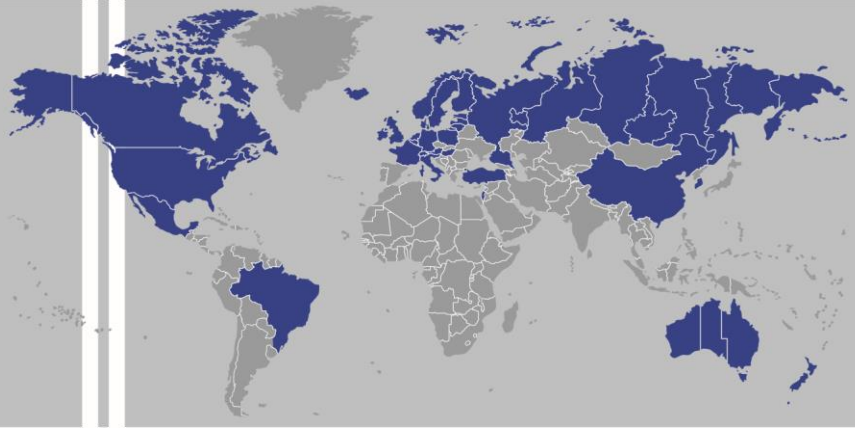

dunkermotoren
advanced motion solutions

DME 230 / 400

Profinet Schnittstelle
der Servoregler mit der Typenbezeichnung
- BN6773
- BN6783

Typ:	Part No:
DME 230x4-PN	81703.00103
DME 400x8-PN	81703.00112

Publikation Ref: 160616

Dunkermotoren GmbH | Allmendstraße 11 | D-79848 Bonndorf/ Schwarzwald
Phone +49 (0) 7703 930-0 | Fax +49 (0) 7703 930-210/ 212 | info@dunkermotoren.com

AMETEK®
PRECISION MOTION CONTROL

Servoregler Neue Generation

Digitale Servoregler
für direkten Netzanschluss

Profinet-Schnittstelle

Betriebsanleitung 6770.137, V 1.1

Diese Betriebsanleitung gilt für

- Servoregler Neue Generation, Kompaktbauweise, mit integrierter Sicherheitstechnik
  - BN 6771 bis BN 6774 mit Einbau-Netzgerät für einphasigen Anschluss an Wechselspannung
  - BN 6781 bis BN 6787 mit Einbau-Netzgerät für Drehstromanschluss
- Bedienung über Personalcomputer mit Programm SPP Windows
- Zugriff auf Gerätefunktionen über Kommunikationsschnittstellen

Diese Betriebsanleitung gilt zusammen mit

- Betriebsanleitung 6710.101 (Funktionen und Parameter)*
- Betriebsanleitung 6770.102 (Anschluss und Inbetriebnahme)
- Betriebsanleitung 6710.107 (SPP Windows Bedien- und Inbetriebnahmeprogramm)*

* In der Hilfefunktion von SPP Windows und per Download verfügbar

ESR Pollmeier GmbH
Lindenstraße 20
64372 Ober-Ramstadt
Bundesrepublik Deutschland
Tel. +49 6167 9306-0
Fax +49 6167 9306-77
E-Mail info@esr-pollmeier.de
www.esr-pollmeier.de

Versionen des Textes

2015-08-13	V 1.0, KS/Ri/MH	neu erstellt
2015-09-24	V 1.1, KS	Abschnitt PZD16 Module aktualisiert; Tabellen in Abschnitt Output Process Data und Parameter schreiben (Beispiel) aktualisiert; Extended Services bei Abschnitt Begriffe und Abkürzungen eingefügt; kleinere Korrekturen

Inhalt

Bitte beachten Sie auch das Stichwortverzeichnis am Ende des Dokuments.

1 Vorbemerkungen

1.1 Zu dieser Beschreibung

Diese Betriebsanleitung 6770.137 erläutert die Profinet-Schnittstelle der digitalen Servoregler Neue Generation.

Sie gilt zusammen mit der

- Betriebsanleitung „Anschluss und Inbetriebnahme“ des Servoreglers (gehört zum Lieferumfang)
  - Betriebsanleitung 6770.102 (Servoregler Neue Generation)
- Betriebsanleitung „Funktionen und Parameter“ des Servoreglers (in der Hilfefunktion von SPP Windows und als Download verfügbar)
  - Betriebsanleitung 6710.101

sowie je nach Ausstattung

- Betriebsanleitung „SPP Windows Bedien- und Inbetriebnahmeprogramm“ (in der Hilfefunktion von SPP Windows und als Download verfügbar)
  - Betriebsanleitung 6710.107
- Betriebsanleitung „Teileprogramm“ (Abarbeitung von Bewegungssequenzen unabhängig von einer übergeordneten Steuerung; in der Hilfefunktion von SPP Windows und als Download verfügbar)
  - Betriebsanleitung 6710.131

Bei der Arbeit mit der Profinet-Schnittstelle werden zusätzlich benötigt:

- Betriebsanleitungen der in der Steuerung eingesetzten Profinet-Anschaltbaugruppen und der dort verwendeten Software (IO-Controller) sowie
- ein zum IO-Controller kompatibles Projektierungs-Tool.

Bevor entsprechend dieser Betriebsanleitung über die Profinet-Schnittstelle auf die Servoregler zugegriffen wird, sollte der Servoantrieb (Servoregler und -motor) in Betrieb genommen sein. Für die Inbetriebnahme der Servoantriebe wird ein PC mit dem Bedien- und Inbetriebnahmeprogramm SPP Windows benötigt. Bitte prüfen Sie, ob diese Voraussetzungen erfüllt sind.

1.2 Funktionsbausteine

Für eine einfache Integration der Servoantriebe in Automatisierungssysteme sind Funktionsbausteine erhältlich.

Diese sind verfügbar für Siemens Simatic S7 und verschiedene Steuerungen nach IEC 61131-3. Bei Interesse wenden Sie sich bitte an ESR.

Die Kommunikation erfolgt über Profinet IO.

Unterstützte Funktionen:

- Parametrierung der Servoantriebe durch die Steuerung (z. B. nach Einschalten)

- Auslösen von Bewegungen (relativ/absolut positionieren, Referenzfahrt, Geschwindigkeitsvorgabe ...)
- Beeinflussung der im Antrieb integrierten Positioniersteuerung (Teileprogramm)
- Ein- und Ausgabe von Binärsignalen (Software-Ein-/Ausgänge)
- Beispielprogramme zur Benutzung der Funktionsbibliothek als Ausgangsbasis für die Entwicklung eigener Programme

Die Funktionsbausteine orientieren sich an der PLCopen-Spezifikation „Function blocks for motion control“, die wiederum auf IEC 61131-3 basiert.

Weitere Informationen finden Sie im Datenblatt 6710.160.

Die Funktionsbausteine übernehmen viele der in dieser Betriebsanleitung beschriebenen Funktionen. Welche Schritte zur Parametrierung der Profinet-Schnittstelle Sie selbst durchführen müssen, ist in der Betriebsanleitung zu den Funktionsbausteinen beschrieben.

2 Sicherheitshinweise

Beachten Sie unbedingt die Sicherheitshinweise in der Betriebsanleitung „Anschluss und Inbetriebnahme“ (6770.102) sowie die Warnungen und Hinweise in den Randspalten aller Betriebsanleitungen.

Der Zugriff auf die Servoregler über die Profinet-Schnittstelle kann Antriebsbewegungen auslösen. Wenn der Antrieb und/oder die Maschine nicht vorschriftsmäßig aufgebaut und gesichert sind, können dabei Gesundheit und Leben von Personen gefährdet werden.

Der Zugriff über die Profinet-Schnittstelle ist deshalb solange untersagt, bis die Anforderungen der Maschinenrichtlinie erfüllt sind.

Beim Einsatz von Bussystemen besteht allgemein die Gefahr einer nicht sichtbaren Beeinflussung eines Busteilnehmers von außen. Dies kann zu unerwartetem (nicht kontrolliertem) Systemverhalten führen. Nehmen Sie den Bus erst in Betrieb, nachdem Sie sich vergewissert haben, dass alle Teilnehmer vorschriftsmäßig angeschlossen und konfiguriert sind.

2.1 Art der Hinweise

Beachten Sie unbedingt die Warnungen und Hinweise am Rand:

- **Gefahr** für Gesundheit und Leben durch elektrischen Schlag oder Bewegung des Antriebs.

- **Achtung**: Nichtbeachtung verstößt gegen Sicherheitsvorschriften oder gesetzliche Vorgaben und kann zu Personen- oder Sachschäden führen.

- **Prüfen**: Überprüfen Sie vor Inbetriebnahme, bei Störungen oder auftretenden Problemen zuerst diese Punkte.

- **Tipp**, nützlicher Hinweis.

3 Technische Daten

Die Profinet-Schnittstelle ist als Baugruppe in die Servoregler Neue Generation eingebaut (Option F9). Die Steckerbelegung und die Signalpegel entsprechen der Profinet-Norm IEC 61918 bzw. IEC 61784-5. Der Busanschluss ist vom Servoregler galvanisch entkoppelt ausgeführt.

Weitere Eigenschaften der Profinet-Schnittstelle bei Servoreglern:

Software	Stack-Software in Binary- oder Source-Format
Anforderungen	100 MBit/s full duplex, switched Ethernet
Sprache für GSD-Datei	GSDML
Topologien	Stern, Linie, Baum und Ring
Funktionalität	• PROFINET Device gemäß Spezifikation V2.3, Conformance Class B • Media Redundancy Client • Multicast-Provider und -Subscriber
Anzahl PROFINET Controller für gleichzeitige Kommunikation (Shared Devices)	2
Verbindungsanzahl pro PROFINET Controller	2
Protokolle	Ethernet, IP, ARP, DCP, DHCP, RPC, RT, IRT, ESR
Maximale Größe E/A-Daten pro Kommunikationsbeziehung	1.024 Bytes
Datenaustausch	• E/A-Daten zyklisch • Alarme, Lese- und Schreibzugriff auf Records auch azyklisch
Maximale Teilnehmerzahl	unbegrenzt
Max. Konfigurationsdaten im Device	8 kByte
Max. Parameterdaten im Device	8 kByte
Max. I/O-Daten pro Kommunikationsbeziehung	1440 Byte
Unterstützung von Profilen	ja

4 Profinet-Einführung

PROFINET (Process Field Network) ist der offene Industrial Ethernet Standard von Profibus & Profinet International (PI) für die Automatisierung. Profinet nutzt TCP/IP und IT-Standards, ist Echtzeit-Ethernet-fähig und ermöglicht die Integration von Feldbus-Systemen.

Das Konzept von Profinet ist modular aufgebaut, so dass die Funktionalität vom Anwender selbst gewählt werden kann. Diese unterscheidet sich im Wesentlichen durch die Art des Datenaustauschs, um den Anforderungen an Geschwindigkeit gerecht zu werden.

Bei Profinet gibt es die beiden Varianten Profinet IO und Profinet CBA:

- Profinet IO (Input - Output) ist für die Anbindung von dezentraler Peripherie an eine Steuerung (Controller) geschaffen worden. Für die unterschiedlichen Einsatzgebiete sind die verfügbaren Funktionen und Echtzeit-Eigenschaften in die drei Konformitätsklassen CC-A, CC-B und CC-C aufgeteilt.
- Profinet CBA (Component Based Automation) ist für die komponentenbasierte Kommunikation über TCP/IP und die Real-Time-Kommunikation für Echtzeitanforderungen im modularen Anlagenbau gedacht. Beide Kommunikationswege können parallel genutzt werden.

Profinet IO und Profinet CBA können zur gleichen Zeit am selben Bussystem kommunizieren. Sie können sowohl separat betrieben als auch kombiniert werden, sodass eine Teilanlage mit Profinet IO in der Anlagensicht als eine Profinet CBA Anlage erscheint.

Für die Servoregler Neue Generation wird Profinet IO verwendet.

Ein Profinet-IO-System setzt sich aus den folgenden Geräten zusammen:

- IO-Controller (Steuerung)
- IO-Device (Feldgerät). Es besteht aus mehreren Modulen und Submodulen. Die Submodule enthalten die einzelnen Eingangs- und Ausgangssignale zum Prozess.
- IO-Supervisor ist ein Entwicklungs-Werkzeug, typischerweise basierend auf einem PC, um die einzelnen IO-Devices zu parametrieren und diagnostizieren.

Ein minimales Profinet-IO-System besteht aus mindestens einem IO-Controller, der ein oder mehrere IO-Devices kontrolliert. Zusätzlich können ein oder mehrere IO-Supervisoren bei Bedarf temporär zugeschaltet werden.

Sind zwei IO-Systeme in dem selben IP-Netzwerk vorhanden, können die IO-Controller sich auch ein Eingangssignal als shared-input teilen, indem sie auf dasselbe Submodul in einem IO-Device lesend zugreifen.

Jedes Automatisierungsgerät mit einer Ethernetschnittstelle kann gleichzeitig die Funktionalität eines IO-Controllers und eines IO-Devices erfüllen.

Für die Erhöhung der Verfügbarkeit kann mit Profinet auch eine Systemredundanz realisiert werden. In diesem Fall werden zwei IO-Controller, die dieselben IO-Devices kontrollieren, konfiguriert.

Zwischen einem IO-Controller und einem IO-Device wird eine Application-Relation (AR) aufgebaut. Über diese AR werden Communication-Relations (CR) mit unterschiedlichen Eigenschaften festgelegt:

- Record Data CR für den azyklischen Parametertransfer
- IO Data CR für den zyklischen Prozessdatenaustausch
- Alarm CR für die Signalisierung von Alarmen in Echtzeit

5 Begriffe und Abkürzungen

Dieser Abschnitt gibt einen kurzen Überblick über die wichtigsten Begriffe und Abkürzungen. Er kann die Originaldokumente und ggf. eine entsprechende Schulung nicht ersetzen.

Alarme (azyklische Alarmdaten)

Alarme sind spezielle azyklische Nachrichten, die bei Bedarf vom Peripheriegerät an den Controller übertragen werden. Diese sind zeitkritisch und werden somit wie die zyklischen Daten direkt über Ethernet übertragen. Im Gegensatz zu den zyklischen Daten müssen diese aber vom Empfänger bestätigt werden.

Azyklische Parameterdaten (Record Data CR)

Der azyklische Datenverkehr wird für Ereignisse genutzt, die sich nicht ständig wiederholen. Beispiele für azyklischen Datenverkehr sind das Senden von Parametrierungs- und Konfigurationsdaten beim Anlauf eines Peripheriegeräts an das Gerät oder das Senden einer Diagnosemeldung vom Peripheriegerät zur Zentraleinheit im laufenden Betrieb.

Azyklische Daten nutzen das UDP/IP-Protokoll.

Ausgangsdaten

Prozessdaten, die der Master an den Slave sendet.

Bus-Name

Eindeutiger Gerätename eines Kommunikations-Teilnehmers im Profinet-Netzwerk.

Eingangsdaten

Prozessdaten, mit denen der Slave dem Master antwortet.

Extended-Services

Übertragungsmethode in der Parameter-Kommunikation; ermöglichen die Übertragung von Parametern > 4 Byte Datenlänge.

Siehe „Parameterkanal“.

GSDML-Datei (Gerätestammdaten-Datei)

Elektronisches Datenblatt nach EN 50170; Text-Datei im ASCII-Format.

GSDML-Dateien erlauben eine offene (vom Hersteller unabhängige) Projektierung eines Profinet-Systems. Mit der GSDML-Datei als Eingabe für ein Projektierungs-Tool werden ein oder mehrere Antriebe mit Profinet-Schnittstelle dem Profinet-Master bekannt gemacht.

IO Data CR

Siehe Zyklische Daten.

IP-Adresse

Die IP-Adresse des Servoreglers im Profinet-Netz wird normalerweise im Profinet-Projektierungstool festgelegt und dem Servoregler beim Bushochlauf übermittelt. Alternativ kann sie im Servoregler gespeichert werden.

Master-/Slave-Anwendung

Die Master- bzw. Slave-Applikationen werden oft als Anwendungen bezeichnet. Eine Anwendung aus Sicht des Profinet ist also die Software, die auf die Profinet-Software aufsetzt.

- Master-Anwendung: z. B. SPS-Programm
- Slave-Anwendung: Firmware

Module, Modulbeschreibungen

Profinet-Schnittstellen können modular ausgeführt sein. In der GSDML-Datei werden zwischen den Schlüsselwörtern *Module* und *EndModule* eine oder mehrere Modulbeschreibungen hinterlegt (Modul-Typen definiert). Bei der Projektierung können bis zu Max_Module aus diesen Modul-Typen aktiviert werden.

In der GSDML-Datei der Servoregler sind mehrere Module definiert. Jedes Modul kann höchstens einmal aktiviert werden, muss aber nicht.

Parameter

(Allgemein) eine Klasse von Daten, die per Einzel-Zugriff übertragen werden. Die Parameter-Kommunikation verlangt eine Adressierung der Parameter über Index und Subindex.

Die Parameter der Servoregler und deren Adressen (Indizes) sind in der Betriebsanleitung 6710.101 „Funktionen und Parameter“ aufgeführt.

Parameterkanal

Der Teil der zyklischen Kommunikation, der den Zugriff auf Parameter (Objekte) des Servoreglers über Profinet ermöglicht.

Wenn der Parameterkanal aktiv ist, erhält eine Masteranwendung Zugriff auf alle Parameter der Servoregler (zusätzlich zur Prozessdatensteuerung oder statt dieser).

Für die Übertragung der Daten stehen zwei Übertragungsmethoden zur Verfügung:

- Mit den Standard-Services können Parameter bis 4 Byte Datenlänge in einem Auftrag/Antwort-Paar übertragen werden.
- Mit den Extended-Services können auch Parameter > 4 Byte Datenlänge übertragen werden. Die Daten werden dabei stückweise in einer Sequenz von Auftrag/Antwort-Paaren übertragen (Segmented-Transfer).

Prozessdaten

(Allgemein) eine Klasse von Daten, die zur Steuerung und Überwachung eines Prozesses notwendig sind.

Im Sinne von Profinet werden Prozessdaten als Nutzdaten in zyklischer Wiederholung vom Master mit einem oder mehreren Slaves ausgetauscht.

Richtungsangaben beziehen sich dabei immer auf den Master. So sendet der Master seine zyklischen Ausgangsdaten dem Slave, und der Slave antwortet darauf dem Master mit der Sendung seiner zyklischen Eingangsdaten.

Der Zusatz „zyklisch“ ist im Grunde redundant, da es sich bei Prozessdaten im Sinne von Profinet (und damit bei den Eingangs- und Ausgangsdaten) immer um zyklisch ausgetauschte Daten handelt.

Prozessdatenkanal

Der Teil der zyklischen Kommunikation, der den kontinuierlich zyklischen Zugriff auf die Prozessdaten des Servoreglers über Profinet ermöglicht.

Record Data CR

Siehe Azyklische Parameterdaten.

Segmented-Transfer

Methode der Extended-Services in der Parameter-Kommunikation; Parameter größerer Datenlänge werden stückweise in einer Sequenz von Auftrag/Antwort-Paaren übertragen.

Siehe Parameterkanal.

Standard-Services

Übertragungsmethode in der Parameter-Kommunikation; ermöglichen die Übertragung von Parametern bis zu 4 Byte Datenlänge.

Siehe Parameterkanal.

Zyklische Daten (IO Data CR)

Der Inhalt des zyklischen Datenverkehrs sind die Daten, die die Zentraleinheit an die Peripheriegeräte schickt, sowie die Daten, die ein Peripheriegerät an seinen Eingägen einliest und zur Verarbeitung an die Zentraleinheit schickt. In der Regel geht also in jedem Zyklus ein solches zyklisches Datenpaket von der Zentraleinheit an das Peripheriegerät und umgekehrt.

6 Anschluss und Inbetriebnahme

Für den Anschluss und zur Statusanzeige sind die Servoregler mit folgenden Elementen ausgestattet:

- Busanschluss
- Leuchtdioden
- Drehcodierschalter

Diese Elemente befinden sich auf der Frontplatte der Servoregler mit Profinet-Schnittstelle.

6.1 Busanschluss

X4.1/F7 Profinet-In: RJ45-Stecker

X4.2/F7 Profinet-Out: RJ45-Stecker

Die Profinet-Schnittstelle besteht aus einem Profinet-Switch mit 2 Anschlüssen. Die Steckerbelegung und die Signalpegel entsprechen dem Ethernet-Standard IEEE 802.3 sowie den Normen IEC 61918 bzw. 61784-5.

Auf der Frontplatte der Servoregler mit Profinet-Schnittstelle befinden sich die Stecker X 4.1 und X 4.2., die beide für den Anschluss verwendet werden können. Für den Anschluss an das Profinet-Netz wird nur einer dieser Ports benötigt, der zweite Port ist für den optionalen Anschluss weiterer Geräte vorgesehen.

Durch die Verkabelung mit Standard-Ethernet entfällt eine Terminierung des ersten bzw. letzten Busteilnehmers.

6.2 Drehcodierschalter

Der Drehcodierschalter hat 16 verschiedene Stellungen (0-F). Die ersten 15 Stellungen (0-E) stellen vordefinierte Gerätenamen ein, in der Stellung F muss der Name über das Projektierungstool vergeben werden bzw. es wird der dort vergebene und gespeicherte Name benutzt.

Schalterstellung	Gerätename
0	ac-servo-ng
1	ac-servo-ng-1
2	ac-servo-ng-2
...	
D	ac-servo-ng-13
E	ac-servo-ng-14
F	Projektierungstool

6.3 Leuchtdioden

Über die Bus-LEDs des Servoreglers werden interne Zustände des Profinet-Stacks angezeigt. Die internen Zustände werden teilweise über verschiedene Blink-Codes ausgegeben.

LED Error (rot)

Status	Blink-Sequenz	Beschreibung
NO-ERROR	aus	kein Fehler
DOUBLE_ADDRESS_ERROR	blinkt 2×, dann 1 s aus	Adressenkonflikt entdeckt. Die Netzwerkkommunikation ist möglicherweise gestört.
HARDWARE_ERROR	blinkt 3×, dann 1 s aus	Mögliche Gründe sind fehlerhafter Zugriff auf das MAC oder unterbrochene Datenströme.
APPL_WATCHDOG_EXPIRED	blinkt 1×kurz, dann 1 s aus	Anwendungs-Watchdog abgelaufen.
PROTOCOL_ERROR	an	Ein protokollspezifischer Fehler ist aufgetreten.

LED Run (grün)

Status	Blink-Sequenz	Beschreibung
BEFORE_INIT	aus	Stack nicht initialisiert
INIT	blinkt 1×, dann 1 s aus	Stack wird initialisiert
ONLINE	blinkt	Stack ist online (auf das Gerät kann über die Netzwerkschnittstelle zugegriffen werden)
CONNECTED	an	Stack ist online (auf das Gerät kann über die Netzwerkschnittstelle zugegriffen werden)

LED Aux1 (gelb)

Status	Blink-Sequenz	Beschreibung
NO_PAR	aus	Parameterkanal (PZD8) nicht projektiert oder projektiert aber noch nicht benutzt.
PAR_ACCESS	blinkt	Parameterkanal (PZD8) aktiv, Kommunikation findet statt
PAR_USED	an	Parameterkanal (PZD8) aktiv, es findet jedoch momentan keine Kommunikation statt.

LED Aux2 (gelb)

Die LED Aux2 ist für die Blinkfunktion reserviert und wird daher nur bei entsprechender Funktionsauslösung durch das Profinet-Projektierungstool aktiv.

7 Module aktivieren und deaktivieren

Bei Profinet sind wie bei Profibus auch Module für die Prozessdatenkommunikation vordefiniert. Module werden dabei auf so genannte Modulsteckplätze gesteckt. Der Servoregler hat derzeit 64 Modulsteckplätze, von denen nur die ersten 6 genutzt werden können, da in der GSDML nur 6 unterschiedliche Module definiert sind und diese festen Modulsteckplätzen zugeordnet sind. In der GSDML-Datei sind derzeit folgende Module definiert:

Modul	Steckplatz	Funktion
PAR8 IN/OUT	1	Parameterkanal
PZD16 IN/OUT	2	Prozessdatenkanal
PAR8 IN	3	Alternativer Parameterkanal IN
PAR8 OUT	4	Alternativer Parameterkanal OUT
PZD16 IN	5	Testkanal IN
PZD16 OUT	6	Testkanal OUT

PZD = Prozessdatenkanal; PAR = Parameterkanal

Standardmäßig werden nur die beiden ersten Module belegt. Das PAR8 IN/OUT Modul entspricht in Belegung und Funktion dem PAR Modul des Profibus (Parameterkanal), das PZD16 IN/OUT dem PZD-Modul des Profibus (Prozessdatenkanal). Es existiert kein kombiniertes 24 Byte PAR+PZD Modul. Dies ist nicht notwendig, da beim Profinet die Module unabhängig voneinander genutzt werden können. Über die Module PAR8 IN und PAR8 OUT kann ein zweiter Parameterkanal betrieben werden, die PZD16 IN und PZD16 OUT Module sind nur für interne Testzwecke gedacht und sollten nicht verwendet werden.

7.1 PAR8 Modul

Das PAR8-Modul entspricht in Funktion und Belegung exakt dem PAR-Modul des Profibus, es werden alle dort definierten Services unterstützt. Das PAR-Modul ist als Octet-String 8 definiert, also als 8 Byte Array ohne weitere Funktionskennung der einzelnen Bytes.

7.2 PZD16 Modul

Das PZD16-Modul entspricht in Funktion und Belegung dem PZD-Modul des Profibus. In der GSDML-Datei ist der PZD16-Kanal als Struktur mit seiner tatsächlichen Belegung (Steuerwort, Lageziel, Statuswort, Lage-Istwert ...) definiert. Je nach Projektierungstool werden diese Informationen nicht genutzt, das PZD16 wird wie beim Profibus als 16 Byte Array behandelt. Andere Tools interpretieren diese Information korrekt und stellen die einzelnen Variablen des Prozessdatenkanals zur Verfügung.

7.3 Modulparameter

Den Standard-Modulen PAR IN/OUT und PZD IN/OUT sind weitere Parameter zugeordnet, die über das Projektierungstool zugänglich sind. Diese sind

Parameter	Default	Funktion
Diagnose-Alarm aktivieren	Aus	Aktiviert die Meldung von Alarmen über dieses Modul
Verbindungs-Überwachung aktivieren	Aus	Aktiviert die Verbindungs-Überwachung über dieses Modul

7.4 Verbindungsüberwachung

Wird die Verbindungsüberwachung bei mindestens einem Modul aktiviert, wird bei einem Verbindungsverlust nach dem Hochlauf des Busses eine Störung im Servoregler ausgelöst. Diese Funktion entspricht daher in etwa der Master-Watchdog-Überwachung im Profibus.

8 Prozessdaten-Kommunikation

Die Prozessdaten-Kommunikation ist bei der Profinet-Schnittstelle der Servoregler über einen so genannten Prozessdatenkanal realisiert.

Der Prozessdatenkanal umfasst 16 Bytes je Richtung (Ausgangs- und Eingangs-Prozessdaten). Auf den Prozessdatenkanal sind Variablen der digitalen Servoregler abgebildet; die Abbildung der Variablen auf die einzelnen Bytes des Prozessdatenkanals ist in den folgenden Tabellen jeweils getrennt für die Ausgangs- und Eingangs-Prozessdaten beschrieben.

Wenn der Prozessdatenkanal aktiv ist, erhöht sich die Länge der zyklischen Daten um 16 Byte. Der Prozessdatenkanal belegt die letzten 16 Byte im zyklischen Kanal.

Andere Längen des Prozessdatenkanals und die Abbildung anderer Variablen sind auf Anfrage möglich; bitte wenden Sie sich bei Interesse an ESR.

Die Funktion der Variablen ist in der Betriebsanleitung 6710.101 „Funktionen und Parameter“ beschrieben. Beachten Sie dort speziell den Teil Variablen-Beschreibungen.

(?) Damit die Prozessdaten-Kommunikation im zyklischen Kanal stattfinden kann, müssen den einzelnen Eingangs- und Ausgangsdaten der zyklischen Kommunikation die korrekten Nutzdaten-Adressen im Master zugeordnet werden.

8.1 Ausgangs-Prozessdaten

Die Ausgangs-Prozessdaten werden vom Master (z. B. SPS, PC) an den Servoregler übertragen.

Ausgangs-Prozessdaten Profinet		
Prozessdaten		Variable
Byte	Bit	
1		Achsen-Steuerwort (Bit 0 .. 7)
2		Achsen-Steuerwort (Bit 8 .. 15)
3		Verfahrgeschwindigkeit (Bit 31 .. 24)
4		Verfahrgeschwindigkeit (Bit 23 .. 16)
5		Verfahrgeschwindigkeit (Bit 15 .. 8)
6		Verfahrgeschwindigkeit (Bit 7 .. 0)
7		Lageziel (Bit 31 .. 24)
8		Lageziel (Bit 23 .. 16)
9		Lageziel (Bit 15 .. 8)
10		Lageziel (Bit 7 .. 0)
11	7 .. 4	Digitale Ausgänge A 2.3 .. 2.0 (Software-Ausgänge)
	3 .. 0	Digitale Ausgänge A 1.3 .. 1.0 (Schalt-Ausgänge an X7)
12		Digitale Eingänge E 9.7 .. 9.0 (Software-Eingänge)
13		Momentensollwert (Bit 15 .. 8)
14		Momentensollwert (Bit 7 .. 0)

Ausgangs-Prozessdaten Profinet		
Prozessdaten		Variable
Byte	Bit	
15		Achsen-Betriebsart (Bit 15 .. 8)
16		Achsen-Betriebsart (Bit 7 .. 0)

8.2 Eingangs-Prozessdaten

Die Eingangs-Prozessdaten werden vom Servoregler zum Master (z. B. SPS, PC) übertragen.

Eingangs-Prozessdaten Profinet		
Prozessdaten		Variable
Byte	Bit	
1		Achsen-Statuswort (Bit 0 .. 7)
2		Achsen-Statuswort (Bit 8 .. 15)
3		Geschwindigkeits-Istwert (Bit 31 .. 24)
4		Geschwindigkeits-Istwert (Bit 23 .. 16)
5		Geschwindigkeits-Istwert (Bit 15 .. 8)
6		Geschwindigkeits-Istwert (Bit 7 .. 0)
7		Lage-Istwert (Bit 31 .. 24)
8		Lage-Istwert (Bit 23 .. 16)
9		Lage-Istwert (Bit 15 .. 8)
10		Lage-Istwert (Bit 7 .. 0)
11		Digitale Eingänge E 1.7 .. 1.0 (Schalt-Eingänge an X7)
12		Digitale Eingänge E 3.7 .. 3.0
13		Strom-Istwert (Bit 15 .. 8)
14		Strom-Istwert (Bit 7 .. 0)
15		Digitale Ausgänge A 8.7 .. 8.0 (Software-Ausgänge)
16		Digitale Ausgänge A 9.7 .. 9.0 (Software-Ausgänge)

9 Parameter-Kommunikation

Die Parameter-Kommunikation ist bei der Profinet-Schnittstelle über einen Parameterkanal realisiert. Der Parameterkanal erlaubt den Zugriff von einem Profinet-Master auf alle Parameter (Variablen) des Servoreglers. Die Variablen können über den Parameterkanal gelesen und geschrieben werden, sie werden dabei über Index und Subindex adressiert. Nähere Informationen zu den Variablen enthält die Betriebsanleitung 6710.101 „Funktionen und Parameter“.

Die Parameter werden im zyklischen Kanal zusätzlich zu den klassischen Prozessdaten übertragen. Der Parameterkanal kann von der Master-Seite aktiviert oder deaktiviert werden.

Wenn der Parameterkanal aktiv ist, erhöht sich die Länge der zyklischen Daten um 8 Byte. Der Parameterkanal belegt die ersten 8 Byte im zyklischen Kanal.

9.1 Standard- und Extended-Services

Mit den Standard-Services kann auf alle Parameter (Variablen) des Servoreglers zugegriffen werden, deren Länge bis zu 4 Byte beträgt. Über die Extended-Services können auch Parameter mit einer Länge von mehr als 4 Byte übertragen werden. Das betrifft nur einige wenige Parameter, auf die in den meisten Anwendungen nicht vom Master zugegriffen wird:

Parameter mit Länge > 4 Byte			
Index	**Name**	**Datentyp**	**Bemerkung**
5ee6	Motorbeschreibung	VisStr 16	
5f43	Tracebuffer 1	Array 16 von OctStr 128	Puffer für Datenaufzeichnung, werden von Oszilloskop-Funktionen des Bedien- und Inbetriebnahmeprogramms SPP Windows genutzt
5f45	Tracebuffer 2		
5f47	Tracebuffer 3		
5f5f	Teileprogramm	Array 71 von OctStr 128	Wird mit Teileprogramm-Editor von SPP Windows erstellt
5f81	Störungs-Detail	Array 4 von VisStr 16	Zusatzinformation für einige wenige Achsen-Störungscodes

Die Übertragungsmethode „Standard-Services“ oder „Extended-Services“ wird über das Service-Byte gewählt.

9.2 Standard-Services

Mit den Standard-Services kann auf nahezu alle Parameter (Variablen) des Servoreglers zugegriffen werden.

Benötigt die Masteranwendung keinen Zugriff auf die wenigen Parameter, die länger als 4 Byte sind, müssen dort nur die Standard-Services implementiert werden. Das verringert den Implementierungsaufwand auf Masterseite erheblich.

9.2.1 Aufbau des Parameterkanals

Der Parameterkanal ist für beide Übertragungsrichtungen identisch aufgebaut, als Auftrag in den Ausgangsdaten vom Master zum Slave und als Antwort in den Eingangsdaten vom Slave zum Master.

Der Master darf einen zweiten Auftrag nicht absenden, bevor nicht die Antwort des ersten Auftrags bei ihm eingegangen ist.

Wenn der Parameterkanal aktiv ist, werden in den Standard-Services die folgenden 8 Bytes im zyklischen Kanal vor den eigentlichen Prozessdaten übertragen:

Parameterkanal, Standard-Services	
Byte	**Inhalt**
1	Service-Byte
2	Subindex
3	Index (Bit 15 .. 8)
4	Index (Bit 7 .. 0)
5	Data/Error 1 (höchstwertiges Byte)
6	Data/Error 2
7	Data/Error 3
8	Data/Error 4 (niederstwertiges Byte)

Über das Service-Byte (Byte 1) findet die Auftrags- und Antwortsteuerung des Parameterkanals statt. Mit ihm werden auch die Standard-Services als Übertragungsmethode gewählt. Der Aufbau des Service-Bytes ist im folgenden Abschnitt ausführlich beschrieben.

Bytes 2 bis 4 adressieren den Parameter. Der Auftrag enthält Index und Subindex zur Adressierung des Parameters. In der Antwort werden der Index und Subindex zurückgeliefert, über die die Adressierung bei Zugriff tatsächlich erfolgte.

Bytes 5 bis 8 enthalten Daten oder einen Fehlercode.

9.2.2 Aufbau des Service-Bytes

Das Service-Byte (Byte 1 im Parameterkanal) ist das zeitlich zuerst übertragene Byte. Es hat in der Regel die niedrigste relative Adresse im zyklischen Kanal (Adresse 0).

Die einzelnen Bits haben folgende Bedeutung:

Service-Byte (Standard-Services)	
Bit	**Inhalt**
0 .. 2	Service-Codierung: • 000_{bin} = 0: kein Auftrag, Standard-Services • 001_{bin} = 1: Lese-Auftrag, Standard-Services (Daten vom Servoregler lesen) • 010_{bin} = 2: Schreib-Auftrag, Standard-Services (Daten zum Servoregler schreiben) • 100_{bin} = 4: Extended-Services (Lesen oder Schreiben)
3	reserviert (redundant, sollte jedoch immer mit 0 belegt werden)
4 .. 5	Standard-Services: Länge der Daten (Anzahl der gültigen Bytes) in den Feldern Data/Error • 00_{bin} = 0: 1 Byte • 01_{bin} = 1: 2 Byte • 10_{bin} = 2: 3 Byte • 11_{bin} = 3: 4 Byte Extended-Services: Kennung für Lese- oder Schreib-Auftrag
6	Handshake (Kennung, dass ein neuer Auftrag anliegt): Dieses Bit wird vom Master bei jedem neuen Auftrag gewechselt. Der Servoregler kopiert das Bit in sein Antwort-Telegramm.
7	Status (Fehlerinformation vom Servoregler): Mit diesem Bit teilt der Servoregler in der Antwort an den Master mit, ob der Auftrag fehlerfrei ausgeführt wurde. • 0 = Auftrag ohne Fehler ausgeführt • 1 = Auftrag nicht ausgeführt, ein Fehler ist aufgetreten. Die Daten im Feld Data/Error sind als Fehlermeldung zu interpretieren.

In den Bits 0 bis 2 ist der gewünschte Service codiert. Da ein Slave keinen Auftrag senden darf, sind die Werte 001_{bin} (Lese-Auftrag) und 010_{bin} (Schreib-Auftrag) dem Master vorbehalten. ESR-Servoregler antworten mit 000_{bin} (kein Auftrag).

Bei Empfang einer ungültigen Service-Codierung antworten ESR-Servoregler mit einem Fehler-Status-Telegramm (Fehlercode $06\ 05\ 00\ 10_{hex}$, „Unzulässiger Auftragsparameter“).

Die Bits 4 und 5 geben in den Standard-Services die Länge der Daten (Anzahl der gültigen Bytes) in den Feldern Data/Error 1 bis 4 an:

- Beim Schreib-Auftrag des Masters und bei der Lese-Antwort des Slaves muss hier angegeben werden, wie viele Bytes in den Feldern Data/Error 1 bis 4 übertragen werden (00_{bin} für 1 Byte, 01_{bin} für 2 Byte usw.). Der Empfänger prüft diese Angabe.
- Beim Lese-Auftrag des Masters und bei der Schreib-Antwort des Slaves ist diese Angabe redundant und kann entfallen. Der Empfänger muss die Angabe nicht prüfen. ESR-Servoregler senden in der Schreib-Antwort immer 11_{bin}, ebenso wie in der Fehler-Status-Antwort, da ausschließlich Fehlercodes der Länge 4 Byte definiert sind.

Mit Bit 6 („Handshake“) wird der Parameterkanal gesteuert. Die Telegramme von Auftrag und Antwort werden im zyklischen Kanal solange übertragen, bis

sie von den Teilnehmern wieder überschrieben werden. Dabei werden zwei Zustände am Bus unterschieden:

- Ungleiche Handshake-Bits im Auftrag und in der Antwort:
  - Der Master hat einen neuen Auftrag gesendet und wartet auf Antwort. Das Auftragstelegramm des Masters ist gültig.

    Der Master überprüft die Handshake-Bits in den Antworten vom Slave und führt ansonsten keine weiteren Aktionen durch. Insbesondere darf er keinen weiteren Auftrag senden, bevor er nicht eine Antwort vom Slave auf den aktuellen Auftrag empfangen hat (mit gleichem Handshake-Bit).
  - Der Slave erkennt den neuen Auftrag und verarbeitet ihn. Das alte Antworttelegramm des Slaves ist ungültig.

    Wenn der Slave den neuen Auftrag empfangen hat, überprüft er die Handshake-Bits in den Aufträgen vom Master nicht weiter. In der Antwort zum neuen Auftrag übernimmt der Slave das Handshake-Bit (damit sind die Handshake-Bits in Auftrag und Antwort gleich).
- Gleiche Handshake-Bits im Auftrag und in der Antwort:
  - Der Slave hat eine Antwort gesendet und wartet auf einen neuen Auftrag. Das Antworttelegramm des Slaves ist gültig.

    Der Slave überprüft die Handshake-Bits in den Aufträgen vom Master und führt ansonsten keine weiteren Aktionen durch. Insbesondere wird er keine weitere Antwort senden, bevor er nicht einen neuen Auftrag vom Master empfangen hat (mit ungleichem Handshake-Bit).
  - Der Master erkennt die Antwort vom Slave und verarbeitet sie. Das alte Auftragstelegramm des Masters ist ungültig.

    Wenn der Master die Antwort empfangen hat, überprüft er die Handshake-Bits in den Antworten vom Slave nicht weiter. Wenn er einen neuen Auftrag erstellt, wechselt der Master das Handshake-Bit (damit sind die Handshake-Bits in Auftrag und Antwort verschieden).

Auf der Seite des Masters hat sich folgende Vorgehensweise bewährt:

- Unmittelbar nach der Initialisierung wird das Handshake-Bit auf 0 gesetzt.
- Der Master liest das Handshake-Bit des letzten (noch auf dem Bus anstehenden) Auftragstelegramms, negiert es und überträgt es in sein neues Auftragstelegramm.
- Das Service-Byte (Byte 1) mit dem geänderten Handshake-Bit wird als letztes auf den Bus durchgestellt. Damit wird der neue Auftrag gültig.

Bit 7 („Status“) bestimmt den Inhalt der Felder Data/Error in der Antwort vom Slave:

- Status = 0: Data

  In den Feldern Data/Error wird ein Parameterwert übertragen, der je nach Datenformat 1 bis 4 Byte belegt.
- Status = 1: Error

  In den Feldern Data/Error stehen Angaben zur Beschreibung eines Fehlers.

Im Auftrag des Masters muss dieses Bit auf 0 gesetzt werden.

9.2.3 Übertragungsformat der Daten

In den Feldern Data/Error können Parameterwerte bis 4 Byte Länge übertragen werden. Wenn die Länge des Parameters größer als 1 Byte ist, erfolgt die Übertragung im Motorola-Format (höchstwertiges Byte/Wort zuerst); Parameter, die kürzer als 4 Byte sind, werden in den ersten Data-Feldern übertragen:

Parameterkanal		**Übertragung von ...**			
Byte	**Data**	**Byte**	**Wort**	**3 Bytes ***	**Doppelwort**
5	Data 1	Bit 7 .. 0	Bit 15 .. 8	Byte 1	Bit 31 .. 24
6	Data 2	–	Bit 7 .. 0	Byte 2	Bit 23 .. 16
7	Data 3	–	–	Byte 3	Bit 15 .. 8
8	Data 4	–	–	–	Bit 7 .. 0

* nur Zeichenketten der Länge 3

9.2.4 Beispiele: Parameter-Kommunikation mit dem Servoantrieb

Damit die Parameter-Kommunikation im zyklischen Kanal stattfinden kann, müssen den Eingangs- und Ausgangsdaten der zyklischen Kommunikation die korrekten Nutzdaten-Adressen im Profinet-Master zugeordnet werden.

Um auszuschließen, dass ein unvollständiger Auftrag an den Servoregler gesendet und von ihm verarbeitet wird, empfiehlt es sich, den Auftrag „von hinten“ aufzubauen. Dadurch wird das erste (Service-)Byte zuletzt gesetzt, das mit dem Bit „Handshake“ dem Servoregler einen neuen Auftrag signalisiert.

Der Master sollte nur eine begrenzte Zeit auf die Antwort des Slaves warten. Es ist sinnvoll, hier eine Zeitüberwachung zu implementieren.

Ein Lese- oder Schreibauftrag des Masters wird wie folgt erstellt (in dieser Reihenfolge):

- Feld „Data/Error“:
  - bei Leseauftrag 0 eintragen.
  - bei Schreibauftrag Parameterwert eintragen.
- Adresse des gewünschten Parameters in die Felder „Index“ und „Subindex“ eintragen.
- Service-Byte:
  - Bits „Service-Codierung“ entsprechend der gewünschten Auftragsart (Lesen oder Schreiben) setzen.
  - Länge des Parameters in den Bits „Länge“ eintragen.
  - Bit „Status“ auf 0 setzen („kein Fehler“).
  - Bit „Handshake“ wechseln.
- Zeitüberwachung starten.

Die Antwort des Servoreglers wird folgendermaßen ausgewertet:

- Prüfen, ob das Bit „Handshake“ bei den Eingangsdaten und Ausgangsdaten identisch ist.

  Ist das Bit identisch, wurde die Antwort empfangen und die Zeitüberwachung kann gestoppt werden. Läuft die Zeitüberwachung ab, ohne dass eine neue Antwort empfangen wurde, sollte der Master eine entsprechende Fehlermeldung ausgeben.

- Prüfen, ob das Bit „Status“ gesetzt ist:
  - Ist das Bit nicht gesetzt, wurde der Auftrag erfolgreich ausgeführt. Das Feld „Data/Error“ enthält den gewünschten Parameterwert (Lese-Antwort) bzw. ist leer (Schreib-Antwort).
  - Ist das Bit gesetzt, wurde der Auftrag nicht ausgeführt. Ein Fehler ist aufgetreten. Im Feld „Data/Error“ befindet sich die Fehlerinformation.

9.2.4.1 Parameter schreiben (Beispiel)

Der Strom-Max-Betrag (*IImpuls*, index 6073 $_{hex}$, Datentyp Unsigned16 = Wort) soll auf 200% gesetzt werden (entspricht einem Zahlenwert von 2000).

- Telegramm vom Master an den Servoregler:

  5200 6073 07D0 0000 $_{hex}$ (falls Bit „Handshake“ vorher 0 war)
  1200 6073 07D0 0000 $_{hex}$ (falls Bit „Handshake“ vorher 1 war)

Byte	Wert	Inhalt	
1	52 $_{hex}$ oder 12 $_{hex}$	Bit 0 .. 2:	xxxx x010 = Schreibauftrag
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx01 xxxx = 2 Byte Daten im Feld Data/Error
		Bit 6:	x1xx xxxx – falls Handshake vorher 0 war
			x0xx xxxx – falls Handshake vorher 1 war
		Bit 7:	0xxx xxxx = kein Fehler
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	60 $_{hex}$	Index = 6073 $_{hex}$	
4	73 $_{hex}$		
5	07 $_{hex}$	Data = 07D0 $_{hex}$ (= 2000)	
6	D0 $_{hex}$		
7	00 $_{hex}$		
8	00 $_{hex}$		

- Antwort des Antriebs bei fehlerfreier Ausführung:

  5000 6073 0000 0000 $_{hex}$ (falls Bit „Handshake“ 0 war)
  1000 6073 0000 0000 $_{hex}$ (falls Bit „Handshake“ 1 war)

Byte	Wert	Inhalt	
1	50_{hex} oder 10_{hex}	Bit 0 .. 2:	xxxx x000 = kein Auftrag
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx01 xxxx = 2 Byte Daten im Feld Data/Error
		Bit 6:	x1xx xxxx = Handshake 1 zurücksenden
			x0xx xxxx = Handshake 0 zurücksenden
		Bit 7:	0xxx xxxx = Auftrag ausgeführt
2	00_{hex}	Subindex = 00_{hex}	
3	60_{hex}	Index = 6073_{hex}	
4	73_{hex}		
5 .. 8	00_{hex}	Data = 0000 0000 $_{hex}$ (ohne Bedeutung)	

9.2.4.2 Parameter lesen (Beispiel)

Die Zwischenkreisspannung (*UZwKreis*, Index 6079 $_{hex}$, Datentyp Unsigned16 = Wort) soll ausgelesen werden.

- Telegramm vom Master an den Servoregler:
  5100 6079 0000 0000 $_{hex}$ (falls Bit „Handshake“ vorher 0 war)
  1100 6079 0000 0000 $_{hex}$ (falls Bit „Handshake“ vorher 1 war)

Byte	Wert	Inhalt	
1	51_{hex} oder 11_{hex}	Bit 0 .. 2:	xxxx x001 = Leseauftrag
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx01 xxxx = 2 Byte Daten im Feld Data/Error
		Bit 6:	x1xx xxxx – falls Handshake vorher 0 war
			x0xx xxxx – falls Handshake vorher 1 war
		Bit 7:	0xxx xxxx = kein Fehler
2	00_{hex}	Subindex = 00_{hex}	
3	60_{hex}	Index = 6079_{hex}	
4	79_{hex}		
5 .. 8	00_{hex}	Data = 0000 0000 $_{hex}$ (ohne Bedeutung)	

- Antwort des Antriebs bei fehlerfreier Ausführung:
  5000 6079 0140 0000 $_{hex}$ (falls Bit „Handshake“ 1 war)
  1000 6079 0140 0000 $_{hex}$ (falls Bit „Handshake“ 0 war)

Byte	Wert	Inhalt	
1	50 $_{hex}$ oder 10 $_{hex}$	Bit 0 .. 2:	xxxx x000 = kein Auftrag
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx01 xxxx = 2 Byte Daten im Feld Data/Error
		Bit 6:	x1xx xxxx = Handshake 1 zurücksenden
			x0xx xxxx = Handshake 0 zurücksenden
		Bit 7:	0xxx xxxx = Auftrag ausgeführt
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	60 $_{hex}$	Index = 6079 $_{hex}$	
4	79 $_{hex}$		
5	01 $_{hex}$	Data = 0140 $_{hex}$ (= 320 V)	
6	40 $_{hex}$		
7	00 $_{hex}$	Data = 0000 $_{hex}$ (ohne Bedeutung)	
8	00 $_{hex}$		

9.2.4.3 Antwort mit Fehler-Status (Beispiel)

Die Motorbeschreibung (*MotorName*, Index 5EE6 $_{hex}$, Datentyp VisibleString16 = Zeichenkette mit 16 Byte Länge) soll über einen Standard-Lese-Auftrag (Länge bis 4 Byte) versucht werden zu lesen.

- Telegramm vom Master an den Servoregler:

  7100 5EE6 0000 0000 $_{hex}$ (falls Bit „Handshake“ vorher 0 war)
  3100 5EE6 0000 0000 $_{hex}$ (falls Bit „Handshake“ vorher 1 war)

Byte	Wert	Inhalt	
1	71 $_{hex}$ oder 31 $_{hex}$	Bit 0 .. 2:	xxxx x001 = Leseauftrag
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx11 xxxx = 4 Byte Daten im Feld Data/Error
		Bit 6:	x1xx xxxx – falls Handshake vorher 0 war
			x0xx xxxx – falls Handshake vorher 1 war
		Bit 7:	0xxx xxxx = kein Fehler
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	5E $_{hex}$	Index = 5EE6 $_{hex}$	
4	E6 $_{hex}$		
5 .. 8	00 $_{hex}$	Data = 0000 0000 $_{hex}$ (ohne Bedeutung)	

- Antwort mit Fehler-Status des Antriebs:

  F000 5EE6 0605 0012 $_{hex}$ (falls Bit „Handshake“ 1 war)
  B000 5EE6 0605 0012 $_{hex}$ (falls Bit „Handshake“ 0 war)

Byte	Wert	Inhalt	
1	$F0_{hex}$ oder $B0_{hex}$	Bit 0 .. 2:	xxxx x000 = kein Auftrag
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx11 xxxx = 4 Byte Daten im Feld Data/Error
		Bit 6:	x1xx xxxx = Handshake 1 zurücksenden
			x0xx xxxx = Handshake 0 zurücksenden
		Bit 7:	1xxx xxxx = Auftrag nicht ausgeführt, Fehler
2	00_{hex}	Subindex = 00_{hex}	
3	$5E_{hex}$	Index = $5EE6_{hex}$	
4	$E6_{hex}$		
5	06_{hex}	Fehlercode = 06 05 00 12 $_{hex}$ („Datenlänge zu groß oder zu klein“)	
6	05_{hex}		
7	00_{hex}		
8	12_{hex}		

9.3 Extended-Services

Die Extended-Services werden eingesetzt, wenn Parameter übertragen werden sollen, die länger als 4 Byte sind. Es ist zulässig, Parameter mit einer Länge kleiner oder gleich 4 Byte über die Extended-Services zu übertragen (dafür werden allerdings zwei Auftrag/Antwort-Paare benötigt).

(?) Benötigt die Masteranwendung keinen Zugriff auf die wenigen Parameter, die länger als 4 Byte sind, müssen dort nur die Standard-Services implementiert werden. Der Implementierungsaufwand für die Extended-Services kann in diesem Fall auf der Masteranwendungsseite eingespart werden.

9.3.1 Aufbau des Parameterkanals

Der Parameterkanal ist für beide Übertragungsrichtungen identisch aufgebaut, als Auftrag in den Ausgangsdaten vom Master zum Slave und als Antwort in den Eingangsdaten vom Slave zum Master.

Der Master darf einen zweiten Auftrag nicht absenden, bevor nicht die Antwort des ersten Auftrags bei ihm eingegangen ist.

Wenn der Parameterkanal aktiv ist, werden in den Extended-Services die folgenden 8 Bytes im zyklischen Kanal vor den eigentlichen Prozessdaten übertragen:

Parameterkanal, Extended-Services		
Byte	**Inhalt 1. Telegramm (Initiate) oder Fehler-Status-Telegramm**	**Inhalt weitere Telegramme (Read Segment oder Write Segment)**
1	Service-Byte	Service-Byte
2	Subindex	Steuerungs-Byte
3	Index (Bit 15 .. 8)	Data 1
4	Index (Bit 7 .. 0)	Data 2
5	Data/Error 1 (höchstwertiges Byte)	Data 3
6	Data/Error 2	Data 4
7	Data/Error 3	Data 5
8	Data/Error 4 (niederstwertiges Byte)	Data 6

Der Aufbau des ersten Telegramms oder des Fehler-Status-Telegramms ist identisch mit dem Aufbau in den Standard-Services. In den folgenden Telegrammen werden weitere Bytes zur Datenübertragung genutzt, mit dem Steuerungs-Byte wird die sequenzielle Übertragung gesteuert.

Über das Service-Byte (Byte 1) findet die Auftrags- und Antwortsteuerung des Parameterkanals statt. Mit ihm werden auch die Extended-Services als Übertragungsmethode gewählt.

Das Steuerungs-Byte gibt die Anzahl der gültigen Datenbytes in den Feldern Data 1 bis Data 6 an. Es enthält ein Statusbit, das anzeigt, ob weitere Segmente folgen oder die Übertragung des Parameters beendet ist.

9.3.2 Aufbau des Service-Bytes

Das Service-Byte (Byte 1 im Parameterkanal) ist das zeitlich zuerst übertragene Byte. Es hat in der Regel die niedrigste relative Adresse im zyklischen Kanal (Adresse 0).

Die einzelnen Bits haben folgende Bedeutung:

Service-Byte (Extended-Services)	
Bit	**Inhalt**
0 .. 2	Service-Codierung: • 000_{bin} = 0: Abbruch der Datenübertragung durch Antrieb oder Antwort auf Abbruch durch Master (Bit 7 = 1), Extended-Services • 100_{bin} = 4: Extended-Services (Lesen oder Schreiben, siehe Bits 4 und 5) • 001_{bin} = 1 und 010_{bin} = 2: Lese- bzw. Schreib-Auftrag, Standard-Services
3	reserviert (redundant, sollte jedoch immer mit 0 belegt werden)
4 .. 5	Extended-Services: Kennung für Lese- oder Schreib-Auftrag • 00_{bin} = 0: Lese-Auftrag einleiten (Initiate Segment Read) oder kein Auftrag (in der Antwort vom Slave) • 01_{bin} = 1: Datensegment vom Servoregler lesen (Read Segment) • 10_{bin} = 2: Schreib-Auftrag einleiten (Initiate Segment Write) • 11_{bin} = 3: Datensegment zum Servoregler schreiben (Segment Write) Standard-Services: Länge der Daten

Service-Byte (Extended-Services)	
Bit	**Inhalt**
6	Handshake (Kennung, dass ein neuer Auftrag anliegt): Dieses Bit wird vom Master bei jedem neuen Auftrag gewechselt. Der Servoregler kopiert das Bit in sein Antwort-Telegramm.
7	Status (Fehlerinformation vom Servoregler): Mit diesem Bit teilt der Servoregler in der Antwort an den Master mit, ob der Auftrag fehlerfrei ausgeführt wurde. In den Extended-Services führt der Wert „1“ zum Abbruch der sequenziellen Datenübertragung. Dieser Abbruch darf auch vom Master ausgelöst werden. • 0 = Auftrag ohne Fehler ausgeführt • 1 = Auftrag nicht ausgeführt, ein Fehler ist aufgetreten. Die sequenzielle Datenübertragung wird abgebrochen. Die Daten im Feld Data/Error sind als Fehlermeldung zu interpretieren.

In den Bits 0 bis 2 werden die Extended-Services mit 100_{bin} codiert . Anders als bei den Standard-Services, in denen der Slave immer mit der Service-Codierung 000_{bin} (= kein Auftrag) antwortet, antwortet er in den Extended-Services im Nicht-Fehlerfall mit der Service-Codierung 100_{bin} (= Extended-Services). Dadurch sind die Antworten in den Extended-Services eindeutig von den Antworten in den Standard-Services zu unterscheiden.

Bei Empfang einer ungültigen Service-Codierung antworten ESR-Servoregler mit einem Fehler-Status-Telegramm (Fehlercode $06\ 05\ 00\ 10_{hex}$, „Unzulässiger Auftragsparameter“).

Die Bits 4 und 5 geben in den Extended-Services die Art des Auftrags an (Lesen oder Schreiben).

Mit Bit 6 („Handshake“) wird der Parameterkanal wie in den Standard-Services gesteuert.

Bit 7 („Status“) bestimmt den Inhalt der Felder Data/Error in der Antwort vom Slave. Mit Status = 1 wird die sequenzielle Übertragung abgebrochen, dieser Abbruch darf auch vom Master ausgelöst werden.

9.3.3 Aufbau des Steuerungs-Bytes

Das Steuerungs-Byte wird in der sequenziellen Datenübertragung ab dem zweiten Telegramm als Byte 2 im Parameterkanal übertragen.

Die einzelnen Bits haben folgende Bedeutung:

Steuerungs-Byte	
Bit	**Inhalt**
0	Status der sequenziellen Übertragung: • 0: weitere Segmente vorhanden • 1: keine weiteren Segmente vorhanden
1 .. 3	Anzahl der gültigen Datenbytes in Data 1 .. Data 6, zulässige Werte: 001_{bin} (1) bis 110_{bin} (6)
4 .. 7	reserviert (redundant, sollte jedoch immer mit 0 belegt werden)

9.3.4 Sequenzielle Datenübertragung

9.3.4.1 Lese-Auftrag einleiten (Initiate Segment Read)

Mit dem Service-Byte im einleitenden Lese-Auftrags-Telegramm des Masters an den Antrieb bestimmt der Master die Übertragungsmethode (Bits 0 bis 2 im Service-Byte = 100_{bin}, Extended-Services) und die Art des Auftrags (Bits 4 und 5 = 00_{bin}, Start eines Lese-Auftrags, Initiate Segment Read). In den Feldern Subindex und Index wird die Adresse des Parameters angegeben, der gelesen werden soll.

Telegramm zum Antrieb: Start Lese-Auftrag in den Extended-Services (Initiate Segment Read)		
Byte	**Wert**	**Inhalt**
1	44_{hex} oder 04_{hex}	Bit 0 .. 2: xxxx x100 = Extended-Services
		Bit 3: xxxx 0xxx (reserviert)
		Bit 4 .. 5: xx00 xxxx = Start eines Lese-Auftrags (Initiate Segment Read)
		Bit 6: x1xx xxxx – falls Handshake vorher 0 war
		x0xx xxxx – falls Handshake vorher 1 war
		Bit 7: 0xxx xxxx = kein Fehler
2	...	Subindex
3 .. 4	...	Index
5 .. 8	00_{hex}	Data = $0000\ 0000_{hex}$ (ohne Bedeutung)

Der Antrieb liefert bei fehlerfreier Ausführung in seiner Antwort die Länge des Parameters als vorzeichenlosen 32-Bit-Wert (im Motorola-Format) zurück:

Antwort vom Antrieb (auf Initiate Segment Read)		
Byte	**Wert**	**Inhalt**
1	44_{hex} oder 04_{hex}	Bit 0 .. 2: xxxx x100 = Extended-Services
		Bit 3: xxxx 0xxx (reserviert)
		Bit 4 .. 5: xx00 xxxx = kein Auftrag
		Bit 6: x1xx xxxx – falls Handshake vorher 0 war
		x0xx xxxx – falls Handshake vorher 1 war
		Bit 7: 0xxx xxxx = kein Fehler
2	...	Subindex
3 .. 4	...	Index
5	...	Länge (Bit 31 .. 24)
6	...	Länge (Bit 23 .. 16)
7	...	Länge (Bit 15 .. 8)
8	...	Länge (Bit 7 .. 0)

Derzeit ist die Länge eines Parameters maximal 128 Byte.

9.3.4.2 Daten lesen (Read Segment)

Nachdem der Lese-Auftrag eingeleitet wurde, folgen je nach Datenlänge ein oder mehrere Datensegment-Lesen-Aufträge (Bits 4 und 5 im Service-Byte = 01_{bin}, Read Segment):

Telegramm zum Antrieb: Datensegment lesen (Read Segment)			
Byte	**Wert**	**Inhalt**	
1	54_{hex} oder 14_{hex}	Bit 0 .. 2:	xxxx x100 = Extended-Services
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx01 xxxx = Datensegment lesen (Read Segment)
		Bit 6:	x1xx xxxx – falls Handshake vorher 0 war
			x0xx xxxx – falls Handshake vorher 1 war
		Bit 7:	0xxx xxxx = kein Fehler
2	00_{hex}	Steuerungs-Byte = 00_{hex} (ohne Bedeutung)	
3 .. 8	00_{hex}	Data = 0000 0000 0000 $_{hex}$ (ohne Bedeutung)	

In der Antwort gibt der Slave über das Steuerungs-Byte die Anzahl der gültigen Datenbytes in Data 1 .. Data 6 an („nnn“ = 1 bis 6, beginnend mit Data 1) und zeigt an, ob weitere Segmente folgen:

Antwort vom Antrieb (auf Read Segment)			
Byte	**Wert**	**Inhalt**	
1	44_{hex} oder 04_{hex}	Bit 0 .. 2:	xxxx x100 = Extended-Services
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx00 xxxx = kein Auftrag
		Bit 6:	x1xx xxxx – falls Handshake vorher 0 war
			x0xx xxxx – falls Handshake vorher 1 war
		Bit 7:	0xxx xxxx = kein Fehler
2	$0x_{hex}$	Bit 0 :	xxxx xxx0 – falls weitere Segmente folgen
			xxxx xxx1 – falls keine weiteren Segmente folgen
		Bit 1 .. 3:	xxxx nnnx = Anzahl der gültigen Datenbytes in Data 1 bis Data 6
		Bit 4 .. 7:	0000 xxxx (reserviert)
3	...	Data 1	
4	...	Data 2	
5	...	Data 3	
6	...	Data 4	
7	...	Data 5	
8	...	Data 6	

Ist die Anzahl der übertragenen Datenbytes kleiner als 6, stehen die gültigen Daten immer vorne (z. B. „nnn“ = 4: Data 1 bis Data 4 gültig).

9.3.4.3 Schreib-Auftrag einleiten (Initiate Segment Write)

Mit dem Service-Byte im einleitenden Schreib-Auftrags-Telegramm des Masters an den Antrieb bestimmt der Master die Übertragungsmethode (Bits 0 bis 2 im Service-Byte = 100_{bin}, Extended-Services) und die Art des Auftrags (Bits 4 und 5 = 10_{bin}, Start eines Schreib-Auftrags, Initiate Segment Write). In den Feldern Subindex und Index wird die Adresse des Parameters angegeben, der geschrieben werden soll, in den Feldern Data 1 bis Data 4 wird die Länge des Parameters als vorzeichenloser 32-Bit-Wert (im Motorola-Format) angegeben.

Telegramm zum Antrieb: Start Schreib-Auftrag in den Extended-Services (Initiate Segment Write)		
Byte	**Wert**	**Inhalt**
1	64_{hex} oder 24_{hex}	Bit 0 .. 2: xxxx x100 = Extended-Services
		Bit 3: xxxx 0xxx (reserviert)
		Bit 4 .. 5: xx10 xxxx = Start eines Schreib-Auftrags (Initiate Segment Write)
		Bit 6: x1xx xxxx – falls Handshake vorher 0 war
		x0xx xxxx – falls Handshake vorher 1 war
		Bit 7: 0xxx xxxx = kein Fehler
2	...	Subindex
3 .. 4	...	Index
5	...	Länge (Bit 31 .. 24)
6	...	Länge (Bit 23 .. 16)
7	...	Länge (Bit 15 .. 8)
8	...	Länge (Bit 7 .. 0)

Derzeit ist die Länge eines Parameters maximal 128 Byte.

Der Antrieb quittiert bei fehlerfreier Ausführung in seiner Antwort den Auftrag:

Antwort vom Antrieb (auf Initiate Segment Write)		
Byte	**Wert**	**Inhalt**
1	44_{hex} oder 04_{hex}	Bit 0 .. 2: xxxx x100 = Extended-Services
		Bit 3: xxxx 0xxx (reserviert)
		Bit 4 .. 5: xx00 xxxx = kein Auftrag
		Bit 6: x1xx xxxx – falls Handshake vorher 0 war
		x0xx xxxx – falls Handshake vorher 1 war
		Bit 7: 0xxx xxxx = kein Fehler
2	...	Subindex
3 .. 4	...	Index
5 .. 8	00_{hex}	Data = $0000\ 0000_{hex}$ (ohne Bedeutung)

9.3.4.4 Daten schreiben (Segment Write)

Nachdem der Schreib-Auftrag eingeleitet wurde, folgen je nach Datenlänge ein oder mehrere Datensegment-Schreiben-Aufträge (Bits 4 und 5 im Service-Byte = 11_{bin}, Write Segment). Der Master gibt über das Steuerungs-Byte die Anzahl der gültigen Datenbytes in Data 1 .. Data 6 an („nnn“ = 1 bis 6, beginnend mit Data 1) und zeigt an, ob weitere Segmente folgen:

Telegramm zum Antrieb: Datensegment schreiben (Write Segment)			
Byte	**Wert**	**Inhalt**	
1	54_{hex} oder 14_{hex}	Bit 0 .. 2:	xxxx x100 = Extended-Services
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx11 xxxx = Datensegment schreiben (Write Segment)
		Bit 6:	x1xx xxxx – falls Handshake vorher 0 war
			x0xx xxxx – falls Handshake vorher 1 war
		Bit 7:	0xxx xxxx = kein Fehler
2	$0x_{hex}$	Bit 0 :	xxxx xxx0 – falls weitere Segmente folgen
			xxxx xxx1 – falls keine weiteren Segmente folgen
		Bit 1 .. 3:	xxxx nnnx = Anzahl der gültigen Datenbytes in Data 1 bis Data 6
		Bit 4 .. 7:	0000 xxxx (reserviert)
3	...	Data 1	
4	...	Data 2	
5	...	Data 3	
6	...	Data 4	
7	...	Data 5	
8	...	Data 6	

Ist die Anzahl der übertragenen Datenbytes kleiner als 6, stehen die gültigen Daten immer vorne (z. B. „nnn“ = 4: Data 1 bis Data 4 gültig).

Der Antrieb quittiert bei fehlerfreier Ausführung in seiner Antwort den Auftrag:

Antwort vom Antrieb (auf Write Segment)			
Byte	**Wert**	**Inhalt**	
1	44_{hex} oder 04_{hex}	Bit 0 .. 2:	xxxx x100 = Extended-Services
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx00 xxxx = kein Auftrag
		Bit 6:	x1xx xxxx – falls Handshake vorher 0 war
			x0xx xxxx – falls Handshake vorher 1 war
		Bit 7:	0xxx xxxx = kein Fehler
2	00_{hex}	Steuerungs-Byte = 00_{hex} (ohne Bedeutung)	
3 .. 8	00_{hex}	Data = 0000 0000_{hex} (ohne Bedeutung)	

9.3.4.5 Abbruch der Datenübertragung durch den Antrieb (Abort Transfer durch Antrieb)

Der Antrieb kann die sequenzielle Datenübertragung abbrechen, indem er ein Fehler-Status-Telegramm (Bits 0 bis 2 im Service-Byte = 000_{bin}, Bit 7 = 1) sendet. Die Bytes 5 bis 8 des Fehler-Status-Telegramms enthalten den Fehlercode:

Abbruch der sequenziellen Datenübertragung durch den Antrieb		
Byte	**Wert**	**Inhalt**
1	$F0_{hex}$ oder $B0_{hex}$	Bit 0 .. 2: xxxx x000 = kein Auftrag
		Bit 3: xxxx 0xxx (reserviert)
		Bit 4 .. 5: xx11 xxxx = 4 Byte Daten im Feld Data/Error
		Bit 6: x1xx xxxx = Handshake 1 zurücksenden
		x0xx xxxx = Handshake 0 zurücksenden
		Bit 7: 1xxx xxxx = Auftrag nicht ausgeführt, Fehler
2	...	Subindex
3 .. 4	...	Index
5 .. 8	...	Fehlercode

Der Slave bricht die Datenübertragung ab, weil ein Fehler vorliegt. Die bisher übertragenen Daten verlieren damit ihre Gültigkeit:

- Bei einem Schreib-Auftrag werden die bisher übertragenen Daten vom Antrieb verworfen.
- Bei einem Lese-Auftrag müssen die bisher übertragenen Daten vom Master verworfen werden.

9.3.4.6 Abbruch der Datenübertragung durch den Master (Abort Transfer durch Master)

Auch der Master kann die sequenzielle Datenübertragung jederzeit abbrechen. Er setzt hierzu das Bit 7 („Status“) im Service-Byte auf „1“ (nur erlaubt in den Extended-Services):

Abbruch der sequenziellen Datenübertragung durch den Master		
Byte	**Wert**	**Inhalt**
1	$B4_{hex}$ oder 84_{hex}	Bit 0 .. 2: xxxx x100 = Extended-Services
		Bit 3: xxxx 0xxx (reserviert)
		Bit 4 .. 5: xx00 xxxx (ohne Bedeutung, da Bit 7 = 1)
		Bit 6: x1xx xxxx = Handshake 1 zurücksenden
		x0xx xxxx = Handshake 0 zurücksenden
		Bit 7: 1xxx xxxx = Fehler
2	00_{hex}	Steuerungs-Byte = 00_{hex} (ohne Bedeutung)
3 .. 8	00_{hex}	Data = 0000 0000_{hex} (ohne Bedeutung)

Der Antrieb antwortet mit einem Fehler-Status-Telegramm. Da auf der Seite

des Antriebs kein Fehler vorliegt, ist der Fehlercode 0:

Antwort des Antriebs (auf Abort Transfer durch Master)			
Byte	**Wert**	**Inhalt**	
1	F0 $_{hex}$ oder B0 $_{hex}$	Bit 0 .. 2:	xxxx x000 = kein Auftrag
		Bit 3:	xxxx 0xxx (reserviert)
		Bit 4 .. 5:	xx11 xxxx 4 Byte Daten im Feld Data/Error
		Bit 6:	x1xx xxxx = Handshake 1 zurücksenden
			x0xx xxxx = Handshake 0 zurücksenden
		Bit 7:	1xxx xxxx = Auftrag nicht ausgeführt
2	00 $_{hex}$	Subindex = 00 $_{hex}$ (ohne Bedeutung)	
3 .. 4	00 $_{hex}$	Index = 0000 $_{hex}$ (ohne Bedeutung)	
5 .. 8	00 $_{hex}$	Fehlercode = 0000 0000 $_{hex}$ (kein Fehler vom Antrieb)	

Durch den Abbruch der Datenübertragung verlieren die bisher übertragenen Daten ihre Gültigkeit:

- Bei einem Schreib-Auftrag werden die bisher übertragenen Daten vom Antrieb verworfen.
- Bei einem Lese-Auftrag müssen die bisher übertragenen Daten vom Master verworfen werden.

9.4 Fehlercodes

In den Feldern Data/Error sendet der ESR-Servoregler im Fehlerfall (Statusbit = 1) Angaben zur Beschreibung des Fehlers. Die einzelnen Felder haben folgende Bedeutung:

Parameterkanal		**Inhalt**
Byte	**Error**	
5	Error 1	Error-Class
6	Error 2	Error-Code
7	Error 3	Additional-Code (höchstwertiges Byte)
8	Error 4	Additional-Code (niederstwertiges Byte)

Folgende Fehlercodes sind definiert:

Parameterkanal				**Bedeutung**
Error 1	**Error 2**	**Error 3**	**Error 4**	
Class	**Code**	**Additional-Code**		
00 $_{hex}$	00 $_{hex}$	00 $_{hex}$	00 $_{hex}$	Kein Fehler
06 $_{hex}$	03 $_{hex}$	00 $_{hex}$	00 $_{hex}$	Keine Zugriffsberechtigung
06 $_{hex}$	05 $_{hex}$	00 $_{hex}$	10 $_{hex}$	Unzulässiger Auftragsparameter
06 $_{hex}$	05 $_{hex}$	00 $_{hex}$	11 $_{hex}$	Ungültiger Subindex
06 $_{hex}$	05 $_{hex}$	00 $_{hex}$	12 $_{hex}$	Datenlänge zu groß oder zu klein
06 $_{hex}$	07 $_{hex}$	00 $_{hex}$	00 $_{hex}$	Objekt existiert nicht

Parameterkanal				Bedeutung
Error 1	**Error 2**	**Error 3**	**Error 4**	
Class	**Code**	**Additional-Code**		
06 $_{hex}$	08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	Datentypen stimmen nicht überein
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	20 $_{hex}$	Auftrag momentan nicht ausführbar
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	21 $_{hex}$	Nicht ausführbar wegen Lokalsteuerung
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	22 $_{hex}$	Nicht ausführbar wegen Geräte-Betriebszustand
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	30 $_{hex}$	Wertebereich verlassen
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	40 $_{hex}$	Kollision mit anderen Werten

10 Diagnose-Alarm

Diagnose-Alarme werden vom Servoregler bei kommenden und gehenden Störungen versendet. Sie entsprechen damit funktionell der Diagnose im Profibus, sind aber inhaltlich nicht kompatibel mit den Profibus-Diagnosetelegrammen. Damit Alarme in der Steuerung ankommen, muss der Parameter *Diagnose-Alarm aktivieren* bei mindestens einem Modul aktiviert sein. Alarme werden dann diesem Modul zugeordnet an die Steuerung gemeldet.

Diagnose-Alarme werden als ExtChannelDiag vom Typ PROTOCOL_SPECIFIC (Wert 80000000 $_{hex}$) verschickt. Da die Servoregler-Fehlercodes immer 16-Bit-Nummern sind, bei Profinet im Feld ChannelDiag der Diagnosemeldung aber nur Werte von 0000 $_{hex}$ bis 7FFF $_{hex}$ zulässig sind, wird der Fehlercode auf die Felder ChannelDiag (High-Byte) und ExtChannelDiag (Low-Byte) der Diagnosemeldung verteilt. Da der Wert 0000 $_{hex}$ im Feld ExtChannelDiag nicht zulässig ist, wird das Low-Byte in diesem Feld zusätzlich mit 0100 $_{hex}$ ODER-verknüpft. Beispiel: Der Fehlercode A011 $_{hex}$ wird dann als A0 $_{hex}$ im ChannelDiag-Feld und 0111 $_{hex}$ (0011 $_{hex}$ oder 0100 $_{hex}$) im ExtChannelDiag-Feld übertragen.

Für alle bekannten Fehlercodes sind Einträge (Kombination aus ChannelDiag und ExtChannelDiag) in der GSDML-Datei mit entsprechendem Text hinterlegt, so dass das Projektierungstool sowohl den Störungscode des Servoreglers als auch eine Klartextmeldung des Fehlers ausgeben kann, falls das Tool dies unterstützt (funktioniert z. B. in TIA V13).

11 Protokollvarianten

Die Protokolle Ethernet, IP, ARP, DCP, DHCP, RPC, RT und IRT gehören zum Profinet-IO-Standard. Zusätzlich können Hersteller von Geräten weitere Protokolle implementieren.

Die Servoregler Neue Generation mit Profinet-IO-Schnittstelle unterstützen die genannten Protokolle und zusätzlich das herstellerspezifische ESR-Protokoll, das im Folgenden näher beschrieben wird.

11.1 ESR-Protokoll

Das von SPP Windows über den TCP-Treiber CommTCP benutzte ESR-Protokoll läuft standardmäßig auf Port 6668. SPP Windows kann daher direkt über CommTCP und Ethernet mit dem Servoregler kommunizieren.

Das ESR-Protokoll bildet im Wesentlichen die Schnittstelle der SPP Windows-DLLs auf ein einfaches ASCII-Protokoll ab. Dabei sendet der Client (PC) ein ASCII-Kommando an den Server (Servoregler), der dieses mit einem ASCII-Text beantwortet.

Kommandos beginnen immer mit einem Buchstaben, gefolgt von keinem bis mehreren Parametern in Form eines ASCII-Hexadezimal-Strings. Das Kommando wird mit einer CR/LF-Zeichenfolge abgeschlossen. Der Antwortstring besteht immer aus einem Fehlercode in Form eines ASCII-Hexadezimal-Strings gefolgt von keinem bis mehreren Antwort-Daten, ebenfalls in Form eines ASCII-Hexadezimal-Strings. Der Antwortstring ist mit einem LF-Zeichen abgeschlossen.

Folgende Kommandos sind definiert:

Kommando	**Format**	**Ergebnisdaten**	**Beispiel/Anmerkung**
Read	**R**iiiissll iiii = Index ss = Subindex ll = Länge	dd... Daten des Objektes (nur wenn Fehlercode = 0)	R60410002 (Statuswort lesen) Anzahl der Daten entspricht Objektlänge. Byte-Reihenfolge ist zu beachten.
Write	**W**iiiisslldd... iiii = Index ss = Subindex ll = Länge	keine, nur Fehlercode	W604000020F80 (Wert 800F $_{hex}$ ins Steuerwort schreiben) Länge der Daten muss mit der angegebenen Objektlänge übereinstimmen. Byte-Reihenfolge ist zu beachten.
Identify	**I**	3x16 Datenbytes, jeweils 16 Bytes für die Strings *vendor*, *model* und *revision*	Das erste Byte jedes Strings enthält die Länge des Strings, die Bytes 2 .. 16 (maximal) den String.

Kommando	Format	Ergebnisdaten	Beispiel/Anmerkung
Endian-Mode ermitteln	**M**	1 Datenbyte: 00 = Big-Endian (Motorola) 01 = Little-Endian (Intel)	Bestimmt, in welchem Format die Datenbytes bei Kommandos und Ergebnissen übertragen werden.
Fehlermeldung ermitteln	**E**eeeeeeee eeeeeeee = 32-Bit-Fehlercode, für den die Fehlermeldung ermittelt werden soll	n Datenbytes des Fehlermeldungs-Strings	E000000AC Der Fehlercode enthält hier nicht 0, sondern die Länge des Ergebnisstrings. Byte-Reihenfolge des Parameters eeeeeeee immer im Motorola-Format.
Interface ermitteln	**F**	4 Datenbytes	Interface-Kennung als 32-Bit-Wort. Byte-Reihenfolge ist zu beachten.
Node-Id ermitteln	**N**	4 Datenbytes	Node-Id als 32-Bit-Wort. Byte-Reihenfolge ist zu beachten.

Der Antwort-String hat eine der beiden folgenden Formen:

Variante	Format	Anmerkung
16-Bit-Fehlercodes	eeeedd ... eeee = 16-Bit-Fehlercode dd ... = optionale Ergebnisdaten (siehe Kommando-Tabelle)	Mit Ausnahme des Kommandos E werden keine Daten geliefert, wenn der Fehlercode ungleich 0000 ist.
32-Bit-Fehlercodes	**X**eeeeeeeedd ... (Präfix X) eeeeeeee = 32-Bit-Fehlercode dd ... = optionale Ergebnisdaten (siehe Kommando-Tabelle)	Mit Ausnahme des Kommandos E werden keine Daten geliefert, wenn der Fehlercode ungleich 0000 ist.

Anmerkungen zum Protokoll:

- Die Kommando-Parameter (Index, Errorcode) werden immer im Big-Endian-Format (Motorola) übertragen. Im Gegensatz dazu werden Datenbytes des Read- und Write-Kommandos entweder im Big- oder Little-Endian-Format übertragen. Daher sollte direkt nach dem Verbindungsaufbau das Übertragungsformat mit dem Kommando M ermittelt werden. Die Datenbytes müssen dann entsprechend gedreht werden.

  Beispiel: Das Schreiben des Wertes $800F_{hex}$ auf das Steuerwort erfolgt mit W60400002800F für Mode 0 (Big-Endian) und W604000020F80 für Mode 1 (Little-Endian).
- Kommandos müssen mit CR/LF abgeschlossen werden, da die Auswertung intern mit dem CR beginnt. Eine Zeilenenkennung im UNIX-Format (nur LF) ist daher nicht zulässig. Im Gegensatz dazu enthält der Antwortstring jedoch nur ein LF am Zeilenende.
- Für einfache Tests kann man telnet benutzen (Port 6668 angeben).

Besonderheiten bei direkter Kommunikation mit dem TCP-Server im Servoregler:

- Der Endian-Mode des Servoreglers ist immer Little-Endian (Mode-Kommando liefert immer 01).
- Die Kommandos F (Interface ermitteln) und N liefern unabhängig von der Geräteausstattung und eventuell eingestellten Bus-Namen immer die gleichen konstanten Werte.
- Das Kommando E liefert englischsprachige Fehlermeldungen.

Stichwortverzeichnis

- R -

- S -

- T -

- U -

- V -

- W -

- Z -