WITTENSTEIN | alpha

cynapse®

Beckhoff Steuerung – Beckhoff IO-Link Master

Implementierungsbeispiel

2022-D079891

Revision: 02

WITTENSTEIN alpha GmbH
Walter-Wittenstein-Straße 1
D-97999 Igersheim
Germany

Cybertronic Support

Bei Fragen zu diesem Implementierungsbeispiel wenden Sie sich bitte direkt an:
cybertronic-support@wittenstein.de

Customer Service

		✉	✆
Deutschland	WITTENSTEIN alpha GmbH	service@wittenstein-alpha.de	+49 7931 493-12900
Benelux	WITTENSTEIN BVBA	service@wittenstein.biz	+32 9 326 73 80
Brasil	WITTENSTEIN do Brasil	vendas@wittenstein.com.br	+55 15 3411 6454
中国	威腾斯坦（杭州）实业有限公司	service@wittenstein.cn	+86 571 8869 5856
Österreich	WITTENSTEIN GmbH	office@wittenstein.at	+43 2256 65632-0
Danmark	WITTENSTEIN AB	info@wittenstein.dk	+45 4027 4151
France	WITTENSTEIN sarl	info@wittenstein.fr	+33 134 17 90 95
Great Britain	WITTENSTEIN Ltd.	sales.uk@wittenstein.co.uk	+44 1782 286 427
Italia	WITTENSTEIN S.P.A.	info@wittenstein.it	+39 02 241357-1
日本	ヴィッテンシュタイン株式会社	sales@wittenstein.jp	+81-3-6680-2835
North America	WITTENSTEIN holding Corp.	technicalsupport@wittenstein-us.com	+1 630-540-5300
España	WITTENSTEIN S.L.U.	info@wittenstein.es	+34 93 479 1305
Sverige	WITTENSTEIN AB	info@wittenstein.se	+46 40-26 50 10
Schweiz	WITTENSTEIN AG Schweiz	sales@wittenstein.ch	+41 81 300 10 30
台湾	威騰斯坦有限公司	info@wittenstein.tw	+886 3 287 0191
Türkiye	WITTENSTEIN Güç Aktarma Sistemleri Tic. Ltd. Şti.	info@wittenstein.com.tr	+90 216 709 21 23

Inhaltsverzeichnis

1 Zu dieser Anleitung

Diese Anleitung enthält Vorgehensweisen zur beispielhaften Verwendung des WITTENSTEIN Sensors cynapse®.

In dieser Anleitung wird mit Beispielcode gearbeitet. Falls Sie entsprechende Codebeispiele benötigen, wenden Sie sich bitte an:
cybertronic-support@wittenstein.de

Das Original dieser Anleitung wurde in Deutsch erstellt, alle anderen Sprachversionen sind Übersetzungen dieser Anleitung.

1.1 Informationssymbole und Querverweise

Folgende Informationssymbole werden verwendet:

- fordert Sie zum Handeln auf

➲ zeigt die Folge einer Handlung an

ⓘ gibt Ihnen zusätzliche Informationen zur Handlung

Ein Querverweis bezieht sich auf die Kapitelnummer und die Überschrift des Zielabschnittes (z. B. Kapitel 2 „Hardwareaufbau“).

Ein Querverweis auf eine Tabelle bezieht sich auf die Tabellennummer (z. B. Tabelle „Tbl-1“).

2 Hardwareaufbau

Für das Beispielprojekt wurden folgende Hardwarekomponenten verwendet:

Steuerung:	Beckhoff C6930
IO-LINK-Master:	Beckhoff EL6224
IO-LINK-Device:	WITTENSTEIN cynapse®

Abbildung 1 zeigt die schematische Darstellung des Aufbaus. Der IO-Link Master EL6224 ist hierbei an einem Bus-Koppler EK1100 angeschlossen, der wiederum mittels EtherCAT (grün) mit der Steuerung C6930 verbunden ist. WITTENSTEIN cynapse® ist mit Port 1 des IO-Link-Masters verbunden (schwarz).

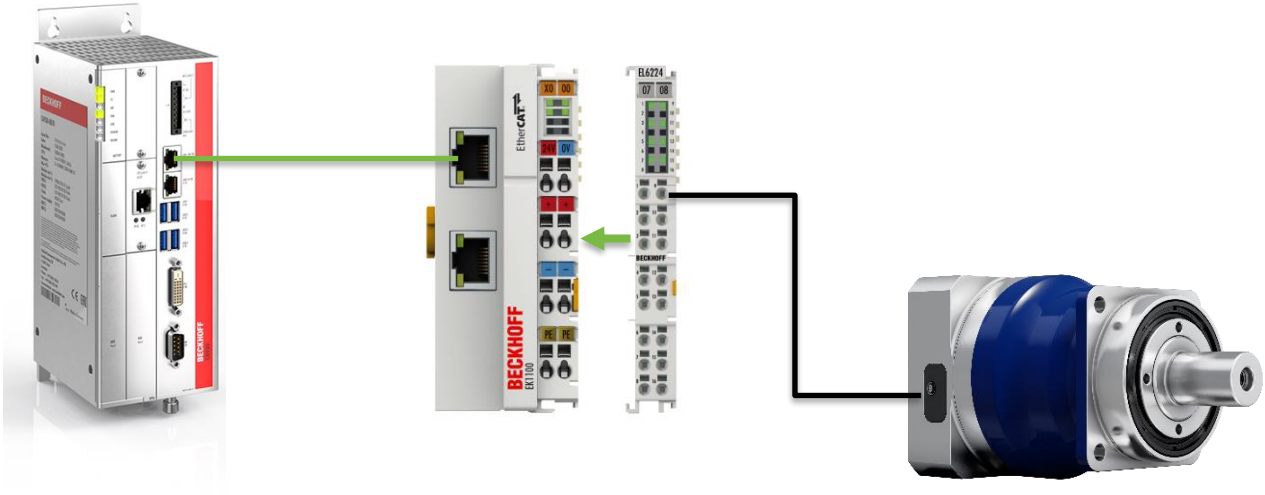


Abbildung 1: Schematischer Aufbau

3 Inbetriebnahme in TwinCAT3

Für die Durchführung der Inbetriebnahme von cynapse® benötigen Sie ein neues TwinCAT Projekt. Des Weiteren werden folgende Punkte vorausgesetzt:

- ➲ Ein Netzwerk Port an der Steuerung ist als EtherCAT-Port konfiguriert.
- ➲ TwinCAT Entwicklungsumgebung ist installiert.
- ➲ Der Hardwareaufbau ist nach Kapitel 2 erfolgt.

3.1 Konfiguration

Öffnen Sie die TwinCAT-Entwicklungsumgebung (Visual Studio) und legen Sie über „Datei“ → „Neu“ → „Projekt…“ ein neues TwinCAT-Projekt an. Im Projektmappen-Explorer ist das neue Projekt sichtbar. Es besteht die Möglichkeit das TwinCAT „lokal“ auf der C6390 oder per „remote“ von einem Engineering-PC zu verwenden. In diesem Implementierungsbeispiel wird TwinCAT „lokal“ eingesetzt.

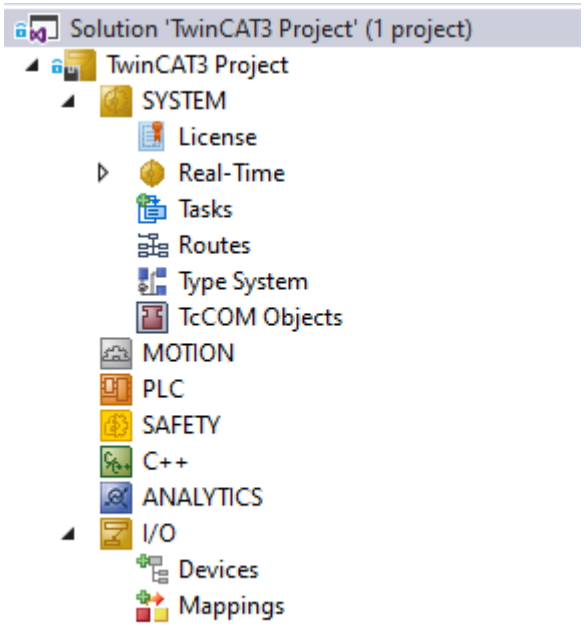


Abbildung 2: Projektmappen-Explorer

3.2 Angeschlossene Geräte einfügen

Um die angeschlossenen Geräte einfügen zu können muss TwinCAT in den „Conti Mode“ versetzt werden, falls es sich noch nicht darin befindet. Dazu Klicken Sie auf das Symbol  oder wählen über das Menü „TWINCAT“ → „Restart TwinCAT (Config Mode)“ aus. Anschließend kann im Projektmappen-Explorers das innerhalb des Elements „I/O“ befindliche „Devices“ selektiert und entweder durch Rechtsklick ein Kontextmenü geöffnet und „Scan“ ausgewählt oder durch Klick auf das Symbol  in der Menüleiste die Aktion gestartet werden.

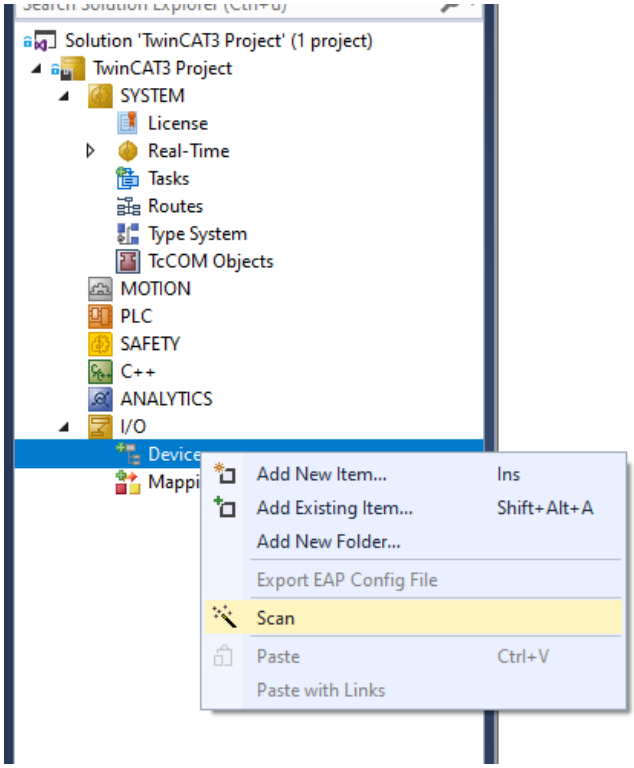


Abbildung 3: Auswahl "Scan"

Die darauffolgende Hinweismeldung ist zu bestätigen und in dem in Abbildung 4 gezeigten Dialog sind die „EtherCAT“-Geräte zu wählen.

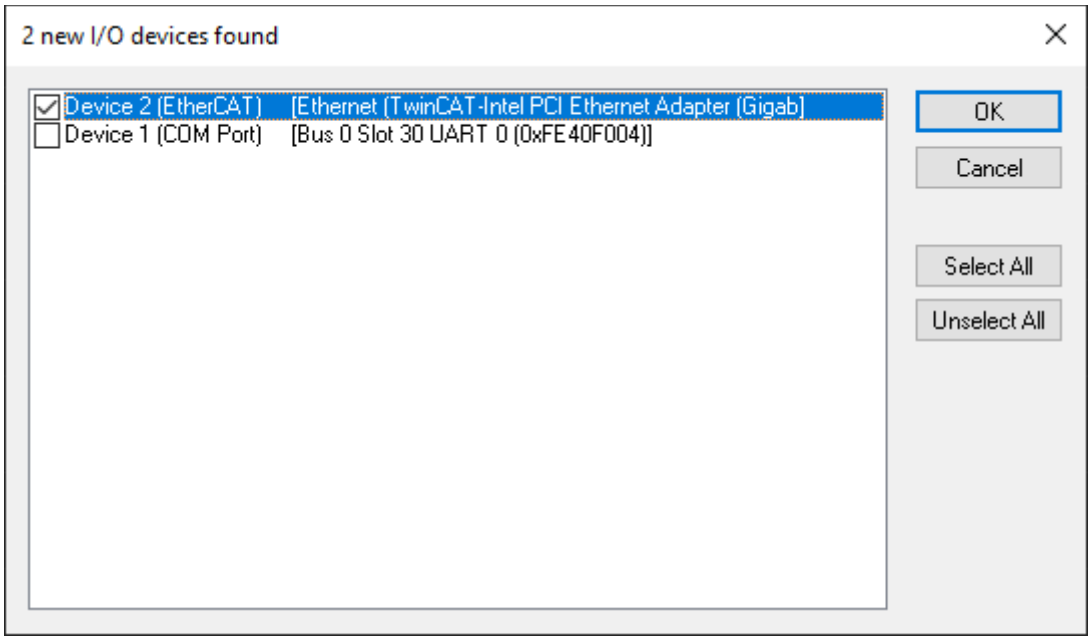


Abbildung 4: Auswahl der einzubindenden E/A-Geräte

Die darauffolgende Meldung „nach neuen Boxen suchen“ ist ebenfalls zu bestätigen, um die an die Geräte angeschlossenen Klemmen zu suchen.

Ausgehend von der hier beschriebenen Konfiguration sieht das Ergebnis wie folgt aus:

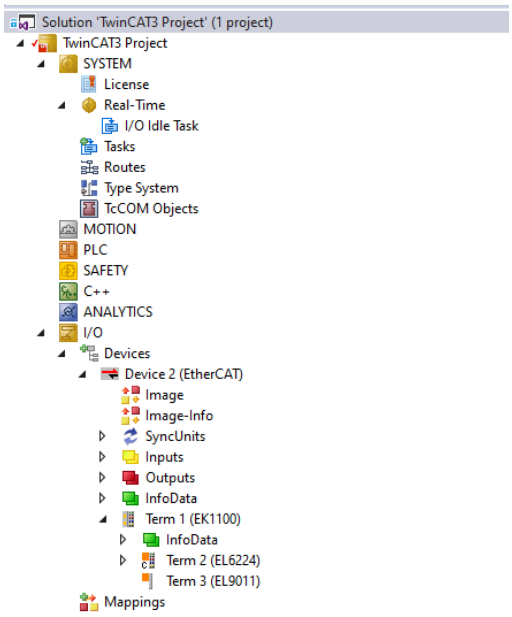


Abbildung 5: Konfiguration der TwinCAT 3 Umgebung

➲ Im Element „Devices“ befindet sich nun ein EtherCAT-Device (hier: Device 2(EtherCAT)). An diesem hängt wiederum der Buskoppler EK1100 sowie der IO-Link-Master EL6224.

3.3 Konfiguration des IO-Link-Masters EL6224

Durch Doppelklick auf den IO-Link-Master EL6224 im Projektmappen-Explorer öffnet sich das Konfigurationsmenü der Klemme. Bei der Anlage des IO-Link Masters wird ein zusätzlicher Karteireiter namens „IO-Link“ angelegt.

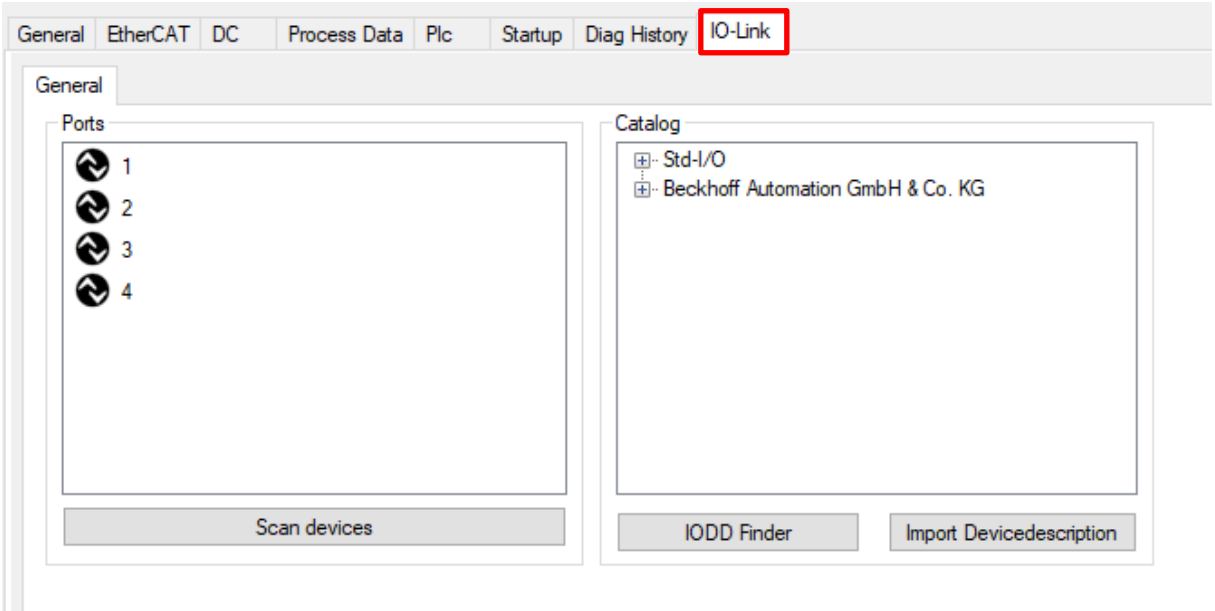


Abbildung 6: Karteireiter "IO-Link"

Um das angeschlossene IO-Link Device, hier WITTENSTEIN cynapse®, einzubinden, gibt es mehrere Möglichkeiten. Zu Beginn steht hier das Einbinden der Gerätebeschreibung IODD.

3.4 Importieren der Gerätebeschreibung IODD

Der Import der Gerätebeschreibung vereinfacht das Einbinden des IO-Link Devices. Mit Hilfe der IODD (IO Device Description) werden die einzelnen Prozessdaten aufgeschlüsselt und eine einfache Parametrierung des Sensors wird ermöglicht. Die IODD muss nur bei der erstmaligen Inbetriebnahme eines neuen IO-Link Devices importiert werden. Der Import der IODD ist Port-unabhängig.

Um die Gerätebeschreibung einzufügen, gibt es die nachfolgend beschriebenen beiden Möglichkeiten:

Mit Hilfe der Funktion „Import DeviceDescription“

Diese Methode setzt voraus, dass die benötigte IODD vorhanden ist. Die aktuelle IODD von cynapse®

kann über folgende Quelle bezogen werden:

➲ IODD Finder (https://ioddfinder.io-link.com)

Beim Import der IODD mit Hilfe des Buttons „Import DeviceDescription“ sollte wie folgt vorgegangen werden:

1. Button „Import DeviceDescription“ im Karteireiter “IO-Link” drücken
2. Die .xml-Datei des gewünschten Sensors auswählen und öffnen
3. Die importierten Dateien werden im Ordner \TwinCAT\3.X\Config\IO\IOLink abgelegt. Es ist wichtig die Dateien nicht direkt in diesen Ordner zu kopieren, sondern über „Import DeviceDescription“ einlesen zu lassen, da sonst wichtige Prüfungen umgangen werden!
4. Online-Konfiguration: Ist das IO-Link Device angeschlossen, so wird dieses über einen Klick auf „Scan Devices“ automatisch und mit den entsprechenden Parametern angelegt.
5. Offline-Konfiguration: Nach erfolgreichem Import wird die IODD im Bereich „Catalog“ angezeigt. Durch Rechtsklick auf das entsprechende Device kann dieses einem Port zugeordnet werden. Eine andere Möglichkeit der Zuweisung ist es, das Device per Drag&Drop aus dem Katalog auf den gewünschten Port zu ziehen.
6. Neustart des EtherCAT Systems oder Neuladen der Konfiguration im Config-Modus
7. Die IO-Link Geräte werden angezeigt und die Prozessdaten angelegt.

Mit Hilfe des IODD Finders

Eine weitere Möglichkeit die IODD zu laden ist über den Button „IODD Finder“. Durch Klick darauf wird eine Verbindung zum Portal IODD Finder hergestellt und alle dort hinterlegten IO-Link Devices in einer Liste angezeigt. Durch Filterung der Liste kann der gewünschte Sensor gesucht werden.

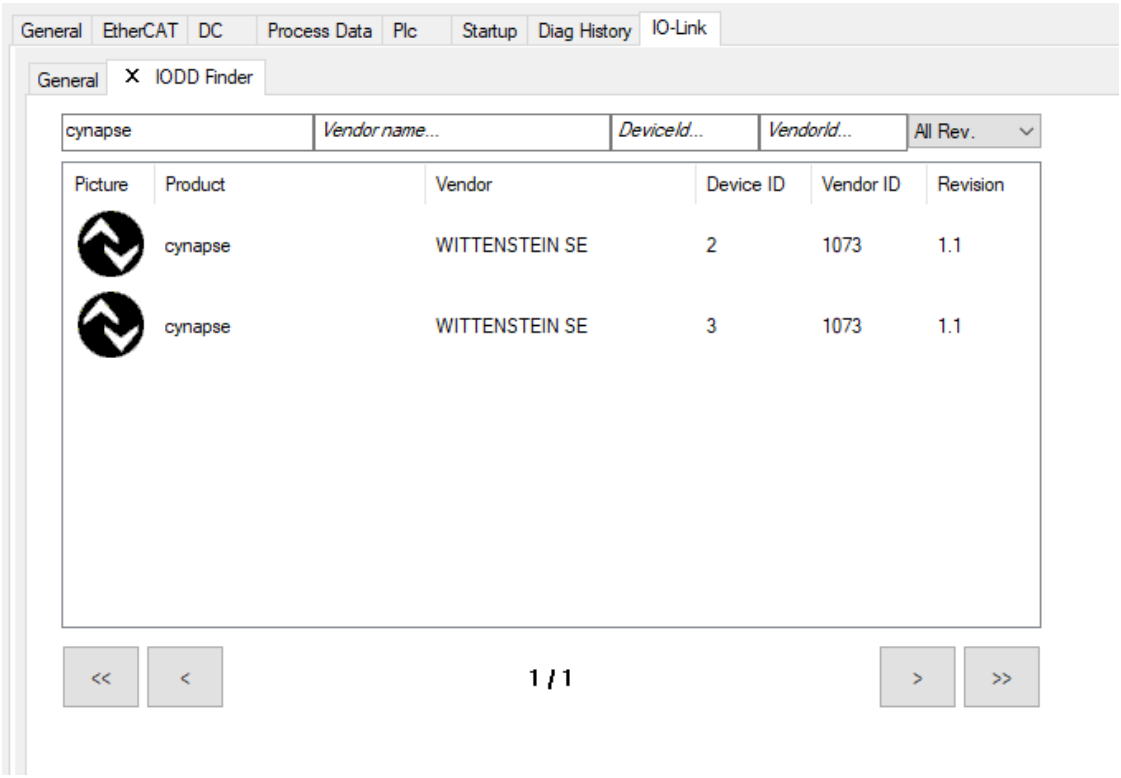


Abbildung 7: IODD Finder

Es werden zwei Einträge für cynapse® gefunden. Für neue cynapse® Sensoren verwenden Sie bitte die aktuelle Version mit der Device ID „3“. Durch einen Klick auf kann die IODD heruntergeladen werden. Hierfür ist eine Internetverbindung notwendig. Anschließend befindet sich die IODD im Katalog und kann wie in Kapitel 3.4 ab Punkt 4. beschrieben einem Port zugewiesen werden.

3.5 Anlegen eines PLC-Programms

Um eine Programmierumgebung zu schaffen, muss über das Kontextmenü von „PLC“ im Projektmappen-Explorer durch Auswahl von „Add New Item…“ ein neues PLC-Unterprojekt hinzugefügt werden:

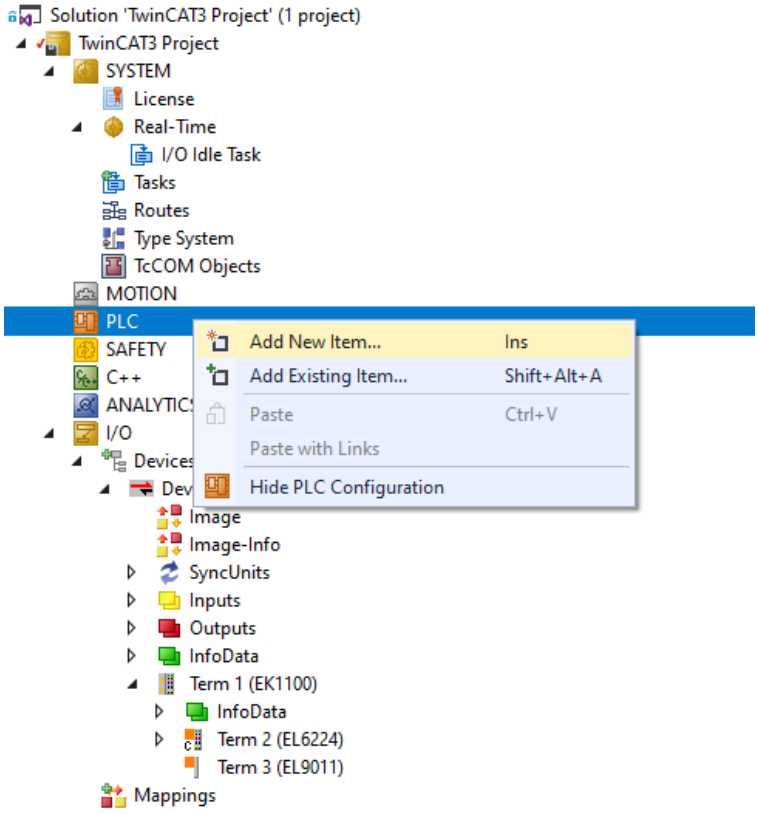


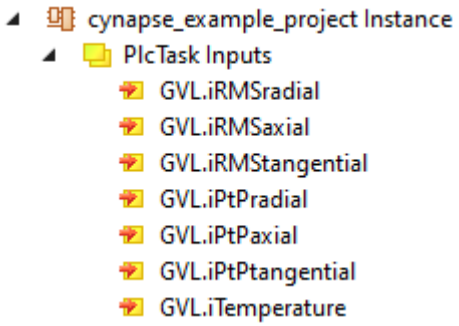
Abbildung 8: Anlegen eines PLC-Programms

In dem darauffolgenden Dialog wird ein „Standard PLC Project“ gewählt und ein Projektname (bspw. „cynapse_example_project“) vergeben. Durch die Auswahl von „Standard PLC Project“ wird das Programm „Main“ automatisch angelegt. Dieses kann über das PLC-Unterprojekt „cynapse_example_project“ in „POUs“ durch Doppelklick geöffnet werden. Außerdem wird eine globale Variablenliste (GVL) angelegt. Hier sind für den weiteren Ablauf Beispielvariablen zu erstellen, die dann mit den Eingangsvariablen von cynapse® verknüpft werden können:

```
GVL
1  {attribute 'qualified_only'}
2  VAR_GLOBAL
3      iRMSradial      AT%I* : INT;
4      iRMSaxial       AT%I* : INT;
5      iRMStangential  AT%I* : INT;
6      iPtPradial      AT%I* : INT;
7      iPtPaxial       AT%I* : INT;
8      iPtPtangential  AT%I* : INT;
9      iTemperature    AT%I* : INT;
10 END_VAR
```

Abbildung 9: Anlage von globalen Variablen

Nach dem Kompiliervorgang des PLC-Projekts über „Build“ liegen die mit „AT%“ gekennzeichneten Variablen in den „Zuordnungen“ vor und können mit den EtherCAT-Eingangsvariablen verknüpft werden.



Mittels Rechtsklick auf die entsprechende Variable wird über „Change Link…“ ein Fenster zur Auswahl dessen Verknüpfung geöffnet.

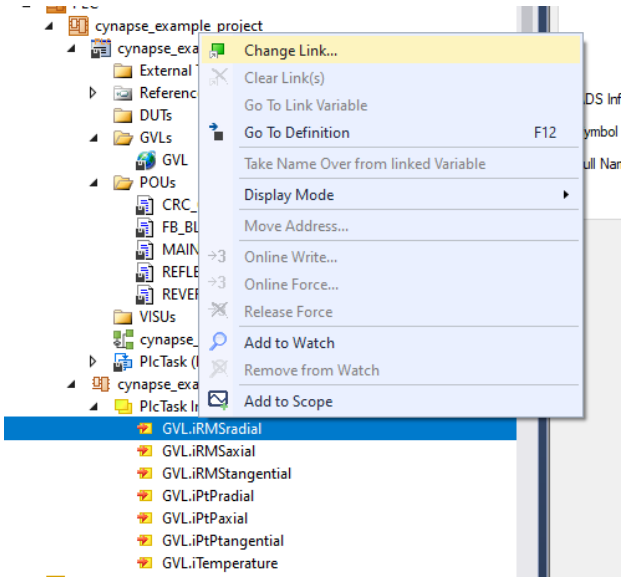


Abbildung 10: Erstellen der Verknüpfungen zwischen PLC-Variable und Prozessobjekten

3.6 Aktivieren der Konfiguration

Durch Aktivieren der Konfiguration über „TWINCAT“ → „Activate Configuration“ oder durch Klick auf das Symbol  wird die TwinCAT-Steuerung in Run-Mode gesetzt und die Prozessdaten von cynapse® zyklisch abgefragt.

4 Zugriff auf IO-Link Daten

Die Beckhoff IO-Link Masterklemme EL6224 teilt sich in zwei Dienste auf. Zum einen stellt sie einen IO-Link Master für die angeschlossenen Sensoren da, zum anderen ist sie ein EtherCAT-Slave des übergeordneten TwinCAT-Masters.

Grundsätzlich werden zyklische und azyklische Daten zwischen IO-Link Master und IO-Link Slave ausgetauscht. Aus Sicht des EtherCAT-Masters kann über die PDOs auf die zyklischen Prozessdaten zugegriffen werden. Der Zugriff auf die azyklischen Daten (Blob, Parameter und Events) erfolgt mittels AoE.

4.1 Prozessdaten lesen

Nach erfolgreicher Konfiguration des jeweiligen IO-Link-Ports werden die Prozessdaten nach einem Neustart des EtherCAT-Systems im Reiter „Process Data“ angezeigt. Die Prozessdaten werden zyklisch im eingestellten Systemtakt abgefragt und können zur weiteren Verwendung mit PLC-Variablen verknüpft werden.

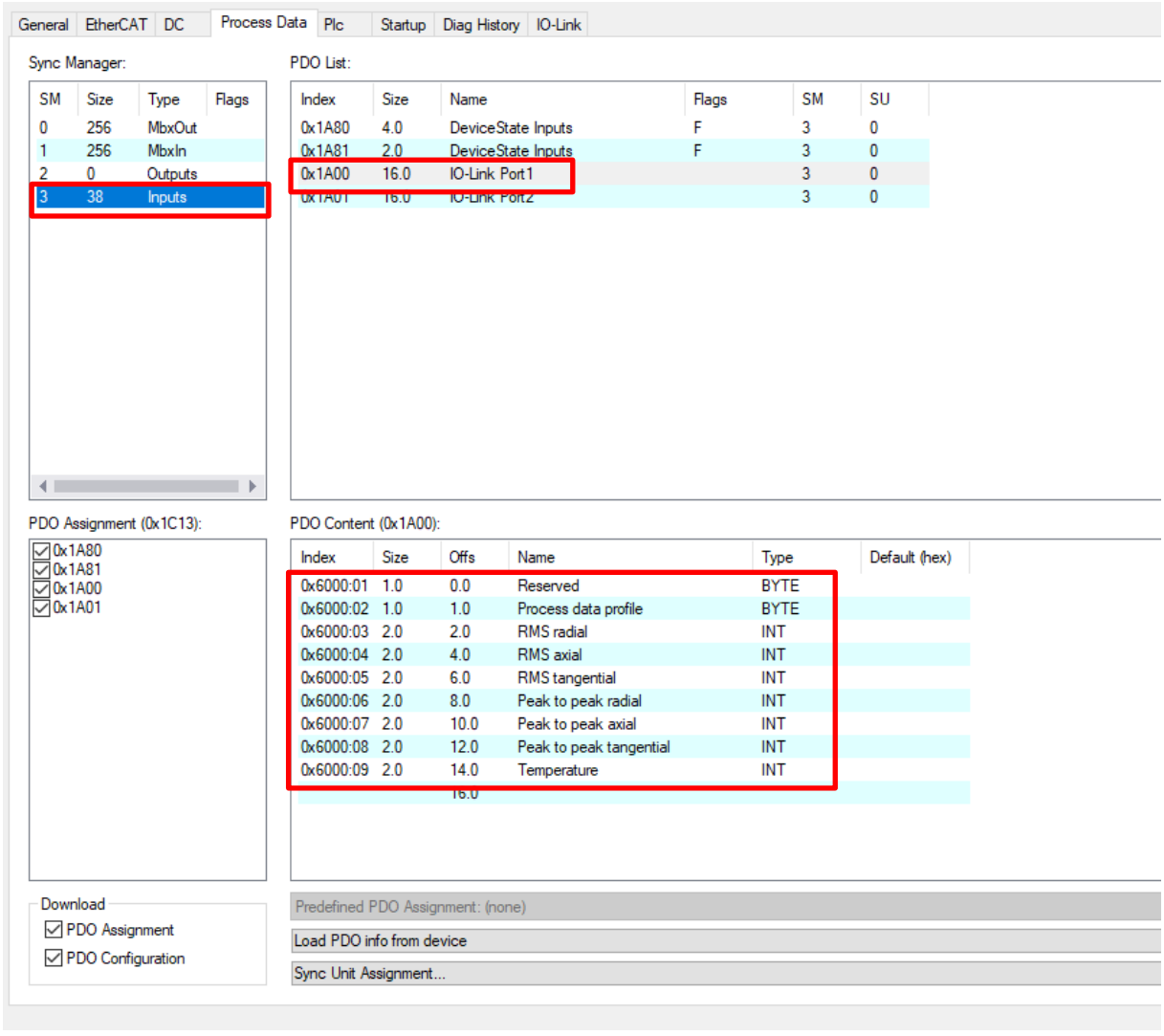


Abbildung 11: Reiter "Process Data"

WITTENSTEIN cynapse® sendet Prozessdaten die aktuelle Temperatur, sowie verschiedene Beschleunigungskennzahlen. Es werden verschiedene Prozessdatenformate bereitgestellt, um bei gleichbleibender Prozessdatenlänge von 16 Byte verschiedene Kenndaten anzubieten. Die Konfiguration des Prozessdatenformats erfolgt wie in Kapitel 4.2 beschrieben.

cynapse® nutzt keine ausgehenden (aus Sicht des IO-Link-Masters) Prozessdaten.

Nähere Informationen zu den Prozessdaten entnehmen Sie bitte der Betriebsanleitung cynapse®.

4.2 Prozessdatenformat konfigurieren

cynapse® bietet verschiedene Prozessdatenformate an. Die Prozessdatenlänge ändert sich nicht. Das Prozessdatenformat kann über den Parameter „Settings“ (Index 0x60, Subindex 0x09) geändert werden. Der Parameter kann über das PLC-Programm (siehe Kapitel 4.4) oder in der Portkonfiguration des IO-Link Masters geändert werden. Letzteres erfolgt über einen Rechtsklick auf den entsprechenden Port im Reiter „IO-Link“ mit anschließendem Klick auf „Parameter“.

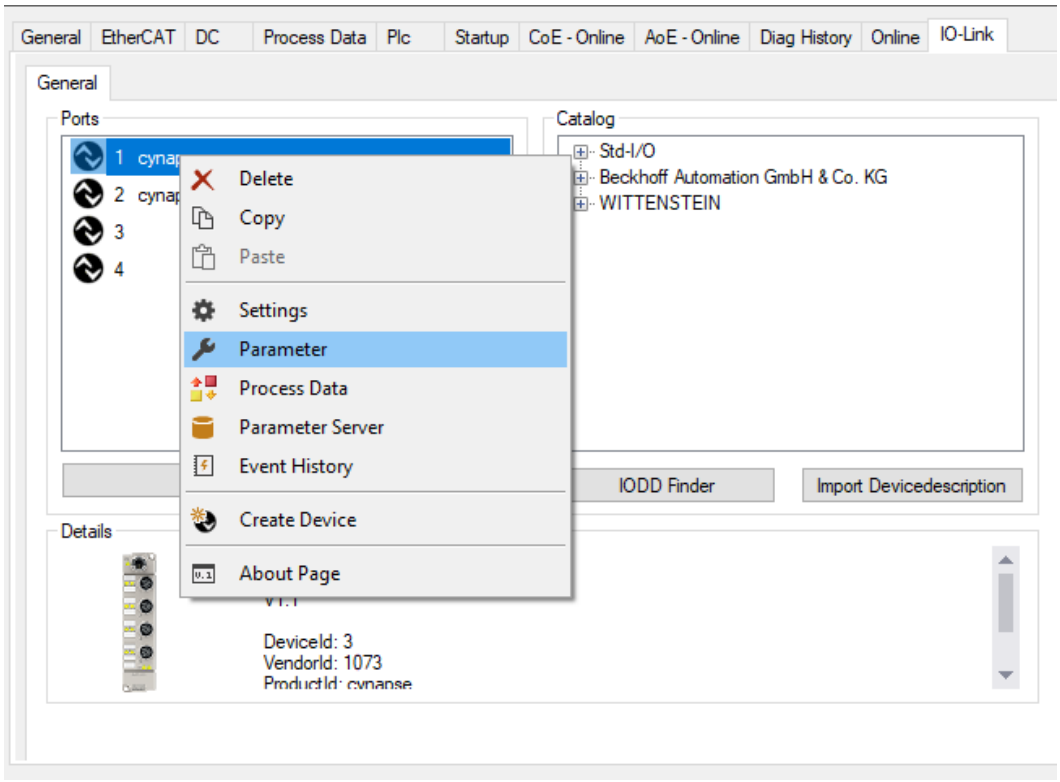


Abbildung 12: Öffnen Reiter "Parameter"

Dort können alle Parameter von cynapse® gelesen bzw. geschrieben werden. Bei der Änderung des Prozessdatenformats ist zu beachten, dass die Namen der Prozessdaten nicht mitgeändert werden. Diese sind unabhängig vom gewählten Prozessdatenformat immer gleich. Nähere Informationen zu den verfügbaren Prozessdatenformaten finden Sie in der Betriebsanleitung cynapse®.

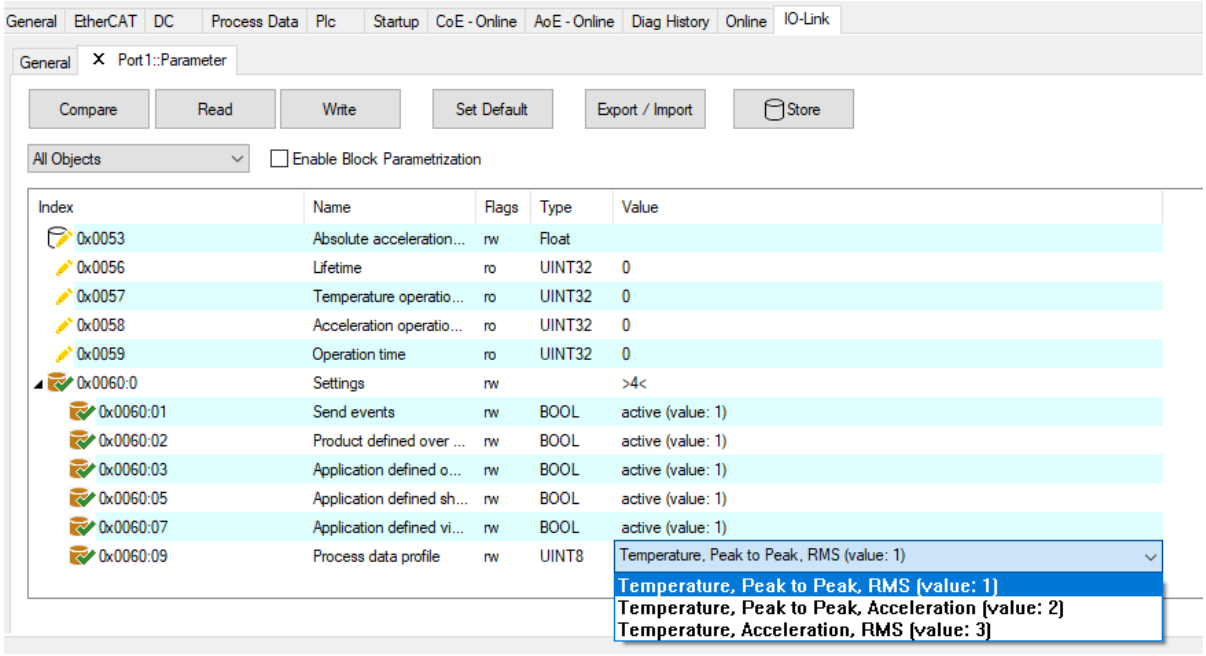


Abbildung 13: Änderung Prozessdatenformat

4.3 Parameter schreiben und lesen

cynapse® unterstützt die Parametrierung durch ISDU (Indexed Service Data Unit). Diese azyklischen Parameter müssen über die SPS explizit angefragt oder gesendet werden. Der Zugriff erfolgt über ADS oder CoE. Nachfolgend wird das Lesen und Schreiben von Parametern über ADS beschrieben.

Eine ADS-Adresse besteht immer aus NetID und Port-Nummer. Ein ADS-Befehl wird von TwinCAT über AoE (ADS over EtherCAT) an den IO-Link-Master AL1330 gesendet und von dort an den Bedarfdatenkanal weitergeleitet.

AoE-NetID

Der EtherCAT-Slave AL1330 besitzt zur Kommunikation eine eigene NetID. Diese kann in der Klemmenkonfiguration im Reiter „EtherCAT“ unter
„Advanced Settings“->“Mailbox“->“AoE“ nachgeschlagen werden.

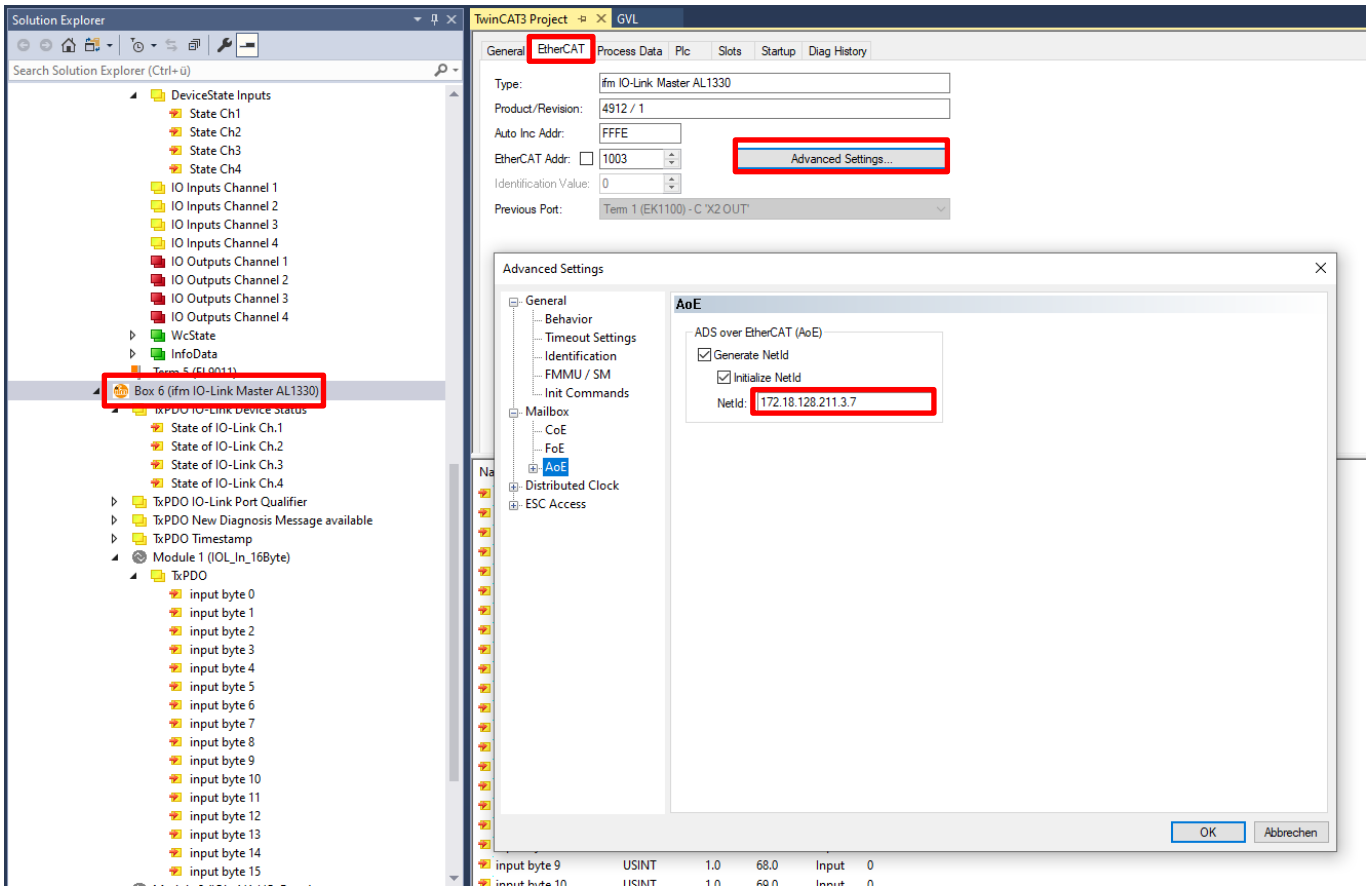


Abbildung 14: ADS-Adresse

Portnummer

Die Zuordnung der Abfrage zu einem einzelnen IO-Link Port erfolgt über die Portnummer. Die Portnummer wird aufsteigend ab 0x1000 vergeben. IO-Link Port 1 entspricht somit 0x1000 und IO-Link Port n entspricht der Portnummer 0x1000+n-1.

Für die hier verwendete Klemme EL6224 mit 4 IO-Link Ports gilt somit folgende Festlegung

IO-Link Port 1 → Portnummer 0x1000

IO-Link Port 2 → Portnummer 0x1001

IO-Link Port 3 → Portnummer 0x1002

IO-Link Port 4 → Portnummer 0x1003

ADS Indexgroup

Die Indexgroup eines ADS Befehls ist auf 0xF302 für den IO-Link-Bedarfsdatenkanal festgelegt.

ADS Indexoffset

Im Indexoffset ist die IO-Link Adressierung mit Index und Subindex codiert. Der Indexoffset ist 4-Byte groß und ist wie folgt aufgeteilt:

- Bit 16-31: Index
- Bit 8-15: reserved
- Bit 0-7: Subindex

Bsp.: Index 0x005D und Subindex 0x02 entspricht dem Indexoffset 0x005D0002

Die verfügbaren azyklischen Parameter lassen sich in der Betriebsanleitung cynapse® nachlesen.

4.4 Beispielprojekt: Parameter lesen/schreiben

Im Folgenden wird beschrieben, wie mit Hilfe der Funktionsbausteine „ADSRead“ und „ADSWrite“ Parameter über AoE gelesen bzw. geschrieben werden können.

Die beiden Bausteine „ADSRead“ und „ADSWrite“ sind Bestandteil der Beckhoff eigenen Bibliothek „TC2_Standard“. Die Bibliothek wird standardmäßig bei der Neuanlage eines PLC-Projekts geladen.

Das nachfolgende Beispiel demonstriert das Schreiben und Lesen des Parameters „Operating Temperature Threshold“ (Temperaturschwellwert) mit Hilfe der Funktionsbausteine „ADSREAD“ und „ADSWRITE“. Zunächst wird ein neuer Schwellwert mit der Instanz „fbIOLWrite“ des „ADSWrite“- Bausteins geschrieben und anschließend mit der Instanz „fbIOLRead“ zur Überprüfung gelesen.

ⓘ Das komplette Beispielprojekt erhalten Sie auf Nachfrage unter: cybertronic-support@wittenstein.de

Deklarationsteil:

```
PROGRAM MAIN
VAR
    fbIOLRead      : ADSRead;
    fbIOLWrite     : ADSWrite;

    iState         : INT;
    bExecute       : BOOL;

    rTemperatureThresholdWrite      : REAL := 40; //°C
    rTemperatureThresholdRead       : REAL;

    bBusy          : BOOL;
    bError         : BOOL;
    nErrID         : UDINT;
END_VAR
```

Implementierung:

```
CASE iState OF
0 : IF bExecute THEN
      bBusy := TRUE;
      bError := FALSE;
      nErrID := 0;

      // Write new Process Data Profile to cynapse®

      fbIOLWrite(
         NETID:='172.18.128.211.3.3' ,  //AoE-NetID EL6224
         PORT:= 16#1000,                //PortNr IO-Link Port
         IDXGRP:= 16#F302,              //Defined by Beckhoff
         IDXOFFS:= 16#00520000,         //Index = 0x0052 and Subindex = 0x00
         LEN:= SIZEOF(rTemperatureThresholdWrite),
         SRCADDR:= ADR(rTemperatureThresholdWrite),
         WRITE := TRUE );

      iState := 1;
   END_IF


1 : fbIOLWrite( WRITE := FALSE, BUSY=>bBusy, ERR=>bError, ERRID=>nErrID );

   IF NOT bBusy THEN
      IF NOT bError THEN
         iState := 2; //Success
      ELSE
         iState := 100; //Error
      END_IF
   END_IF

2 : //Read Process Data Profile
   fbIOLRead(
      NETID:='172.18.128.211.3.3' ,  //AoE-NetID EL6224
      PORT:= 16#1000,                //PortNr IO-Link Port
      IDXGRP:= 16#F302,              //Defined by Beckhoff
      IDXOFFS:= 16#00520000,         //Index = 0x0052 and Subindex = 0x00
      LEN:= SIZEOF(rTemperatureThresholdRead),
      DESTADDR:= ADR(rTemperatureThresholdRead),
      READ := TRUE );

   iState := 3;


3 : fbIOLRead( READ := FALSE, BUSY=>bBusy, ERR=>bError, ERRID=>nErrID );

   IF NOT bBusy THEN
      IF NOT bError THEN
         iState := 4; //Success
      ELSE
         iState := 100; //Error
      END_IF
   END_IF

4 : //Compare Read and Write Value
   IF rTemperatureThresholdRead = rTemperatureThresholdWrite THEN
      iState := 0; //Success
   ELSE
      iState := 100; //Error
   END_IF
```

```
    bExecute := FALSE;

100 :  //Implement Error Handler here

END_CASE
```

4.5 Blob Transfer

IO-Link definiert den Austausch größerer Datenmengen durch das BLOB (**B**inary **l**arge **ob**ject) Transfer Profil. cynapse® nutzt dies zum Versenden gesammelter Daten. Der Transfer erfolgt auf Anfrage. Es werden unterschiedliche Datenpakete von cynapse® bereitgestellt, diese entnehmen Sie bitte der Betriebsanleitung cynapse®.

Nachfolgend wird die Übertragung des Beschleunigungsdatenpakets (acceleration data package) näher beschrieben:

Das Datenpaket enthält neben den Rohdaten des Beschleunigungssensors den Stand der vier Betriebszeitzähler und der Temperatur zum Zeitpunkt der Messaufzeichnung.

Zu Beginn muss zunächst das Datenpaket über das Kommando „request acceleration data package“ angefragt werden. Dazu wird mittels „ADSWRITE“-Baustein der Wert 0xA8 über den IO-Link Index 0x02 gesendet. Ist dies erfolgt, kann das Datenpaket anschließend per Blob mit der ID -4097 ausgelesen werden. Der Ablauf des Blob Transfers ist in Abbildung 15 schematisch dargestellt.

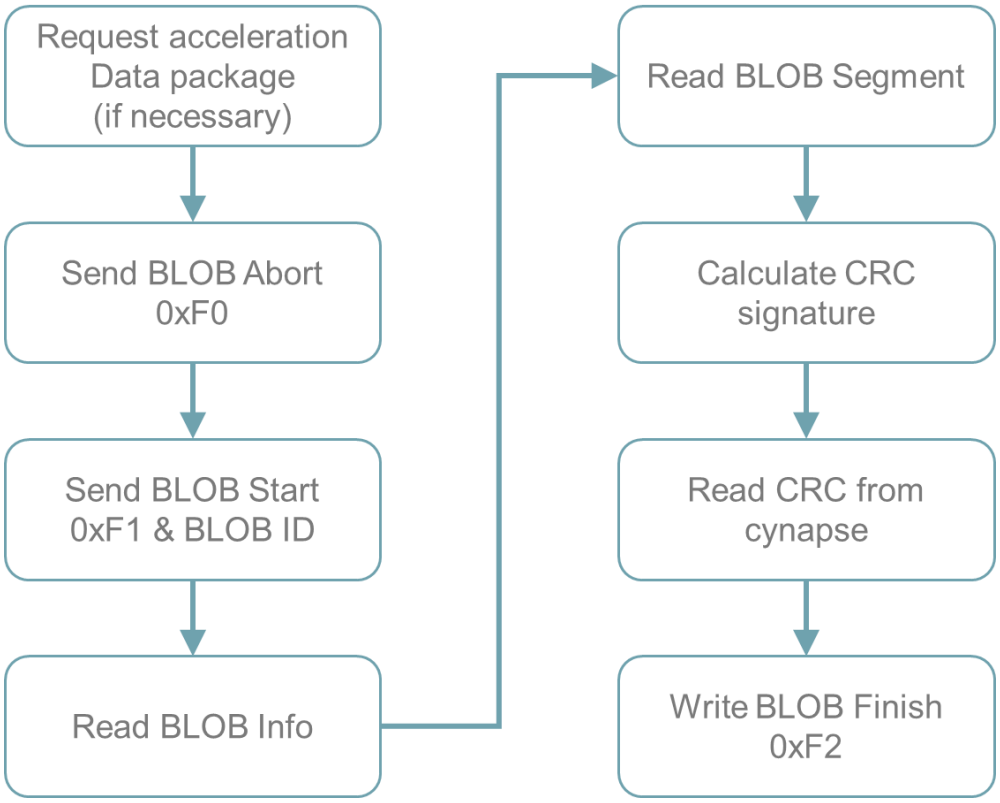


Abbildung 15: Ablauf BLOB Transfer

Nach der Anfrage des Beschleunigungsdatenpakets wird das Kommando „BLOB Abort“ geschrieben. Die Kommunikation läuft während des BLOB Transfers immer über den IO-Link Index 0x32 „BLOB Channel“. Die benötigten Übertragungsobjekte sind in Abbildung 12 aufgelistet. Nähere Informationen dazu entnehmen Sie bitte der „IO-Link Profile BLBs & FW-Update Specification“. Der Befehl „BLOB Abort“ beendet alle aktiven BLOB Übertragungen und setzt die „BLOB State Machine“ des Sensors zurück. Das Kommando ist nicht zwingend erforderlich und dient der Fehlervermeidung.

Anschließend wird das Objekt „BLOB Start“ zusammen mit der „BLOB ID“ gesendet. Die von cynapse® unterstützten „BLOB IDs“ entnehmen Sie bitte der Betriebsanleitung. cynapse® liefert daraufhin die Länge des zu lesenden BLOBs als „BLOB Info“ zurück. Die eigentlichen Messdaten werden in Segmenten mit einer Länge von maximal 232 Byte, wobei das erste Byte keine Messdaten, sondern die Anzahl der gesendeten Segmente enthält.

Während der Segmentübertragung wird eine zyklische Redundanzprüfung (CRC: cyclic redundancy check) durchgeführt und abschließend mit dem CRC-Wert aus cynapse® abgeglichen. Dadurch kann ein Datenverlust bei der Übertragung erkannt werden. Der BLOB Transfer wird mit dem Kommando „BLOB Finish“ beendet.

Das folgende Beispielprogramm zeigt den Ablauf des Blob Transfers in strukturierter Text (ST).

ⓘ Das Projekt mit den beiden Bausteinen FB_BLOB_Read und CRC_Gen (Berechnung des CRC-Wertes) erhalten Sie auf Nachfrage unter cybertronic-support@wittenstein.de

Das Programm hält einen Puffer „ptBlobData“ mit 600000 Byte bereit, dies entspricht der maximalen Länge eines BLOBs von cynapse®. Je nach gewählter BLOB ID kann dieser Puffer kleiner gewählt werden. Siehe hierzu die Betriebsanleitung cynapse®.

Deklarationsteil

```
FUNCTION_BLOCK FB_BLOB_Read
VAR_INPUT
        IOLMasterNetID : T_AmsNetID;            // NetID IO-Link Master
        uiIOLPort      : UINT;                  // IO-Link Port
        bExecute : BOOL;

        ptBlobData : POINTER TO ARRAY[0..600000] OF BYTE;
END_VAR

VAR_OUTPUT
        bError : BOOL;
        bBusy : BOOL;
        bDone : BOOL;
        ErrID : UDINT;
        RD_length : UDINT;
END_VAR

VAR
        BlobData : ARRAY[0..600000] OF BYTE;     //Blob Data
        iStep : INT;

        rTrigExecute : R_TRIG;
        fbTrigDataPackage : ADSWRITE; //FB Request Acceleration Data Package
        DataPackage : BYTE := 16#A8; //Request Acceleration Data Package
        tWaitTimer : TON;
```

```
        fbBlobWrite : ADSWRITE;
        fbBLOBRead : ADSREADEX;

        BlobAbort : BYTE := 16#F0;
        BlobFinish : BYTE := 16#F2;
        BlobStart : ARRAY[0..2] OF BYTE;
        BlobRead : ARRAY[0..255] OF BYTE;
        BlobCRC : ARRAY[0..4] OF BYTE;

        l_Blob_ID : INT := -4097;  //ID for BLOB Transfer

        ptrBlobBuffer : POINTER TO BYTE;
        udiBlobLength : UDINT;
        i : INT;
        iSgmtCnt : INT;
        udiCnt: UDINT;
        dwBlobLength : DWORD;
        CRC : DWORD;
        dwBlobCRC : DWORD;
END_VAR
```

Implementierung:

```
//Function Block ADSWrite: Request Acceleration Data Package 0xA8
fbTrigDataPackage(
    NETID:= IOLMasterNetID,
    PORT:= uiIOLPort,
    IDXGRP:= 16#F302, //Indexgrp of ADS command is specified as 0xF302 for the IO
link data channel
    IDXOFFS:= 16#020000, //IOLink-Index 0x02 Subindex 0x0 (System Command)
    LEN:= SIZEOF(DataPackage),
    SRCADDR:= ADR(DataPackage) );

//Function Block ADSWrite: Write Blobchannel
fbBlobWrite(
    NETID:= IOLMasterNetID,
    PORT:= uiIOLPort,
    IDXGRP:= 16#F302, //Defined by Beckhoff
    IDXOFFS := 16#320000 //IOLink-Index 0x32 Subindex 0x0 (Blob Channel)
    );

//Function Block ADSRead: Read BlobChannel
fbBlobRead(
    NETID:= IOLMasterNetID,
    PORT:= uiIOLPort,
    IDXGRP:= 16#F302, //Defined by Beckhoff
    IDXOFFS := 16#320000 //IOLink-Index 0x32 Subindex 0x0 (Blob Channel)
    );

//Error Handling
IF fbTrigDataPackage.ERR THEN
    iStep := 1000;
    ErrID := fbTrigDataPackage.ERRID;
END_IF

IF fbBlobRead.ERR THEN
    iStep := 1000;
    ErrID := fbBlobRead.ERRID;
END_IF

IF fbBlobWrite.ERR THEN
    iStep := 1000;
```

```
    ErrID := fbBlobWrite.ERRID;
END_IF


//*****************************************************
// BLOB-Data Command
//*****************************************************
//Wait for rising Edge bExecute
rTrigExecute(CLK:= bExecute);

CASE iStep OF
0 : //Wait for bExecute (rising Edge)
    IF rTrigExecute.Q THEN
        bBusy := TRUE;
        bError := FALSE;
        fbTrigDataPackage.WRITE := TRUE;
        iStep := 10;
    END_IF

    bDone := FALSE;

10: //Set Trigger to request Acceleration Data Package
    IF NOT fbTrigDataPackage.BUSY THEN
        fbTrigDataPackage.WRITE := FALSE;
        tWaitTimer.IN := TRUE;
        iStep := 20;
    END_IF

20: // Wait for 1s till Data Package is generated
    tWaitTimer(PT:= T#1000MS);
    IF tWaitTimer.Q THEN
        tWaitTimer.IN := FALSE;
        //Write BlobAbort to close transmission channel
        fbBlobWrite.LEN := SIZEOF(BlobAbort);
        fbBlobWrite.SRCADDR := ADR(BlobAbort);
        fbBlobWrite.WRITE := TRUE;
        iStep := 30;
    END_IF

30: IF NOT fbBlobWrite.BUSY THEN
        fbBlobWrite.WRITE := FALSE;
        iStep := 40;
    END_IF

40: //Write BlobStart to establish transmission channel for Blob ID -4097
    BlobStart[0] := 16#F1; //BLOBStart
    BlobStart[1] := INT_TO_BYTE(SHR(l_Blob_ID,8));
    BlobStart[2] := INT_TO_BYTE(l_Blob_ID);
    fbBlobWrite.LEN := SIZEOF(BlobStart);
    fbBlobWrite.SRCADDR := ADR(BlobStart);
    fbBlobWrite.WRITE := TRUE;
    iStep := 50;

50: IF NOT fbBlobWrite.BUSY THEN
        fbBlobWrite.WRITE := FALSE;

        //Read Blob Info
        fbBlobRead.LEN := SIZEOF(BlobRead);
        fbBlobRead.DESTADDR := ADR(BlobRead);
        fbBlobRead.READ := TRUE;
        iStep := 60;
    END_IF
```

```
60: IF NOT fbBlobRead.BUSY THEN
        fbBlobRead.READ := FALSE;

        //The Blob Read Info contains the BLOB Length in 4 Octets (32Bit)
        dwBlobLength := SHL(BlobRead[1], 24) OR SHL(BlobRead[2], 16) OR
                        SHL(BlobRead[3], 8) OR BlobRead[4];

                //Set BlobLength Counter to ZERO
                udiBlobLength := 0;
                //Set Segment Counter to ZERO
                iSgmtCnt := 0;
                //Initialize Pointer to BlobData
                ptrBlobBuffer := ADR(BlobData);
                iStep := 70;
            END_IF

70: //Read Blob Segment
    fbBlobRead.READ := TRUE;
    iStep := 80;

80: IF NOT fbBlobRead.BUSY THEN
        fbBlobRead.READ := FALSE;

        //Write BlobSegement to Array BlobData
        FOR udiCnt := 1 TO fbBlobRead.COUNT_R - 1 DO
         ptrBlobBuffer := ADR(BlobData)+udiBlobLength+SIZEOF(BYTE)*(udiCnt - 1);
         ptrBlobBuffer^ := BlobRead[udiCnt];
        END_FOR

        //Calculate overall length of BLOB Data
        udiBlobLength := udiBlobLength + (fbBlobRead.COUNT_R - 1);

        iStep := 81;
    END_IF

81: //CRC Check
    IF iSgmtCnt = 0 AND  BlobRead[0] <> 16#30 THEN
        //calculate CRC signature by Function CRC_GEN for Inital Segment
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),
                SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
                PN := 16#741B8CD7, INIT:=16#7FFFFFFF, REV_IN:= TRUE,
                REV_OUT:= FALSE, XOR_OUT:= 16#0);
        iStep := 70;
    ELSIF BlobRead[0] = 16#30 AND iSgmtCnt > 0 THEN
        //calculate CRC signature by Function CRC_GEN for Last Segment
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),
                SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
                PN := 16#741B8CD7, INIT:=CRC , REV_IN:= TRUE, REV_OUT:= TRUE,
                XOR_OUT:= 16#FFFFFFFF );
        iStep := 85;
    ELSIF BlobRead[0] = 16#30 AND iSgmtCnt = 0 THEN
        //calculate CRC signature by Function CRC_GEN
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),
                SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
                PN := 16#741B8CD7, INIT:=16#7FFFFFFF, REV_IN:= TRUE,
                REV_OUT:= TRUE, XOR_OUT:= 16#FFFFFFFF );
        iStep := 85;
    ELSE
        //calculate CRC signature by Function CRC_GEN
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),
```

```
                SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
                PN := 16#741B8CD7, INIT:=CRC, REV_IN:= TRUE, REV_OUT:= FALSE,
                XOR_OUT:= 16#0 );
        iStep := 70;
    END_IF

    //increment Segment Counter
    iSgmtCnt := iSgmtCnt + 1;

85: //CRC CHECK
    //calculate CRC signature by Function CRC_GEN
    // Read CRC signature from cynapse® 0x40
    fbBlobRead.LEN := SIZEOF(BlobCRC);
    fbBlobRead.DESTADDR := ADR(BlobCRC);
    fbBlobRead.READ := TRUE;
    iStep := 86;

86: IF NOT fbBlobRead.BUSY THEN
        fbBlobRead.READ := FALSE;

        // The Blob CRC contains the BLOB signature in 4 Octets (32Bit)
        dwBlobCRC := SHL(BlobCRC[1], 24) OR SHL(BlobCRC[2], 16) OR
                        SHL(BlobCRC[3], 8) OR BlobCRC[4];

        IF CRC = dwBlobCRC THEN
            iStep := 90;
        ELSE
            // if not CRC = dwBlobCRC then generate Error Message
            iStep := 1000;
        END_IF
    END_IF

90: //Write BLOB Finish 0xF2
    fbBlobWrite.LEN := SIZEOF(BlobFinish);
    fbBlobWrite.SRCADDR := ADR(BlobFinish);
    fbBlobWrite.WRITE := TRUE;
    iStep := 100;

100:IF NOT fbBlobWrite.BUSY THEN
        fbBlobWrite.WRITE := FALSE;
        bBusy := FALSE;
        bDone := TRUE;
        iStep := 0;
        MEMCPY(ptBlobData,ADR(BlobData),udiBlobLength);
        RD_Length := udiBlobLength;
    END_IF

1000://Errorhandling
    bError := TRUE;
    iStep := 0;
END_CASE
```

4.6 Events

cynapse® liefert bei ausgewählten Betriebsbedingungen sogenannte IO-Link-Events bspw. bei der Überschreitung von Vibrations- oder Temperaturschwellwerten. Diese können von der übergeordneten Steuerung ausgewertet werden.

In IO-Link gibt es 3 verschiedenen Arten von Events (Error, Warning, Information). Events vom Typ Error und Warning haben immer einen Start (Appear) und ein Ende (Disappear). Es handeln sich Event-Typen somit um zwei zeitlich versetzte Events, die vom IO-Link-Device gesendet werden. Events vom Typ Information sind sogenannte Singleshot Events. Hier gibt es nur ein einzelnes Event.

ⓘ Die von cynapse® unterstützten Events sind in der Betriebsanleitung cynapse® aufgelistet.

ⓘ Um Events senden zu können, müssen diese in cynapse® freigeschalten werden. Diese Freigabe erfolgt über den Index 0x60. Es ist eine generelle Eventfreigabe (Subindex 0x01) notwendig und eine Parameterabhängige Freigabe (Subindex 0x02 – 0x07) möglich.

4.6.1 Events auslesen mittels „Diag History“

cynapse® leitet auftretende Events an den IO-Link-Master weiter. Dieser signalisiert dies durch Setzen des Status-Bit „Device Diag“. Weiterführende Informationen zu den Events können im Karteireiter Diag History ausgelesen werden.

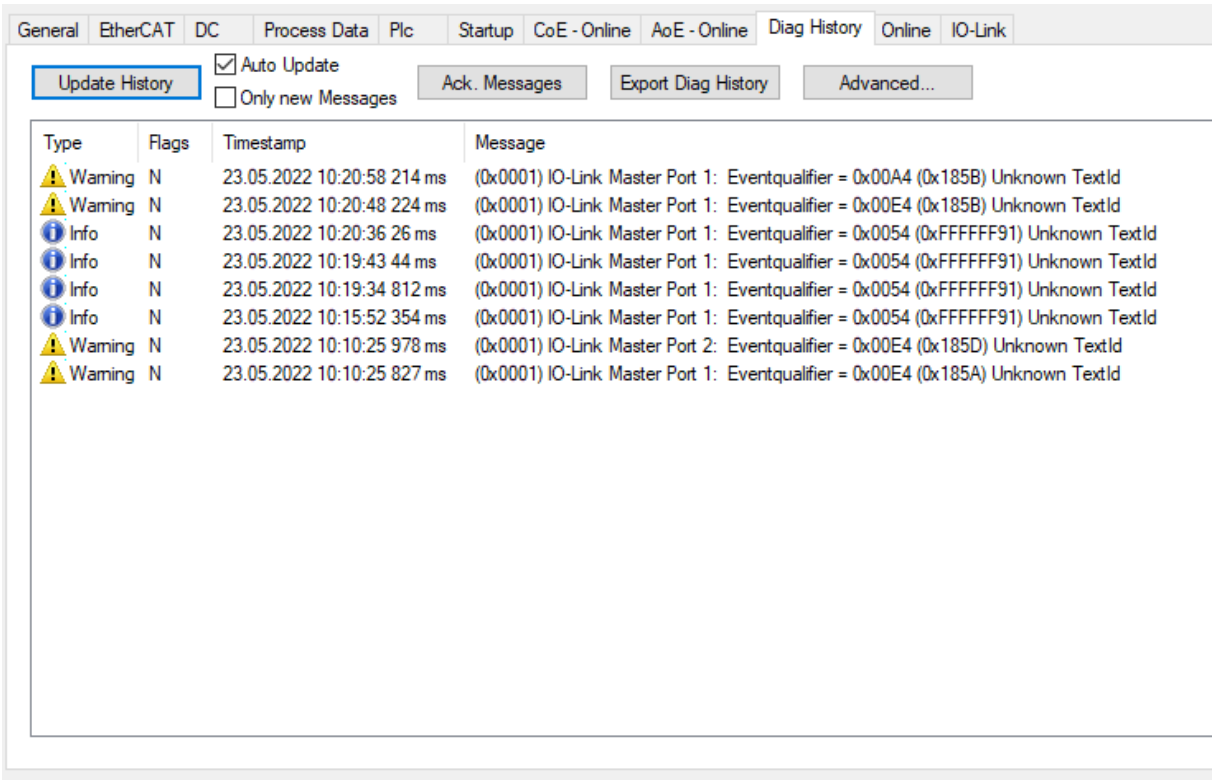


Abbildung 16: Karteireiter Diag History

Die auftretenden Ereignisse werden nach Typ (Information, Warnung, Fehler), Flag, Auftreten des Ereignisses (Zeitstempel) und Nachricht (Port-Nummer & Eventcode) aufgegliedert (siehe Abbildung 16). Anhand der Portnummer kann das IO-Link Device eindeutig zugeordnet werden.

4.6.2 Events auslesen mittels des Parameter „Detailed Device Status”

Events vom Typ Error oder Warning können zusätzlich mittels des Index 0x25 „Detailed Device Status“ ausgelesen werden. Der Parameter enthält nur auftretende Events (Appear). Der Parameter besteht aus einer Aneinanderreihung von Datenpaketen mit je 3 Byte Länge.

cynapse® liefert eine Liste mit 11 Einträgen. Sind die Werte NULL ist kein Event aktiv. Beim ersten leeren Eintrag kann die Suche somit abgebrochen werden, da die aktiven Events am Anfang der Liste enthalten sind.

Jeder 3 Byte-Eintrag teilt sich auf in Event Qualifier (Byte 1) und Event Code (Byte 2 und 3). Die Interpretation des Event Codes können der Betriebsanleitung cynapse® entnommen werden.

Beispiel

Die zyklische Abfrage des Parameters „Detailed Device Status“ Index 0x25 liefert für die ersten 9 Bytes folgendes Ergebnis:

➲ 0xE4185AE4185D000000

Unterteilt man die Antwort nun in Pakete mit der Größe von 3 Byte erhält man folgendes Ergebnis

➲ 0xE4185A 0xE4185D 0x000000

Es liegen zwei Events an. Das dritte Datenpaket ist leer und liefert keinen Eintrag, somit kann hier die Suche nach Events abgebrochen werden. Die ersten beiden Pakete enthalten anstehende Events. Das erste Byte liefert Informationen zum EventQualifier. In beiden Fällen ist dies 0xE4 und bedeutet, dass ein auftretendes Event (Appear) vom Typ „Warning“ vom Device cynapse® gesendet wurde.

ⓘ Eine detaillierte Beschreibung des EventQualifiers kann der IO-Link Specification entnommen werden.

Die beiden darauffolgenden Bytes enthalten den Event Code, der in der Betriebsanleitung cynapse® beschrieben ist.

➲ 0x185A → Die obere Temperaturschwelle des Anwenders wurde überschritten
➲ 0x185D → Die Vibrationsschwelle des Anwenders wurde überschritten

Revisionshistorie

Revision	Datum	Kommentar	Kapitel
01	02.12.2019	Neuerstellung	Alle
02	16.08.2022	cynapse® Trademark, Überarbeitung	Alle

WITTENSTEIN | alpha

WITTENSTEIN alpha GmbH · Walter-Wittenstein-Straße 1 · 97999 Igersheim · Germany
Tel. +49 7931 493-12900 · info@wittenstein.de

WITTENSTEIN – eins sein mit der Zukunft
www.wittenstein-alpha.de