildung 12: Anordnung von Nachrichten in der MTU (große Segmente und MultiSegmentMTU)

3.2.4.6 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.

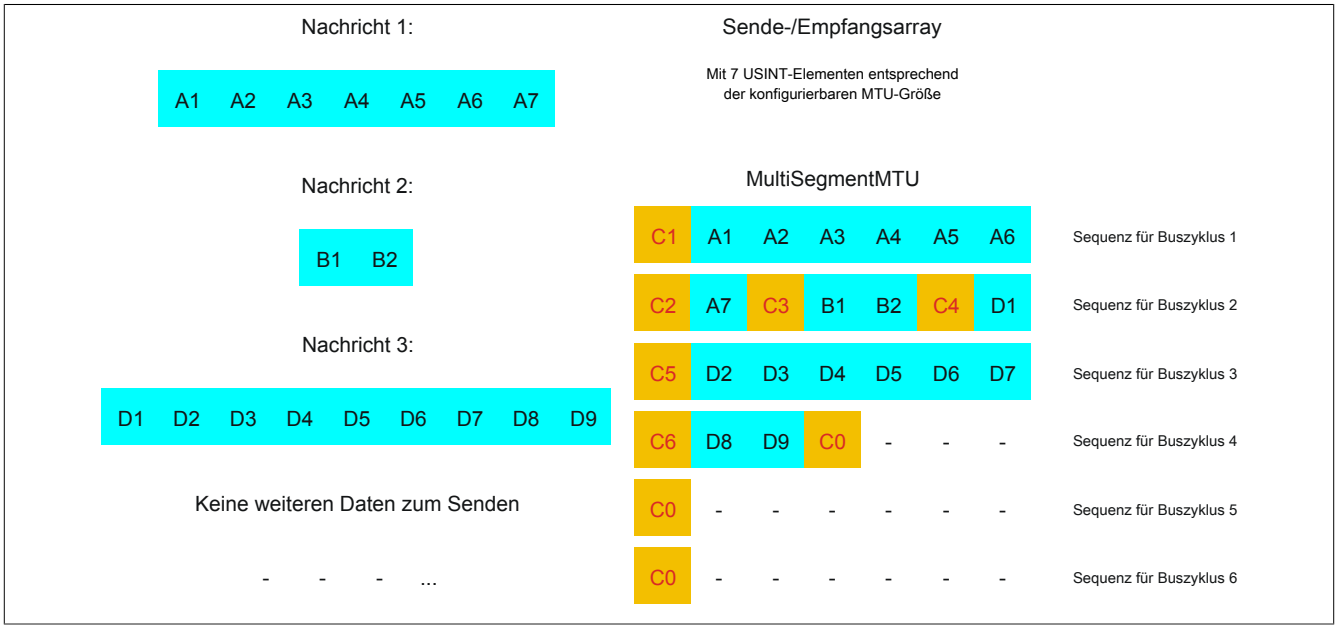


Abbildung 13: Sende-/Empfangsarray (MultiSegmentMTU)

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.

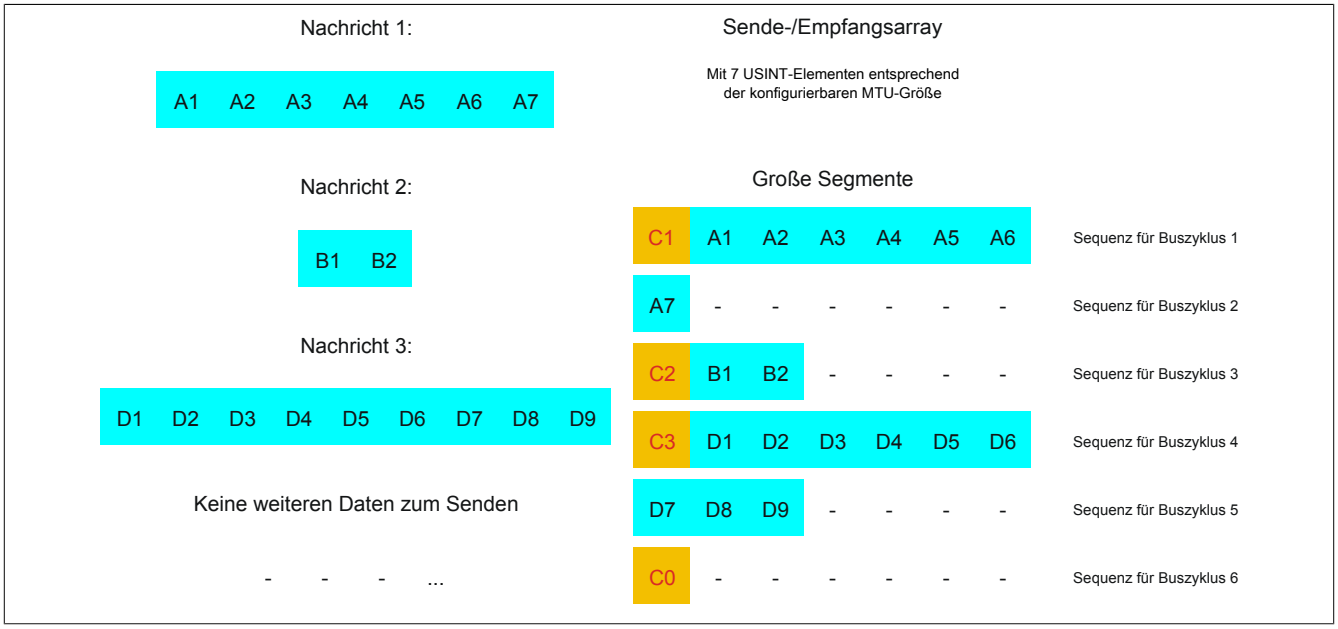


Abbildung 14: Sende-/Empfangsarray (große Segmente)

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.

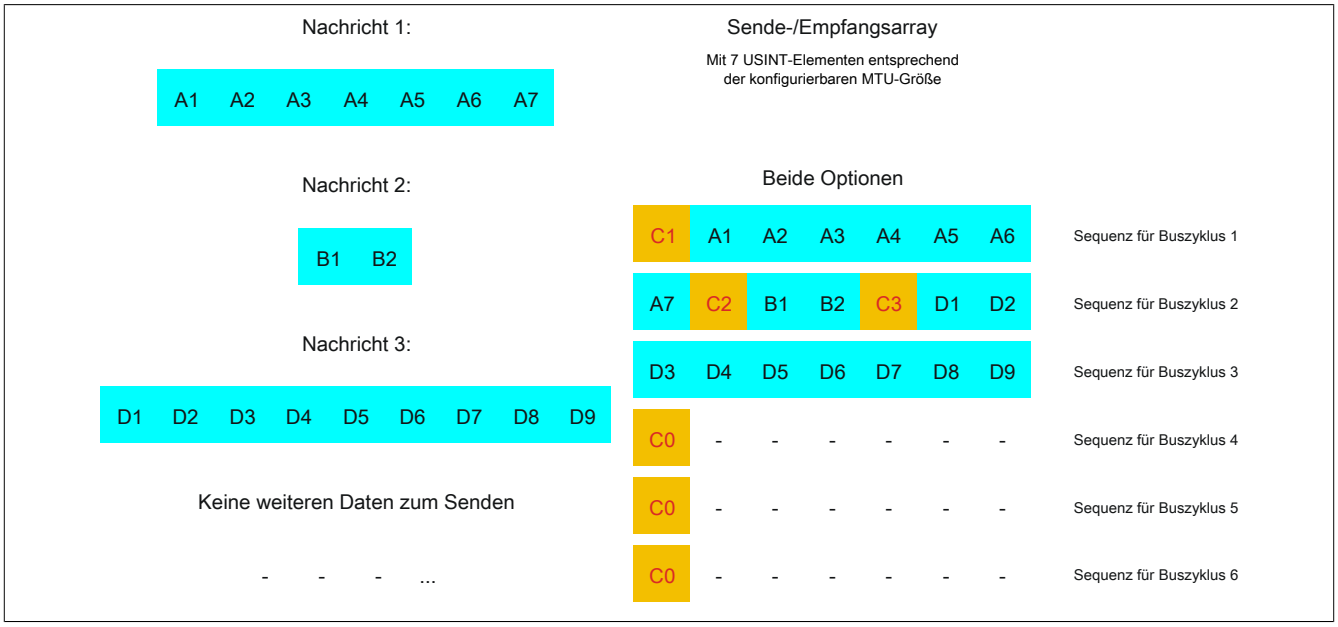


Abbildung 15: Sende-/Empfangsarray (große Segmente und MultiSegmentMTU)

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

3.2.5 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.

3.2.5.1 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)

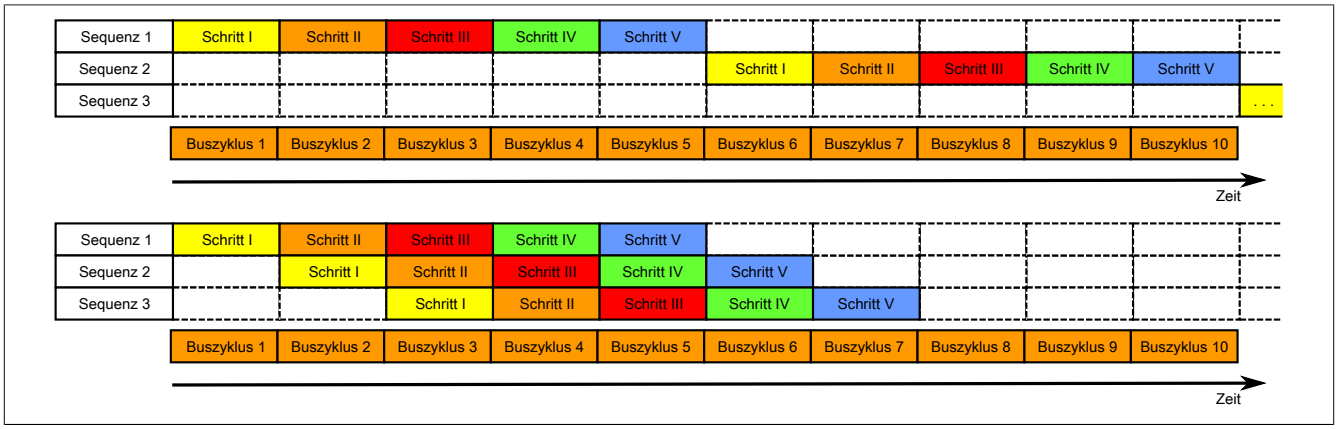


Abbildung 16: Vergleich Übertragung ohne bzw. mit Forward

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.

3.2.5.2 Konfiguration

Die Forward-Funktion muss nur für die Input-Richtung freigeschaltet werden. 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.

Information:

Die Register sind unter "Flatstream-Register" auf Seite 45 beschrieben.

3.2.5.2.1 Verzögerungszeit

Die Verzögerungszeit wird 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.

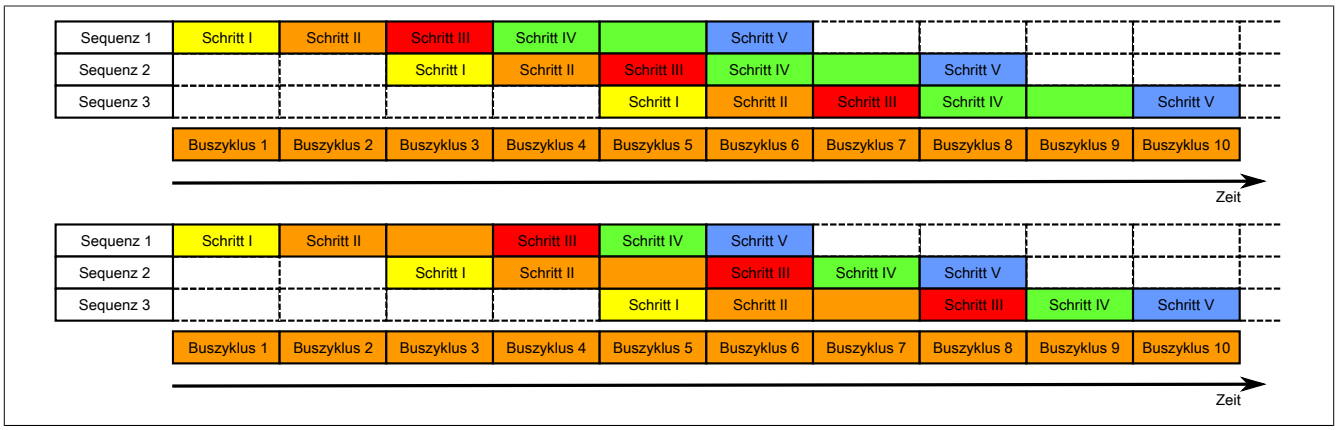


Abbildung 17: Auswirkung des ForwardDelay bei der Flatstream-Kommunikation mit Forward

Im Programmablauf muss sichergestellt werden, dass die Steuerung 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 Steuerung länger Zeit die eintreffende InputSequence bzw. InputMTU zu verarbeiten.

3.2.5.3 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: - Steuerung muss OutputSyncAck prüfen → falls OutputSyncAck = 0; OutputSyncBit zurücksetzen und Kanal resynchronisieren - Steuerung muss Freigabe der OutputMTU prüfen → falls OutputSequenceCounter > OutputSequenceAck + 7, in diesem Fall nicht freigegeben, weil letzte Sequenz noch nicht quittiert
1) Vorbereitung (Sendearray anlegen): - Steuerung muss Nachricht auf zulässige Segmente aufteilen und entsprechende Controlbytes bilden - Steuerung muss Segmente und Controlbytes zu Sendearray zusammenfügen
2) Senden: - Steuerung muss aktuellen Teil des Sendearrays in die OutputMTU übertragen - Steuerung muss OutputSequenceCounter erhöhen, damit Sequenz vom Modul übernommen wird - Steuerung 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): - Steuerung 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: - Steuerung 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): - Steuerung muss Daten aus InputMTU übernehmen und an das Ende des Empfangsarrays anfügen - Steuerung 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 Steuerung 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.

3.2.5.4 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).

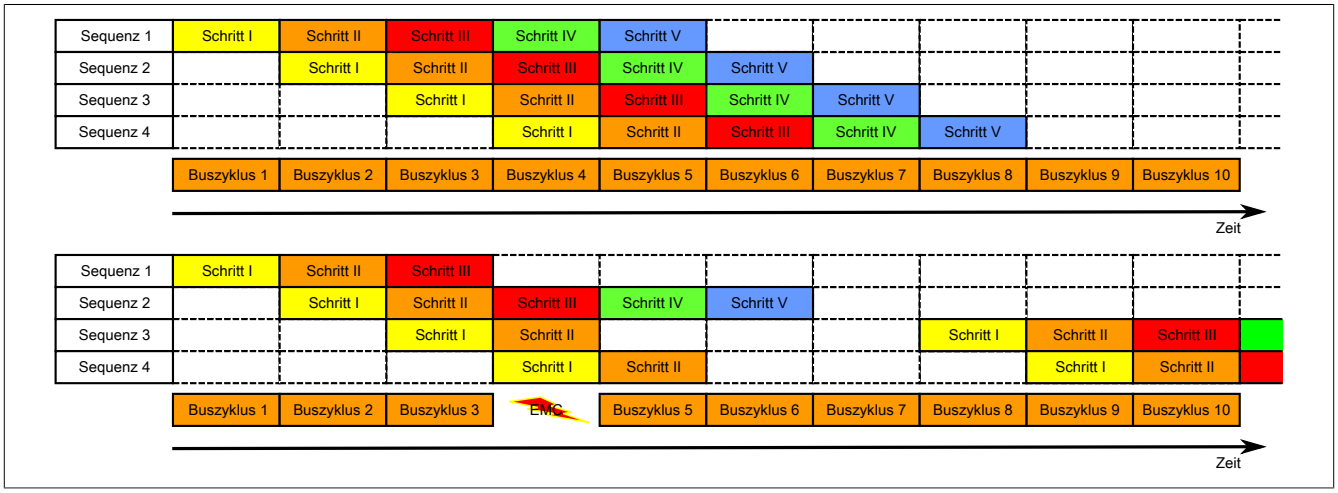


Abbildung 18: Auswirkung eines ausgefallenen Buszyklus

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.

4 Inbetriebnahme

4.1 Verwendung nach X20IF1091-1

Wird dieses Modul nach dem X2X Link Modul X20IF1091-1 betrieben, kann es zu Verzögerungen bei der Flatstream-Übertragung kommen. Für genauere Informationen siehe X20IF1091-1, Abschnitt "Datenübertragung am Flatstream".

4.2 Benutzung des Moduls mit SGC

Information:

Die Benutzung des Moduls mit SGC-Zielsystemen ist nur bei Verwendung des Funktionsmodells "Flatstream" oder "Flat" möglich.

5 UL-Zertifizierungsinformationen

Um das Modul entsprechend den UL-Standard zu installieren, müssen folgenden Regeln beachtet werden.

Information:

- **Nur Kupferkabel verwenden. Mindesttemperaturfestigkeit des Kabels, das an die Feldverdrahtungsklemmen angeschlossen wird: 61°C, 28 - 14 AWG.**
- **Alle Modelle sind für den Einsatz in einem abschließenden Sicherheitsgehäuse vorgesehen, das den Anforderungen zum Schutz vor Brandausbreitung entspricht und eine ausreichende Steifigkeit gemäß UL 61010-1 und UL 61010-2-201 aufweist.**
- **Externen Stromkreise, die an das Gerät angeschlossen werden, müssen galvanisch getrennt sein; durch verstärkte oder doppelte Isolierung vom Stromnetz bzw. von gefährlicher Spannung getrennt sein und die Anforderungen des SELV-/PELV-Stromkreises erfüllen.**
- **Bei nicht bestimmungsgemäßer Verwendung des Geräts kann der Schutz durch das Geräts beeinträchtigt werden.**
- **Reparaturen dürfen nur von B&R durchgeführt werden.**

6 Registerbeschreibung

6.1 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.

6.2 Funktionsmodell 0 - Flat

Im Funktionsmodell "Flat" werden die CAN-Informationen mittels zyklischer Ein- und Ausgangsregister übertragen. Alle Daten eines CAN-Objekts (8 CAN-Datenbytes, Identifier, Status, ...) sind als separate Datenpunkte zugänglich (siehe auch "Das CAN-Objekt" auf Seite 5).

Information:

- **Die Verwendung der Bibliotheken "ArCAN" und "CAN_Lib" ist nicht möglich.**

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Schnittstelle - Konfiguration						
257	ConfigBaudrate	USINT				•
259	ConfigSJW	USINT				•
261	ConfigSPO	USINT				•
266	ConfigTXtrigger	UINT				•
270	CfO_Fifosize_01	UINT				•
673	Cfo_FIFOTXlimit	USINT				•
677	Cfo_TXRXinfoFlags	USINT				•
Streamfilter - Konfiguration						
385	CfO_IF1DefaultCANFilterMode	USINT		•		•
380 + N*16	CfO_IF1CANFilter0N (Index N = 1 bis 4)	UDINT		•		•
388 + N*16	CfO_IF1CANFilterMask0N (Index N = 1 bis 4)	UDINT		•		•
Schnittstelle - Kommunikation						
641	TXCount	USINT			•	
513	TXCountReadBack	USINT	•			
545	TXCountLatchReadBack	USINT	•			
515	RXCount	USINT	•			
547	RXCountLatch	USINT	•			
Sendepuffer						
645	TXDataSize	USINT			•	
652	TXIdent	UDINT			•	
Index * 2 + 657	TXDataByte0 bis TXDataByte7	USINT			•	
Index * 4 + 658	TXDataWord0 bis TXDataWord3	UINT			•	
Index * 8 + 660	TXDataLong0 bis TXDataLong1	UDINT			•	
Empfangspuffer 0						
517	RXDataSize0	USINT	•			
524	RXIdent0	UDINT	•			
Index * 2 + 529	RXData0Byte0 bis RXData0Byte7	USINT	•			
Index * 4 + 530	RXData0Word0 bis RXData0Word3	UINT	•			
Index * 8 + 532	RXData0Long0 bis RXData0Long1	UDINT	•			
Empfangspuffer 1						
549	RXDataSize1	USINT	•			
556	RXIdent1	UDINT	•			
Index * 2 + 561	RXData1Byte0 bis RXData1Byte7	USINT	•			
Index * 4 + 562	RXData1Word0 bis RXData1Word3	UINT	•			
Index * 8 + 564	RXData1Long0 bis RXData1Long1	UDINT	•			

6.3 Funktionsmodell 2 - Stream und Funktionsmodell 254 - Cyclicstream

Die Funktionsmodelle "Stream" und "Cyclicstream" nutzen einen modulspezifischen Treiber des Betriebssystems der Steuerung. Die Schnittstelle kann mit Hilfe der Bibliotheken "ArCAN" und "CAN_Lib" gesteuert und während der Laufzeit umkonfiguriert werden.

Funktionsmodell "Stream"

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

Funktionsmodell "Cyclicstream"

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

Information:

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

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Modul - Konfiguration						
-	AsynSize	-				
Schnittstelle - Konfiguration						
270	CfO_Fifosize_01	UINT				•
6273	CfO_ErrorID0007	USINT				•
Streamfilter - Konfiguration						
385	CfO_IF1DefaultCANFilterMode	USINT		•		•
380 + N*16	CfO_IF1CANFilter0N (Index N = 1 bis 4)	UDINT		•		•
388 + N*16	CfO_IF1CANFilterMask0N (Index N = 1 bis 4)	UDINT		•		•
Schnittstelle - Kommunikation						
6145	CAN-Fehlerstatus	USINT	•			
	CANwarning	Bit 0				
	CANpassive	Bit 1				
	CANbusoff	Bit 2				
	CANRXoverrun	Bit 3				
6209	CAN-Fehlerquittierung	USINT			•	
	QuitCANwarning	Bit 0				
	QuitCANpassive	Bit 1				
	QuitCANbussoff	Bit 2				
	QuitCANRXoverrun	Bit 3				

6.4 Funktionsmodell 254 - Flatstream

Der Flatstream ermöglicht eine unabhängige Kommunikation zwischen einem X2X-Master und dem Modul. Für das CAN-Modul wurde diese Schnittstelle als separates Funktionsmodell implementiert. Die CAN-Informationen (Identifier, Status usw.) werden mittels zyklischer Ein- und Ausgangsregister übertragen. Zur Steuerung dieses Datenstroms werden die sogenannten Sequenz- und Steuerbytes genutzt (siehe "Die Flatstream-Kommunikation" auf Seite 6).

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

Information:

- **Die Verwendung der Bibliotheken "ArCAN" und "CAN_Lib" ist nicht möglich.**
- **Im Vergleich zum Funktionsmodell "Flat" sind höhere Datenraten zwischen X2X-Master und Modul möglich.**

Register	Name	Datentyp	Lesen		Schreiben	
			Zyklisch	Azyklisch	Zyklisch	Azyklisch
Schnittstelle - Konfiguration						
257	ConfigBaudrate	USINT				•
259	ConfigSJW	USINT				•
261	ConfigSPO	USINT				•
266	ConfigTXtrigger	UINT				•
270	CfO_Fifosize_01	UINT				•
6273	CfO_ErrorID0007	USINT				•
Streamfilter - Konfiguration						
385	CfO_IF1DefaultCANFilterMode	USINT		•		•
380 + N*16	CfO_IF1CANFilter0N (Index N = 1 bis 4)	UDINT		•		•
388 + N*16	CfO_IF1CANFilterMask0N (Index N = 1 bis 4)	UDINT		•		•
Schnittstelle - Kommunikation						
6145	CAN-Fehlerstatus	USINT	•			
	CANwarning	Bit 0				
	CANpassive	Bit 1				
	CANbusoff	Bit 2				
	CANRXoverrun	Bit 3				
6209	CAN-Fehlerquittierung	USINT			•	
	QuitCANwarning	Bit 0				
	QuitCANpassive	Bit 1				
	QuitCANbussoff	Bit 2				
	QuitCANRXoverrun	Bit 3				
Flatstream - Konfiguration						
193	outputMTU	USINT				•
195	inputMTU	USINT				•
197	mode	USINT				•
199	forward	USINT				•
206	forwardDelay	UINT				•
Flatstream - Kommunikation						
0	InputSequence	USINT	•			
Index * 1 + 0	RxByte1 bis RxByte27	USINT	•			
32	OutputSequence	USINT			•	
Index * 1 + 32	TxByte1 bis TxByte27	USINT			•	

6.5 Funktionsmodell 254 - Bus Controller

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

Register	Offset[1)]	Name	Datentyp	Lesen		Schreiben	
				Zyklisch	Azyklisch	Zyklisch	Azyklisch
Schnittstelle - Konfiguration							
257	-	ConfigBaudrate	USINT				•
259	-	ConfigSJW	USINT				•
261	-	ConfigSPO	USINT				•
266	-	ConfigTXtrigger	UINT				•
270	-	CfO_Fifosize_01	UINT				•
6273	-	CfO_ErrorID0007	USINT				•
Streamfilter - Konfiguration							
385	-	CfO_IF1DefaultCANFilterMode	USINT		•		•
380 + N*16	-	CfO_IF1CANFilter0N (Index N = 1 bis 4)	UDINT		•		•
388 + N*16	-	CfO_IF1CANFilterMask0N (Index N = 1 bis 4)	UDINT		•		•
Schnittstelle - Kommunikation							
6145	-	CAN-Fehlerstatus	USINT		•		
		CANwarning	Bit 0				
		CANpassive	Bit 1				
		CANbusoff	Bit 2				
		CANRXoverrun	Bit 3				
6209	-	CAN-Fehlerquittierung	USINT				•
		QuitCANwarning	Bit 0				
		QuitCANpassive	Bit 1				
		QuitCANbussoff	Bit 2				
		QuitCANRXoverrun	Bit 3				
Flatstream - Konfiguration							
193	-	outputMTU	USINT				•
195	-	inputMTU	USINT				•
197	-	mode	USINT				•
199	-	forward	USINT				•
206	-	forwardDelay	UINT				•
Flatstream - Kommunikation							
0	0	InputSequence	USINT	•			
Index * 1 + 0	Index * 1 + 0	RxByte1 bis RxByte7	USINT	•			
32	0	OutputSequence	USINT			•	
Index * 1 + 32	Index * 1 + 0	TxByte1 bis TxByte7	USINT			•	

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

6.5.1 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".

6.5.2 CAN-I/O Bus Controller

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

6.6 Schnittstelle - Konfiguration

6.6.1 Übertragungsrate

Name:
ConfigBaudrate

"Baudrate" in der Automation Studio I/O-Konfiguration.

Konfiguration der CAN-Übertragungsrate für die Schnittstelle.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	0

Bitstruktur:

Bit	Beschreibung	Wert	Information
0 - 3	Übertragungsrate	0	Schnittstelle ausgeschaltet (Bus Controller Default)
		1	10 kBit/s
		2	20 kBit/s
		3	50 kBit/s
		4	100 kBit/s
		5	125 kBit/s
		6	250 kBit/s
		7	500 kBit/s
		8	800 kBit/s
		9	1000 kBit/s
4 - 7	Reserviert	-	

6.6.2 Synchronisationssprungweite

Name:
ConfigSJW

"Synchronisationssprungweite" in der Automation Studio I/O-Konfiguration.

Die Synchronisationssprungweite (synchronisation jump width, SJW) dient dazu, innerhalb eines CAN-Telegramms die Abtastzeitpunkte nachzusynchronisieren.

Eine genauere Beschreibung für die Synchronisationssprungweite kann der CAN-Spezifikation entnommen werden.

Datentyp	Werte	Bedeutung
USINT	1 bis 4	Synchronisationssprungweite; Bus Controller Default: 3

6.6.3 Offset für den Abtastzeitpunkt

Name:
ConfigSPO

"Abtastzeitpunkt Offset" in der Automation Studio I/O-Konfiguration.

Offset für den Abtastzeitpunkt der einzelnen Bits am CAN-Bus.

Eine genauere Beschreibung des Abtastzeitpunkt-Offsets kann der CAN-Spezifikation entnommen werden.

Datentyp	Werte	Bedeutung
USINT	0 bis 1	Bus Controller Default: 0

6.6.4 Start des Sendevorgangs

Name:
ConfigTXtrigger

"TX Objekte / TX Trigger" in der Automation Studio I/O-Konfiguration.

Bestimmt die Anzahl der CAN-Objekte die in den Sendepuffer übertragen werden müssen, bevor der Sendevorgang gestartet wird.

Datentyp	Werte	Bedeutung
UINT	0 bis 8	Anzahl der CAN-Objekte im Sendepuffer bevor der Sendevorgang gestartet wird; Bus Controller Default: 1

6.6.5 Konfiguration der Fehlermeldungen

Name:
CfO_ErrorID0007

Mit diesem Register müssen die zu übertragenden Fehlermeldungen zuerst konfiguriert werden. Wenn das entsprechende Aktivierungsbit nicht gesetzt ist, wird beim Auftreten des Fehlers auch kein Fehlerstatus an das übergeordnete System gemeldet.

Datentyp	Werte	Bus Controller Default
USINT	Siehe Bitstruktur	0

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	CANwarning	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
1	CANpassive	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
2	CANbussoff	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
3	CANRXoverrun	0	Deaktiviert (Bus Controller Default)
		1	Aktiviert
4 - 7	Reserviert	-	

6.6.6 Größe des Sendepuffers

Name:
Cfo_FIFOTXlimit

"TX FIFO Größe" in der Automation Studio I/O-Konfiguration.

Bestimmt die Größe des Sendepuffers für die jeweilige Schnittstelle.

Datentyp	Werte	Bedeutung
USINT	0 bis 18	Größe des Sendepuffers

6.6.7 Größe des Fifo Speichers

Name:
CfO_Fifosize_01

"Größe des Fifo Speichers" in der Automation Studio I/O-Konfiguration.

Bestimmt die Größe des Fifo Speichers für die jeweilige Schnittstelle.

Datentyp	Werte	Bedeutung
UINT	20 bis 4096	Größe des Fifo Speichers in Byte

6.6.8 Anzeige der noch unverarbeiteten Elemente im Sende- bzw. Empfangspuffer

Name:
Cfo_TXRXinfoFlags

Mit Hilfe dieser Register kann für die Schnittstelle konfiguriert werden, dass in den oberen 4 Bits der Register "TXCountReadBack" und "RXCount" die Anzahl der noch unverarbeiteten Elemente im Sende- bzw. Empfangspuffer angezeigt wird.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	TxFifoInfo "Mode of channel TXCountReadBack" in der Automation Studio I/O-Konfiguration	0	In den Registern "TXCountReadBack" auf Seite 41 und "TXCountLatchReadBack" auf Seite 42 wird der "TXCount" zurückgelesen.
		1	In den unteren 4 Bits der Register "TXCountReadBack" auf Seite 41 und "TXCountLatchReadBack" auf Seite 42 wird der "TXCount" zurückgelesen. In den oberen 4 Bits wird die Anzahl der ungesendeten Frames im Sendepuffer zurückgegeben.
1	RxFifoInfo "Mode of channel RXCount" in der Automation Studio I/O-Konfiguration	0	In den Registern "RXCount" auf Seite 42 und "RXCountLatch" auf Seite 42 wird die Anzahl der empfangenen Telegramme angezeigt.
		1	In den unteren 4 Bits der Register "RXCount" auf Seite 42 und "RXCountLatch" auf Seite 42 wird die Anzahl der empfangenen Telegramme angezeigt. In den oberen 4 Bits wird die Anzahl der empfangenen und noch nicht quittierten Telegramme im Empfangspuffer angezeigt.
2 - 7	Reserviert	-	

6.6.9 Streamfilter

Je CAN-Schnittstelle können bis zu 4 Stream-Filter konfiguriert werden. Diese bestimmen, welche CAN-IDs über den CyclicStream an die Steuerung weitergeleitet werden.
Die Filter werden in der numerischen Reihenfolge durchlaufen. Der erste zur eintreffenden CAN-Nachricht passende Filter wird verwendet, alle weiteren Filter werden ignoriert. Wenn kein Filter zur eintreffenden CAN Nachricht passt, bestimmt eine globale Konfiguration, ob die Nachricht verworfen oder akzeptiert wird (Default: Nachricht akzeptieren).
Jeder Filter hat eine einstellbare ID und eine einstellbare Filtermaske. Es werden nur jene Bits der ID verglichen, welche in der Maske mit 0 gesetzt sind.

6.6.9.1 CANFilterMode

Name:
CfO_IF1DefaultCANFilterMode

Diese Register geben die default Einstellungen für IDs an, die keinem der eingestellten Filter entsprechen.

Datentyp	Werte	Information
USINT	0	Kein Ansprechen des Filters, CAN-Frame wird verworfen.
	1	Kein Ansprechen des Filters, CAN-Frame wird über Stream übertragen.

6.6.9.2 CANFilter

Name:
CfO_IF1CANFilter01 bis CfO_IF1CANFilter04

In diesen Registern sind die Filtereigenschaften definiert.

Datentyp	Werte
UDINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0 bis 28	Filter-ID	x	Zu filternder Identifier-Wert.[1)]
29	Frameformat	0	Standard-Frameformat (SFF) mit 11-Bit Identifier; Mögliche FilterID-Werte: 0 bis 2047 (0x7FF)
		1	Extended-Frameformat (EFF) mit 29-Bit Identifier; Mögliche FilterID-Werte: 0 bis 536 870 911 (0x1FFFFFFF)
30	Reserviert	-	
31	Enable	0	Filter inaktiv
		1	Filter aktiv

1) Dieser Wert wird mit dem Identifier-Wert und dem Maskenwert verknüpft (siehe Beispiel).

Beispiele

Beispiel 1

Das folgende Beispiel zeigt den Zusammenhang zwischen Filtermaske, Filter-ID und den tatsächlich empfangenen 11-Bit CAN-Nachrichten.

Filtermaske[1)]	Filter-ID	CAN-Nachricht-ID	Information
000 0011 1110	110 0100 0000	110 0110 1010	Relevante Bits von Filter-ID und CAN-Nachricht sind identisch → Filter spricht an, der Frame wird entsprechend der Moduseinstellung verworfen oder weitergeleitet.
000 0011 1110	110 0100 0000	110 0110 1011	Relevante Bits nicht identisch → Nächster Filter bzw. Defaultmodus wird bearbeitet
000 0011 1111	110 0100 0000	110 0110 1011	Relevante Bits von Filter-ID und CAN-Nachricht sind identisch → Filter spricht an, der Frame wird entsprechend der Moduseinstellung verworfen oder weitergeleitet
000 0001 1111	110 0100 0000	110 0110 1011	Relevante Bits nicht identisch → Nächster Filter bzw. Defaultmodus wird bearbeitet

1) Rot = relevante bits

Beispiel 2

Konfiguration von 2 Filtern mit unterschiedlichem Filtermodus.

	Modus	Filter-ID	Filtermaske	Beschreibung
Filter 1	ignore	16#300	16#07F	CANopen PDO2
Filter 2	accept	16#005	16#780	NodeID 5
Default	ignore			

Bei CANopen sind die 11-Bit CAN-IDs (COB-IDs) aus 4-Bit Funktionscode und 7-Bit NodeID zusammengesetzt. Hier werden zunächst alle CANopen PDO2 Objekte abgewiesen. Danach werden nur die Frames vom CAN Device mit der NodeID 5 angenommen.

6.6.9.3 CANFilterMask

Name:
CfO_IF1CANFilterMask01 bis CfO_IF1CANFilterMask04

In diesen Registern sind Filtermaske und Filtermode definiert.

Datentyp	Werte
UDINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0 bis 28	Filtermaske	x	Vergleichs-Bitmuster für Filter-ID[1)]
29	Frameformat Maske	0	Das Frameformat der empfangenen Nachricht muss mit der Konfiguration in "CfO_IF1CANFilter" auf Seite 39 übereinstimmen.
		1	Der Filter bezieht sich auf beide Frameformate. 11-Bit und 29-Bit Identifier werden gefiltert.
30	Reserviert	-	
31	Modus	0	Bei Ansprechen des Filters wird der CAN-Frame übertragen.
		1	Bei Ansprechen des Filters wird der CAN-Frame verworfen.

1) Es werden nur jene Bits der Filter-ID verglichen, welche in der Maske auf 0 gesetzt sind (siehe "Beispiele" auf Seite 39).

6.7 Schnittstelle - Kommunikation

6.7.1 CAN-Fehlerstatus

Name:
CAN-Fehlerstatus

Die Bits in diesem Register zeigen die im CAN-Protokoll festgelegten Fehlerzustände an. Wenn ein Fehler auftritt, wird das entsprechende Bit gesetzt. Damit ein Fehlerbit wieder rückgesetzt wird, muss das entsprechende Bit quittiert werden (siehe "CAN-Fehlerquittierung" auf Seite 41).

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	CANwarning	0	Kein Fehler
		1	CANwarning Fehler auf IF1 aufgetreten
1	CANpassive	0	Kein Fehler
		1	CANpassive Fehler auf IF1 aufgetreten
2	CANbusoff	0	Kein Fehler
		1	CANbusoff Fehler auf IF1 aufgetreten
3	CANRXoverrun	0	Kein Fehler
		1	CANRXoverrun Fehler auf IF1 aufgetreten
4 - 7	Reserviert	-	

CANwarning

Am CAN-Bus wurde ein fehlerhafter Frame erkannt. Dazu gehören z. B. Bit-Fehler, Bit-Stuffing-Fehler, CRC-Fehler, Formatfehler im Telegramm und Acknowledgement-Fehler.

CANpassive

Der interne Sende- und/oder Empfangs-Fehlerzähler ist größer als 127. Die CAN-Kommunikation läuft noch, jedoch kann die Schnittstelle nur mehr einen "passiven Errorframe" absetzen. Ebenso bekommen "Error passive Teilnehmer" nur mehr verzögert die Möglichkeit, neue Telegramme zu senden.

CANbusoff

Der interne Sende-Fehlerzähler ist größer als 255. Der Bus wird abgeschaltet und es findet keine CAN-Kommunikation mit dem Modul mehr statt.

CANRXoverrun

Im Empfangspuffer des Moduls ist ein Überlauf aufgetreten.

6.7.2 CAN-Fehlerquittierung

Name:
CAN-Fehlerquittierung

Durch Setzen des jeweiligen Bits in diesem Register wird der dem Bit zugeordnete Fehler quittiert und das entsprechende Bit im Register "CAN-Fehlerstatus" gelöscht. Die Anwendung teilt damit dem Modul mit, dass sie den Fehlerstatus erkannt hat.

Datentyp	Werte
USINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	QuitCANwarning	0	Keine Quittierung
		1	Quittiere CANwarning Fehler auf IF1
1	QuitCANpassive	0	Keine Quittierung
		1	Quittiere CANpassive Fehler auf IF1
2	QuitCANbussoff	0	Keine Quittierung
		1	Quittiere CANbusoff Fehler auf IF1
3	QuitCANRXoverrun	0	Keine Quittierung
		1	Quittiere CANRXoverrun Fehler auf IF1
4 - 7	Reserviert	-	

6.7.3 Neues CAN-Telegramm für Sendepuffer

Name:
TXCount

Durch Erhöhen dieses Wertes teilt die Anwendung dem Modul mit, dass ein neues CAN-Telegramm in den Sendepuffer übertragen werden soll.

Datentyp	Werte
USINT	0 bis 255

6.7.4 Rücklesen von "TXCount"

Name:
TXCountReadBack

Der Wert des "TXCount" wird vom Modul in dieses Register umkopiert. Dadurch kann der Anwendungstask verifizieren, dass die Daten für das CAN-Telegramm vom Modul richtig übernommen wurden.

Die Bedeutung des Wertes ist abhängig vom Bit "TxFifoInfo". Dieses befindet sich im Register "Cfo_TXRXinfoFlags" auf Seite 38.

Datentyp	Werte	Bit "TxFifoInfo"	Bedeutung
USINT	0 bis 255	0	Rückgelesener "TXCount"
		1	Siehe Bitstruktur

Bitstruktur:

Bit	Bedeutung	Werte	Information
0 - 3	Rückgelesener "TXCount"	0 bis 15	Nur die unteren 4 Bits
4 - 7	Anzahl der noch ungesendeten Frames im Sendepuffer	0 bis 15	Wenn diese Anzahl den Wert 15 übersteigt (maximal 18 möglich), wird der Wert 15 zurückgegeben

6.7.5 "TXCount" vom vorangegangenen Zyklus rücklesen

Name:
TXCountLatchReadBack

In dieses Register wird vom Modul der Wert des "TXCount" vom vorangegangenen Zyklus kopiert. Dadurch kann im Falle eines Übertragungsfehlers am X2X Link oder POWERLINK Netzwerk verifiziert werden, ob der Fehler auf dem Weg von der Steuerung zum Modul oder auf dem Weg vom Modul zur Steuerung aufgetreten ist (siehe "Betrachtung von Fehlerfällen beim Senden" auf Seite 43).

Die Bedeutung des Wertes ist abhängig vom Bit "TxFifoInfo" im Register "Cfo_TXRXinfoFlags" auf Seite 38.

Datentyp	Werte	Bit "TxFifoInfo"	Bedeutung
USINT	0 bis 255	0	Rückgelesener "TXCount" vom vorangegangenen Zyklus
		1	Siehe Bitstruktur

Bitstruktur:

Bit	Bedeutung	Werte	Information
0 - 3	Rückgelesener "TXCount" vom vorangegangenen Zyklus	0 bis 15	Nur die unteren 4 Bits
4 - 7	Anzahl der noch ungesendeten Frames im Sendepuffer	0 bis 15	Vom vorangegangenen Zyklus

6.7.6 Zähler für empfangene CAN-Telegramme

Name:
RXCount

Dieser Zähler wird mit jedem empfangenen CAN-Telegramm um 1 erhöht. Der Anwendungstask kann damit den Empfang neuer Daten erkennen und diese entsprechend aus den "RXData" Registern abholen.

Die Bedeutung des Wertes ist abhängig vom Bit "RxFifoInfo" im Register "Cfo_TXRXinfoFlags" auf Seite 38.

Datentyp	Werte	Bit "RxFifoInfo"	Bedeutung
USINT	0 bis 255	0	Zähler für empfangene Telegramme
		1	Siehe Bitstruktur

Bitstruktur:

Bit	Bedeutung	Werte	Information
0 - 3	Zähler für empfangene Telegramme	0 bis 15	Nur die unteren 4 Bits
4 - 7	Anzahl der noch unquittierten Telegramme im Empfangspuffer	0 bis 15	

6.7.7 "RXCount" vom vorangegangenen Zyklus rücklesen

Name:
RXCountLatch

Dieses Register beinhaltet immer den Wert des "RXCount" aus dem vorangegangenen Zyklus. Dies kann verwendet werden um Übertragungsfehler vom Modul zur Steuerung zu erkennen (siehe "Betrachtung von Fehlerfällen beim Senden" auf Seite 43).

Die Bedeutung des Wertes ist abhängig vom Bit "RxFifoInfo" im Register "Cfo_TXRXinfoFlags" auf Seite 38.

Datentyp	Werte	Bit "RxFifoInfo"	Bedeutung
USINT	0 bis 255	0	Zähler für empfangene Telegramme aus dem vorangegangenen Zyklus
		1	Siehe Bitstruktur

Bitstruktur:

Bit	Bedeutung	Werte	Information
0 - 3	Zähler für empfangene Telegramme aus dem vorangegangenen Zyklus	0 bis 15	Nur die unteren 4 Bits
4 - 7	Anzahl der Telegramme im Empfangspuffer aus dem vorangegangenen Zyklus	0 bis 15	

6.8 Sendepuffer

6.8.1 Anzahl der CAN-Nutzdatenbytes

Name:
TXDataSize

Anzahl der zu sendenden CAN-Nutzdatenbytes. Bei einem Wert kleiner 0 wird dieses CAN-Telegramm als ungültig markiert und dadurch nicht in den Sendepuffer übernommen. Dies ist in Verbindung mit der Übertragungsfehlererkennung zwischen dem Modul und der Steuerung sinnvoll (siehe "Betrachtung von Fehlerfällen beim Senden" auf Seite 43).

Datentyp	Werte	Bedeutung
USINT	-128 bis 8	Anzahl der zu sendenden CAN-Nutzdaten

6.8.2 Identifier des CAN-Telegramms

Name:
TXIdent

Identifier des zu sendenden CAN-Telegramms. Weiters wird in diesem Register das Frameformat, sowie das Identifierformat festgelegt.

Datentyp	Werte
UDINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	Frameformat	0	Standard-Frameformat (SFF) mit 11 Bit Identifier
		1	Extended-Frameformat (EFF) mit 29 Bit Identifier
1	Frametyp	0	Datenframe
		1	Remoteframe (RTR)
2	Reserviert	-	
3 - 31	CAN-Identifier des zu sendenden Telegramms	x	Extended-Frameformat (EFF) mit 29 Bits Standard-Frameformat (SFF) mit 11 Bits[1)]

1) Nur Bits 21 bis 31 werden verwendet; Bits 3 bis 20 = 0

6.8.3 Konfiguration der zu sendenden CAN-Nutzdaten

Name:
TXDataByte0 bis TXDataByte7
TXDataWord0 bis TXDataWord3
TXDataLong0 bis TXDataLong1

CAN-Nutzdaten in Senderichtung. Je nach Bedarf können die 8 Nutzdatenbytes eines Telegramms als 8 einzelne Bytes, 4 Word oder 2 Long Datenpunkte verwendet werden.

Datentyp	Werte	Beschreibung
USINT	0 bis 255	Gesendete CAN-Nutzdaten als Byte
UINT	0 bis 65535	Gesendete CAN-Nutzdaten als Word
UDINT	0 bis 4.294.967.295	Gesendete CAN-Nutzdaten als Long

6.8.4 Betrachtung von Fehlerfällen beim Senden

Durch Übertragungsstörungen können Daten am POWERLINK Netzwerk oder X2X Link verloren gehen. Ein einmaliger Ausfall der zyklischen Daten wird durch die I/O-Systeme toleriert. Dies ist möglich, da im darauffolgenden Zyklus alle I/O-Daten neu übertragen werden. Eine Übertragungsstörung ist an den I/O-Variablen nicht ersichtlich, diese bleiben auf den Wert des letzten Zyklus eingefroren.

Durch dieses Tolerieren einmaliger I/O-Ausfälle kann es zum Verlust oder verzögertem Senden von CAN-Telegrammen kommen. Die Zählerrückmeldung wird am Modul gebildet und dient dazu diese Fälle zu erkennen.

Register für Zählerrückmeldung:

- "TXCountReadBack" auf Seite 41
- "TXCountLatchReadBack" auf Seite 42

6.9 Empfangspuffer 0 und 1

6.9.1 Anzahl der gültigen CAN-Nutzdatenbytes

Name:
RXDataSize0
RXDataSize1

Anzahl der gültigen CAN-Nutzdatenbytes.

Weiters zeigt dieses Register durch den Wert -1 (0xFF) einen generellen Fehler bzw. eine Lücke im Inputdatenstream an. Details zum aufgetretenen Fehler werden im Register "CAN-Fehlerstatus" auf Seite 40 angezeigt.

Datentyp	Werte	Bedeutung
USINT	1 bis 8	Anzahl der CAN-Nutzdaten
	-1	Fehler

6.9.2 Identifier der empfangenen Daten

Name:
RXIdent0
RXIdent1

Identifier dem die empfangenen Daten zugewiesen sind. Weiters kann aus diesem Register das Frameformat sowie das Identifierformat gelesen werden.

Datentyp	Werte
UDINT	Siehe Bitstruktur

Bitstruktur:

Bit	Beschreibung	Wert	Information
0	Frameformat	0	Standard-Frameformat (SFF) mit 11 Bit Identifier
		1	Extended-Frameformat (EFF) mit 29 Bit Identifier
1	Frametyp	0	Datenframe
		1	Remoteframe (RTR)
2	Reserviert	-	
3 - 31	CAN-Identifier des zu sendenden Telegramms	x	Extended-Frameformat (EFF) mit 29 Bits Standard-Frameformat (SFF) mit 11 Bits[1)]

1) Nur Bits 21 bis 31 werden verwendet; Bits 3 bis 20 = 0

6.9.3 Konfiguration der zu empfangenen CAN-Nutzdaten

Name:
RXData0Byte0 bis RXData0Byte7
RXData0Word0 bis RXData0Word3
RXData0Long0 bis RXData0Long1

RXData1Byte0 bis RXData1Byte7
RXData1Word0 bis RXData1Word3
RXData1Long0 bis RXData1Long1

In diese Register werden die Nutzdaten des CAN-Objekts abgelegt, die im aktuellen Zyklus vom Empfangspuffer zur Steuerung übertragen werden sollen. Wenn neue Daten empfangen werden oder sich noch weitere CAN-Objekte im Empfangspuffer befinden, werden diese Register im nächsten Zyklus mit den neuen Daten überschrieben.

Um sicherzustellen, dass möglichst keine CAN-Objekte verloren gehen, ist es erforderlich, dass die Applikation unmittelbar auf eine Änderung des "RXCount" reagiert und die Daten aus diesen Registern umkopiert.

Wahlweise können die maximal 8 Bytes eines CAN-Telegramms als 8 einzelne Bytes, als 4 Word oder als 2 Long Datenpunkte verwendet werden.

Datentyp	Werte	Beschreibung
USINT	0 bis 255	Empfangene CAN-Nutzdaten als Byte
UINT	0 bis 65535	Empfangene CAN-Nutzdaten als Word
UDINT	0 bis 4.294.967.295	Empfangene CAN-Nutzdaten als Long

6.10 Flatstream-Register

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.

> **Information:**
>
> **Für detaillierte Informationen zum Flatstream siehe "Die Flatstream-Kommunikation" auf Seite 6.**

6.10.1 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.

Datentyp	Werte
USINT	Siehe Registerübersicht

6.10.2 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 unterschiedlich.)

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".

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

Datentyp	Werte
USINT	0 bis 255

6.10.3 Kommunikationsstatus der Steuerung

Name:
OutputSequence

Dieses Register enthält Informationen über den Kommunikationsstatus der Steuerung. Es wird von der Steuerung 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 Steuerung abgeschickt wurden. Über den OutputSequenceCounter weist die Steuerung das Modul an, eine Sequenz zu übernehmen (zu diesem Zeitpunkt muss die Output-Richtung synchronisiert sein).

OutputSyncBit

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

InputSequenceAck

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

InputSyncAck

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

6.10.4 Kommunikationsstatus des Moduls

Name:
InputSequence

Dieses Register enthält Informationen über den Kommunikationsstatus des Moduls. Es wird vom Modul geschrieben und sollte von der Steuerung 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 Steuerung 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 Steuerung die Synchronität des Output-Kanals. Das Modul zeigt damit an, dass es bereit ist, Daten zu empfangen.

6.10.5 Flatstream Modus

Name:
FlatstreamMode

Mit Hilfe dieses Registers kann eine kompaktere Anordnung beim eintreffenden Datenstrom erreicht werden.

Datentyp	Werte
USINT	Siehe Bitstruktur

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		

6.10.6 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

6.10.7 Verzögerungszeit

Name:
ForwardDelay

Mit diesem Register wird die Verzögerungszeit in µs vorgegeben.

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

6.11 Azyklische Framegröße

Name:
AsynSize

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

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

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

6.12 Minimale Zykluszeit

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

Minimale Zykluszeit
200 µs

6.13 Minimale I/O-Updatezeit

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

Minimale I/O-Updatezeit
200 µs