Your Global Automation Partner

TURCK

TBEN-L…-4RFID…-LNX
Kompaktes RFID-Interface

Betriebsanleitung

Inhaltsverzeichnis

1 Über diese Anleitung ... 5
1.1 Zielgruppen ... 5
1.2 Symbolerläuterung ... 5
1.3 Weitere Unterlagen ... 5
1.4 Feedback zu dieser Anleitung ... 5
2 Hinweise zum Produkt ... 6
2.1 Produktidentifizierung ... 6
2.2 Lieferumfang ... 6
2.3 Rechtliche Anforderungen ... 6
2.4 Hersteller und Service ... 6
3 Zu Ihrer Sicherheit ... 7
3.1 Bestimmungsgemäße Verwendung ... 7
3.2 Allgemeine Sicherheitshinweise ... 7
4 Produktbeschreibung ... 8
4.1 Geräteübersicht ... 8
4.1.1 Bedienelemente ... 8
4.2 Eigenschaften und Merkmale ... 8
4.3 Funktionsprinzip ... 9
4.4 Funktionen und Betriebsarten ... 9
4.4.1 Linux-Distribution – Software-Komponenten ... 9
4.5 USB-Host-Port ... 9
4.6 Technisches Zubehör ... 9
5 Montieren ... 10
5.1 Gerät im Freien montieren ... 10
5.2 Gerät erden ... 11
5.2.1 Erdungs- und Schirmungskonzept ... 11
5.2.2 Gerät erden (FE) ... 12
6 Anschließen ... 13
6.1 Module an Ethernet anschließen ... 13
6.2 Versorgungsspannung anschließen ... 14
6.3 RFID-Schreib-Lese-Köpfe anschließen ... 15
6.4 Digitale Sensoren und Aktuatoren anschließen ... 16
7 In Betrieb nehmen ... 17
7.1 IP-Adresse einstellen ... 17
7.1.1 IP-Adresse über Schalter am Gerät einstellen ... 17
7.1.2 IP-Adresse über das Turck Service Tool einstellen ... 19
7.2 RFID-Kanäle programmieren ... 21
7.2.1 GPIOs der RFID-Kanäle – Übersicht ... 22
7.2.2 Slave-Controller über Skript anpassen ... 23
7.2.3 RFID-Kanäle mit Python 3 programmieren ... 25
7.2.4 RFID-Kanäle über Node.js programmieren ... 27
7.2.5 RFID-Kanäle über C oder C++ programmieren ... 28
7.3 Digitale Kanäle (DXP) programmieren ... 30
7.3.1 GPIOs der DXP-Kanäle – Übersicht ... 30

7.3.2 DXP-Funktionen über Skript einstellen ... 32
7.3.3 DXP-Kanäle mit Python 3 programmieren ... 33
7.3.4 DXP-Kanäle über Node.js programmieren ... 35
7.3.5 DXP-Kanäle über C oder C++ programmieren ... 37
7.4 LED-Funktionen programmieren ... 40
7.4.1 LEDs – Übersicht ... 40
7.4.2 LED-Funktionen über Skript einstellen ... 41
7.4.3 LED-Funktionen mit Python 3 programmieren ... 42
7.4.4 LED-Funktionen über Node.js programmieren ... 44
7.4.5 LED-Funktionen über C oder C++ programmieren ... 45
7.5 C-Applikation erstellen ... 46
7.6 Applikation automatisch starten (Autostart) ... 48
7.6.1 Autostart – Konfigurationsdatei (Unit-Datei) erstellen ... 48
7.6.2 Beispiel: Unit-Datei nutzen ... 48
7.6.3 Unit-Datei aktivieren ... 49
7.7 Zugriffsrechte verwalten ... 49
7.8 Python-Packages installieren ... 49
7.8.1 Beispiel: Python-Modul installieren ... 49
8 Einstellen ... 52
9 Betreiben ... 53
9.1 LED-Anzeigen ... 53
9.2 Gerät zurücksetzen (Reset) ... 53
10 Störungen beseitigen ... 54
11 Instand halten ... 55
11.1 Firmware-Update über die USB-Schnittstelle durchführen ... 55
11.2 Firmware-Update über die Konsole durchführen ... 56
11.2.1 Beispiel: Firmware-Update über WinSCP und PuTTY durchführen ... 57
12 Reparieren ... 63
12.1 Geräte zurücksenden ... 63
13 Entsorgen ... 63
14 Technische Daten ... 64
15 Anhang: EU-Konformitätserklärung ... 66
16 Anhang: Beispiel – „HelloGPIO“ für Node.js ... 67

1 Über diese Anleitung

Die Anleitung beschreibt den Aufbau, die Funktionen und den Einsatz des Produkts und hilft Ihnen, das Produkt bestimmungsgemäß zu betreiben. Lesen Sie die Anleitung vor dem Gebrauch des Produkts aufmerksam durch. So vermeiden Sie mögliche Personen-, Sach- und Geräteschäden. Bewahren Sie die Anleitung auf, solange das Produkt genutzt wird. Falls Sie das Produkt weitergeben, geben Sie auch diese Anleitung mit.

1.1 Zielgruppen

Die vorliegende Anleitung richtet sich an fachlich geschultes Personal und muss von jeder Person sorgfältig gelesen werden, die das Gerät montiert, in Betrieb nimmt, betreibt, instand hält, demontiert oder entsorgt.

1.2 Symbolerläuterung

In dieser Anleitung werden folgende Symbole verwendet:

	**GEFAHR** GEFAHR kennzeichnet eine gefährliche Situation mit hohem Risiko, die zum Tod oder zu schweren Verletzungen führt, wenn sie nicht vermieden wird.
	**WARNUNG** WARNUNG kennzeichnet eine gefährliche Situation mit mittlerem Risiko, die zum Tod oder zu schweren Verletzungen führen kann, wenn sie nicht vermieden wird.
	**VORSICHT** VORSICHT kennzeichnet eine gefährliche Situation mit mittlerem Risiko, die zu mittelschweren oder leichten Verletzungen führen kann, wenn sie nicht vermieden wird.
	**ACHTUNG** ACHTUNG kennzeichnet eine Situation, die zu Sachschäden führen kann, wenn sie nicht vermieden wird.
	**HINWEIS** Unter HINWEIS finden Sie Tipps, Empfehlungen und nützliche Informationen zu speziellen Handlungsschritten und Sachverhalten. Die Hinweise erleichtern Ihnen die Arbeit und helfen Ihnen, Mehrarbeit zu vermeiden.
▶	**HANDLUNGSAUFFORDERUNG** Dieses Zeichen kennzeichnet Handlungsschritte, die der Anwender ausführen muss.
⇨	**HANDLUNGSRESULTAT** Dieses Zeichen kennzeichnet relevante Handlungsresultate.

1.3 Weitere Unterlagen

Ergänzend zu diesem Dokument finden Sie im Internet unter www.turck.com folgende Unterlagen:

- Datenblatt
- EU-Konformitätserklärung

1.4 Feedback zu dieser Anleitung

Wir sind bestrebt, diese Anleitung ständig so informativ und übersichtlich wie möglich zu gestalten. Haben Sie Anregungen für eine bessere Gestaltung oder fehlen Ihnen Angaben in der Anleitung, schicken Sie Ihre Vorschläge an techdoc@turck.com.

2 Hinweise zum Produkt

2.1 Produktidentifizierung

Diese Anleitung gilt für die folgenden kompakten RFID-Interfaces:

- TBEN-L4-4RFID-8DXP-LNX
- TBEN-L5-4RFID-8DXP-LNX

2.2 Lieferumfang

- Kompaktes RFID-Interface
- Verschlusskappen für M12-Buchsen
- Kurzbetriebsanleitung

2.3 Rechtliche Anforderungen

Das Gerät fällt unter folgende EU-Richtlinien:

- 2014/30/EU (Elektromagnetische Verträglichkeit)
- 2011/65/EU (RoHS II-Richtlinie)

2.4 Hersteller und Service

Hans Turck GmbH & Co. KG
Witzlebenstraße 7
45472 Mülheim an der Ruhr
Germany

Turck unterstützt Sie bei Ihren Projekten von der ersten Analyse bis zur Inbetriebnahme Ihrer Applikation. In der Turck-Produktdatenbank finden Sie Software-Tools für Programmierung, Konfiguration oder Inbetriebnahme, Datenblätter und CAD-Dateien in vielen Exportformaten. Über folgende Adresse gelangen Sie direkt in die Produktdatenbank: **www.turck.de/produkte**
Für weitere Fragen ist das Sales-und-Service-Team in Deutschland telefonisch unter folgenden Nummern zu erreichen:

- Vertrieb: +49 208 4952-380
- Technik: +49 208 4952-390

Außerhalb Deutschlands wenden Sie sich bitte an Ihre Turck-Landesvertretung.

3 Zu Ihrer Sicherheit

Das Produkt ist nach dem Stand der Technik konzipiert. Dennoch gibt es Restgefahren. Um Personen- und Sachschäden zu vermeiden, müssen Sie die Sicherheits- und Warnhinweise beachten. Für Schäden durch Nichtbeachtung von Sicherheits- und Warnhinweisen übernimmt Turck keine Haftung.

3.1 Bestimmungsgemäße Verwendung

Die Geräte sind ausschließlich zum Einsatz im industriellen Bereich bestimmt.
Das Blockmodul TBEN-L…-4RFID-8DXP-LNX… ist ein programmierbares RFID-Interface zum Einsatz im Turck RFID-System. Das Turck RFID-System dient dem berührungslosen Austausch von Daten zwischen einem Datenträger und einem Schreib-Lese-Kopf zur Identifizierung von Objekten oder Produkten. Zum Anschluss von BL ident®-Schreib-Lese-Köpfen besitzt das Gerät vier RFID-Kanäle. Zusätzlich stehen acht konfigurierbare digitale Kanäle zur Verfügung. Die Interfaces kommunizieren über TCP/IP mit Drittsystemen wie beispielsweise ERP-Systemen.
Das Gerät darf nur wie in dieser Anleitung beschrieben verwendet werden. Jede andere Verwendung gilt als nicht bestimmungsgemäß. Für daraus resultierende Schäden übernimmt Turck keine Haftung.

3.2 Allgemeine Sicherheitshinweise

- Nur fachlich geschultes Personal darf das Gerät montieren, installieren, betreiben, parametrieren und instand halten.
- Das Gerät nur in Übereinstimmung mit den geltenden nationalen und internationalen Bestimmungen, Normen und Gesetzen einsetzen.
- Das Gerät erfüllt ausschließlich die EMV-Anforderungen für den industriellen Bereich und ist nicht zum Einsatz in Wohngebieten geeignet.

4 Produktbeschreibung

Die Geräte sind in einem vollvergossenen Kunststoffgehäuse in Schutzart IP67/IP69K ausgeführt. Zum Anschluss von Schreib-Lese-Köpfen stehen vier RFID-Kanäle zur Verfügung. Zusätzlich lassen sich Sensoren und Aktuatoren über acht frei als Eingänge oder Ausgänge konfigurierbare digitale I/O-Kanäle anschließen. Die Anschlüsse für Schreib-Lese-Köpfe und für digitale I/Os sind als M12-Buchsen ausgeführt. Zum Anschluss an das Ethernet steht eine M12-Buchse zur Verfügung. Die Anschlüsse für die Versorgungsspannung sind als 4-polige (TBEN-L4) oder 5-polige (TBEN-L5) 7/8"-Buchse ausgeführt.

4.1 Geräteübersicht

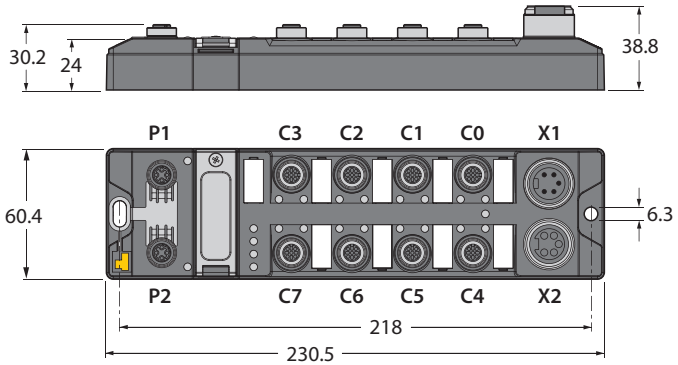

Abb. 1: Abmessungen

4.1.1 Bedienelemente

Die Geräte verfügen über die folgenden Bedienelemente:

- Drehcodierschalter und DIP-Schalter zur Einstellung der IP-Adresse
- SET-Taster zur Aktivierung der Schreibzugriffe der USB-Host-Port-Funktionen

4.2 Eigenschaften und Merkmale

- TCP/IP
- Frei programmierbares Kompaktmodul basierend auf Linux
- Programmiersprache C, C++, NodeJS, Python
- API und SDK auf Anfrage verfügbar
- Implementierung des Protokolls für die Schreib-Lese-Köpfe erforderlich
- 4 Kanäle mit M12-Anschluss für RFID
- 8 konfigurierbare digitale Kanäle als pnp-Eingänge und/oder Ausgänge 2 A
- Mehrere LEDs zur Statusanzeige
- Integrierter Ethernet-Switch ermöglicht Linientopologie
- Übertragungsrate 10 Mbps/100 Mbps
- Glasfaserverstärktes Gehäuse
- Schock- und Schwingungsgeprüft
- Vollvergossene Modulelektronik
- Schutzart IP65/IP67/IP69K

4.3 Funktionsprinzip

Die RFID-Interfaces verbinden das RFID-System mit weiteren Systemen, die über TCP/IP kommunizieren (z. B. ERP-Systemen). Die Interfaces verfügen über eine Ethernet-Schnittstelle und RFID-Schnittstellen.

Über die TCP/IP-Schnittstelle wird das RFID-System an ein Drittsystem wie beispielsweise ein ERP-System angekoppelt. Über die RFID-Schnittstellen werden die Schreib-Lese-Köpfe an die Interfaces angeschlossen. Zusätzlich können die Interfaces Signale von Sensoren und Aktuatoren über 8 konfigurierbare digitale Kanäle verarbeiten.

4.4 Funktionen und Betriebsarten

An die RFID-Kanäle können HF- und UHF-Schreib-Lese-Köpfe angeschlossen werden. Auch der parallele Betrieb von HF- und UHF-Schreib-Lese-Köpfen an einem Gerät ist möglich.
An die konfigurierbaren digitalen Kanäle können Sensoren und Aktuatoren angeschlossen werden. Insgesamt lassen sich bis zu vier 3-Draht-PNP-Sensoren bzw. vier PNP-DC-Aktuatoren mit einem maximalen Ausgangsstrom von 2 A pro Ausgang anschließen. Die Gerätefunktionen können über das Betriebssystem Linux mit C, C++, NodeJS oder Python programmiert werden. Zudem können Middleware-Funktionalitäten auf dem Gerät integriert werden.

4.4.1 Linux-Distribution – Software-Komponenten

Die Linux-Distribution des Geräts enthält die folgenden Software-Komponenten:

- SSH
- SFTP
- HTTP
- IBTP
- MTXP
- DHCP
- SNTP
- Node.js 6.9.5 (LTS)
- Python 3.x

4.5 USB-Host-Port

Das Gerät verfügt über einen USB-Host-Port zum Anschluss von USB-Speichersticks. Der USB-Host-Port ist als USB2.0-A-Buchse ausgelegt. Über die Schnittstelle kann die Geräte-Firmware aktualisiert werden. Eine Speichererweiterung über den USB-Host-Port ist nicht möglich.

4.6 Technisches Zubehör

Optional erhältliches Zubehör für Montage, Anschluss und Parametrierung finden Sie in der Turck-Produktdatenbank unter **www.turck.com**. Das Zubehör ist nicht im Lieferumfang enthalten.

5 Montieren

Das Gerät muss auf einer ebenen, vorgebohrten und geerdeten Montagefläche befestigt werden.

- Modul mit zwei M6-Schrauben auf der Montagefläche befestigen. Das maximale Anzugsdrehmonent für die Befestigung der Schrauben beträgt 1,5 Nm.

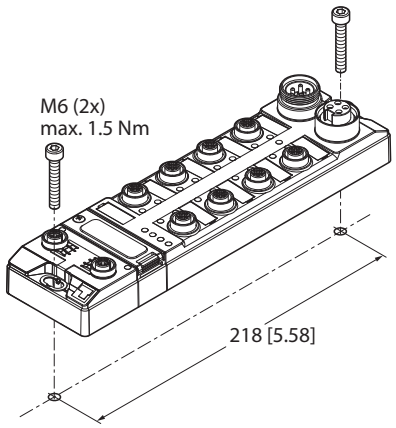

Abb. 2: Gerät auf Montageplatte befestigen

5.1 Gerät im Freien montieren

Das Gerät ist UV-beständig gemäß DIN EN ISO 4892-2. Bei direkter Sonneneinstrahlung kann es zu Materialabrieb und Farbveränderungen kommen. Die mechanischen und elektrischen Eigenschaften des Geräts werden nicht beeinträchtigt.

- Um Materialabrieb und Farbveränderungen zu vermeiden: Gerät z. B. durch die Verwendung von Schutzblechen vor direkter Sonneneinstrahlung schützen.

5.2 Gerät erden

5.2.1 Erdungs- und Schirmungskonzept

Das Erdungs- und Schirmungskonzept der TBEN-L-Module ermöglicht das getrennte Erden von Feldbus- und I/O-Teil.

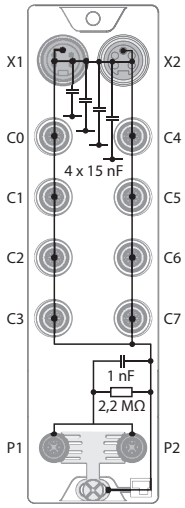

Abb. 3: Ersatzschaltbild, Schirmungskonzept

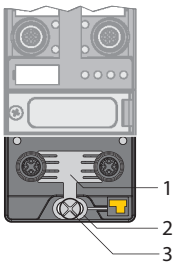

Abb. 4: Erdungskomponenten

Die Erdungsspange (1) an den M12-Steckverbindern für den Feldbusanschluss (P1, P2) verbindet den Schirm der Feldbusleitungen. Der Metallring (2) ist unterhalb der Erdungsspange angebracht und verbindet die Funktionserde der 7/8“-Steckverbinder (Pin 3) für die Spannungsversorgung mit der Funktionserde der M12-Steckverbinder (Pin 5) für den Anschluss der Schreib-Lese-Köpfe, Sensoren und Aktuatoren. Eine Metallschraube (3) verbindet das Gerät mit dem Bezugspotenzial der Anlage.

5.2.2 Gerät erden (FE)

Erdungsspange und Metallring sind miteinander verbunden. Eine Befestigungsschraube durch das untere Montageloch des Moduls verbindet die Schirmung der Feldbusleitungen mit der Funktionserde von Spannungsversorgung und angeschlossenen Geräten und dem Bezugspotenzial der Anlage. Ist ein gemeinsames Bezugspotenzial nicht erwünscht, Erdungsspange zur Entkopplung des Feldbusschirms entfernen oder Modul mit einer Kunststoffschraube befestigen.

Erdungsspange entfernen

- ▶ Erdungsspange mit einem flachen Schlitz-Schraubendreher nach oben hebeln und entfernen.

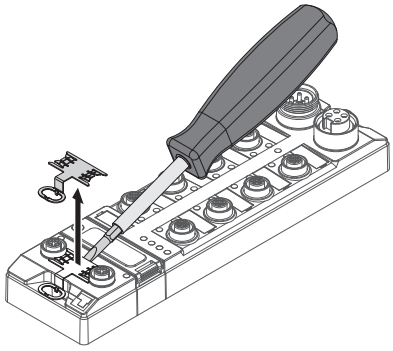

Abb. 5: Erdungsspange entfernen

Erdungsspange montieren

- ▶ Erdungsspange ggf. mit Hilfe eines Schraubendrehers zwischen den Feldbus-Steckverbindern so wieder einsetzen, dass Kontakt zum Metallgehäuse der Steckverbinder besteht.
- ⇨ Der Schirm der Feldbusleitungen liegt auf der Erdungsspange auf.

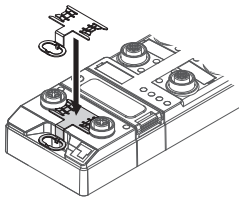

Abb. 6: Erdungsspange montieren

6 Anschließen

6.1 Module an Ethernet anschließen

Zum Anschluss an ein Ethernet-System verfügt das Gerät über einen integrierten Autocrossing-Switch mit zwei 4-poligen M12-Ethernet-Steckverbindern. Das max. Anzugedrehmoment beträgt 0,6 Nm.

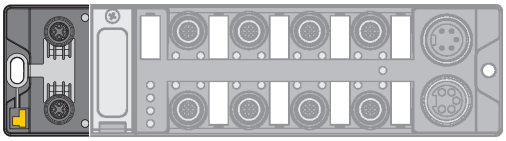

Abb. 7: M12-Ethernet-Steckverbinder zum Anschluss an den Feldbus

- ▶ Gerät gemäß unten stehender Pinbelegung an den Feldbus anschließen.

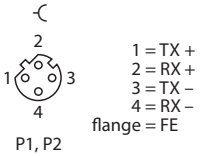

Abb. 8: Pinbelegung Ethernet-Anschlüsse

6.2 Versorgungsspannung anschließen

Zum Anschluss an die Versorgungsspannung verfügt das Gerät über zwei 7/8"-Steckverbinder. Die Steckverbinder sind 4-polig (TBEN-L4) oder 5-polig (TBEN-L5) ausgeführt. V1 und V2 sind galvanisch voneinander getrennt. Das max. Anzugsdrehmoment beträgt 0,8 Nm.

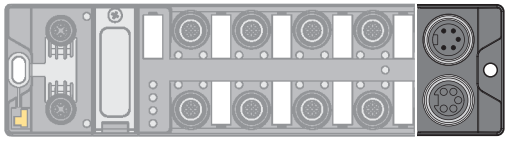

Abb. 9: 7/8"-Steckverbinder zum Anschluss an die Versorgungsspannung

▶ Gerät gemäß unten stehender Pinbelegung an die Versorgungsspannung anschließen.

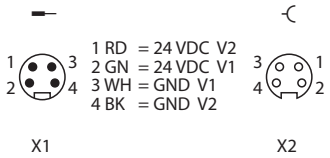

Abb. 10: TBEN-L4… – Pinbelegung Versorgungsspannungs-Anschlüsse

Abb. 11: TBEN-L5… – Pinbelegung Versorgungsspannungs-Anschlüsse

Anschluss	Funktion
X1	Einspeisen der Spannung
X2	Weiterführen der Spannung zum nächsten Teilnehmer
V1	Systemspannung: Versorgungsspannung 1 (inkl. Elektronikversorgung)
V2	Lastspannung: Versorgungsspannung 2

HINWEIS

Die Systemspannung (V1) und die Lastspannung (V2) werden separat eingespeist und überwacht. Bei einer Unterschreitung der zulässigen Spannung werden die Steckplätze gemäß Versorgungskonzept des Modultyps abgeschaltet. Bei einer Unterschreitung von V2 wechselt die LED PWR von Grün auf Rot. Bei einer Unterschreitung von V1 erlischt die LED PWR.

6.3 RFID-Schreib-Lese-Köpfe anschließen

Zum Anschluss von RFID-Schreib-Lese-Köpfen verfügt das Gerät über vier 5-polige M12-Steckverbinder. Das max. Anzugsdrehmoment beträgt 0,8 Nm.

▶ Schreib-Lese-Köpfe gemäß unten stehender Pinbelegung an das Gerät anschließen.

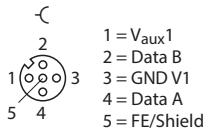

Abb. 12: RS485 – Pinbelegung Schreib-Lese-Kopf-Anschlüsse

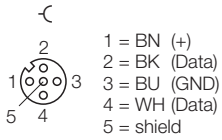

Abb. 13: Verbindungsleitungen …/S2500 – Pinbelegung Schreib-Lese-Kopf-Anschlüsse

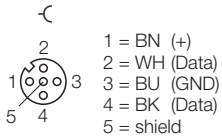

Abb. 14: Verbindungsleitungen …/S2501 – Pinbelegung Schreib-Lese-Kopf-Anschlüsse

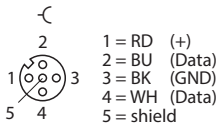

Abb. 15: Verbindungsleitungen …/S2503 – Pinbelegung Schreib-Lese-Kopf-Anschlüsse

6.4 Digitale Sensoren und Aktuatoren anschließen

Zum Anschluss von digitalen Sensoren und Aktuatoren verfügt das Gerät über vier 5-polige M12-Steckverbinder. Das max. Anzugsdrehmoment beträgt 0,8 Nm.

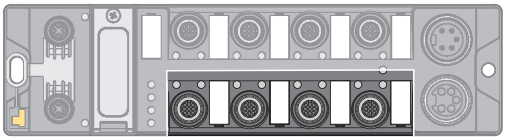

Abb. 16: M12-Steckverbinder zum Anschluss von digitalen Sensoren und Aktuatoren

- ▶ Sensoren und Aktuatoren gemäß unten stehender Pinbelegung an das Gerät anschließen.

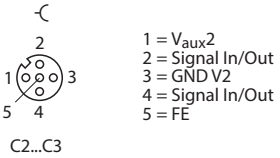

Abb. 17: Anschlüsse für digitale Sensoren und Aktuatoren – Pinbelegung

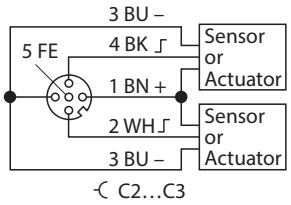

Abb. 18: Anschlüsse für digitale Sensoren und Aktuatoren – Anschlussbild

Die Kanäle sind den Steckplätzen wie folgt zugeordnet:

Kanal	Steckplatz	Pin
DXP8 (Ch8)	C4	4
DXP9 (Ch9)	C4	2
DXP10 (Ch10)	C5	4
DXP11 (Ch11)	C5	2
DXP12 (Ch12)	C6	4
DXP13 (Ch13)	C6	2
DXP14 (Ch14)	C7	4
DXP15 (Ch15)	C7	2

7 In Betrieb nehmen

Über das Betriebssystem Linux können mit C, C++, NodeJS und Python die Gerätefunktionen programmiert werden.
Um über die Konsole auf das Gerät zugreifen zu können, sind zusätzliche Software-Tools erforderlich (z. B. PuTTY). Ein Dateiaustausch zwischen dem Gerät und einem PC kann z. B. WinSCP genutzt werden. Per Default sind die folgenden Login-Daten auf dem Gerät hinterlegt:
User: user
Passwort: password

HINWEIS
Im Auslieferungszustand ist das Schreib-Lese-Kopf-Protokoll nicht implementiert. Das Protokoll muss durch den Anwender implementiert werden.

7.1 IP-Adresse einstellen

Die IP-Adresse lässt sich über zwei dezimale Drehcodierschalter und DIP-Schalter am Gerät, über den Webserver oder über das Turck Service Tool einstellen.

7.1.1 IP-Adresse über Schalter am Gerät einstellen

Die IP-Adresse kann über zwei dezimale Drehcodierschalter und den DIP-Schalter „Mode" am Gerät eingestellt werden. Die Schalter befinden sich gemeinsam mit den USB-Ports und dem SET-Taster unter einer Abdeckung.

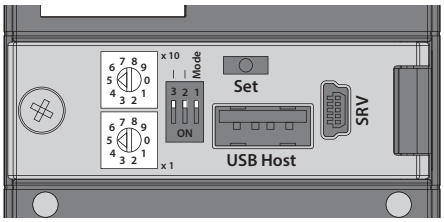

Abb. 19: Schalter zum Einstellen der IP-Adresse

- Abdeckung über den Schaltern öffnen.
- Drehcodierschalter gemäß unten stehender Tabelle auf die gewünschte Position einstellen.
- DIP-Schalter „Mode" gemäß unten stehender Tabelle auf die gewünschte Position einstellen.
- Spannungsreset durchführen.
- ACHTUNG! Bei geöffneter Abdeckung über den Drehcodierschaltern ist die Schutzart IP67 oder IP69K nicht gewährleistet. Geräteschäden durch eindringende Fremdkörper oder Flüssigkeiten sind möglich. Abdeckung über den Schaltern fest verschließen.

Adressierungsmöglichkeiten

Die IP-Adresse der Geräte lässt sich auf unterschiedliche Weise einstellen. Folgende Adressierungsmöglichkeiten können über die Schalter am Gerät ausgewählt werden. Änderungen der Einstellung werden nach einem Spannungsreset aktiv.

Einstellmöglichkeit	DIP-Schalter „MODE"	Drehcodierschalter	Beschreibung
Default-Adresse	0	00	IP-Adresse: 192.168.1.100 Subnetzmaske: 255.255.255.0 Gateway: 192.168.1.1
Rotary-Modus	0	1…99	Im Rotary-Modus kann das letzte Byte der IP-Adresse manuell am Gateway eingestellt werden. Die weiteren Netzwerkeinstellungen sind nichtflüchtig im Speicher des Gateways hinterlegt und können im Rotary-Modus nicht verändert werden. Einstellbar sind Adressen von 1…99.
DHCP-Modus	1	40	Im DHCP-Modus wird die IP-Adresse automatisch von einem DHCP-Server im Netzwerk vergeben. Die vom DHCP-Server zugewiesene Subnetzmaske und die Default-Gateway-Adresse werden nichtflüchtig im Speicher des Gateways hinterlegt. DHCP unterstützt 3 Arten der IP-Adresszuweisung: ■ Automatische Adressvergabe: Der DHCP-Server vergibt eine permanente IP-Adresse an den Client. ■ Dynamische Adressvergabe: Die vom Server vergebene IP-Adresse ist immer nur für einen bestimmten Zeitraum reserviert. Nach Ablauf dieser Zeit oder nach der expliziten Freigabe durch einen Client wird die IP-Adresse neu vergeben. ■ Manuelle Adressvergabe: Ein Netzwerk-Administrator weist dem Client eine IP-Adresse zu. DHCP wird in diesem Fall nur zur Übermittlung der zugewiesenen IP-Adresse an den Client genutzt.
PGM-Modus	1	50	Im PGM-Modus wird die vollständige IP-Adresse manuell über das Turck Service Tool, FDT/DTM oder über einen Webserver vergeben. Im PGM-Modus werden die eingestellte IP-Adresse und die Subnetzmaske im Speicher des Gateways hinterlegt. Alle Netzwerk-Einstellungen (IP-Adresse, Subnetzmaske, Default-Gateway) werden vom internen EEPROM des Moduls übernommen.
PGM-DHCP-Modus	1	60	Im PGM-DHCP-Modus sendet das Gateway so lange DHCP-Requests, bis ihm eine feste IP-Adresse zugewiesen wird. Der DHCP-Client wird automatisch deaktiviert, wenn dem Gateway über den DTM oder einen Webserver eine IP-Adresse zugewiesen wird.
F_Reset	1	90	Der F_Reset-Modus setzt alle Einstellungen des Geräts auf die Default-Werte zurück und löscht alle Daten im internen Flash des Geräts. Die folgenden Werte werden zurückgesetzt bzw. gelöscht: ■ IP-Adresse und Subnetzmaske ■ Parameter
Restore	1	00	■ IP-Adresse: 192.168.1.100

7.1.2 IP-Adresse über das Turck Service Tool einstellen

- Gerät über die Ethernet-Schnittstelle mit einem PC verbinden.
- Turck Service Tool öffnen.
- **Suchen** klicken oder [F5] drücken.

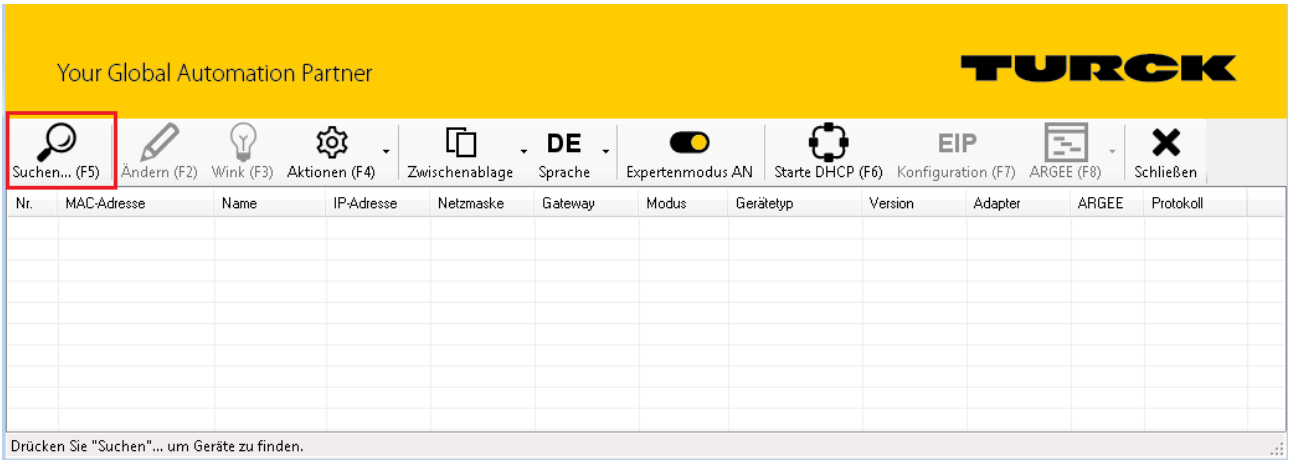

Abb. 20: Turck Service Tool – Startbildschirm

Das Turck Service Tool zeigt die angeschlossenenen Geräte an.

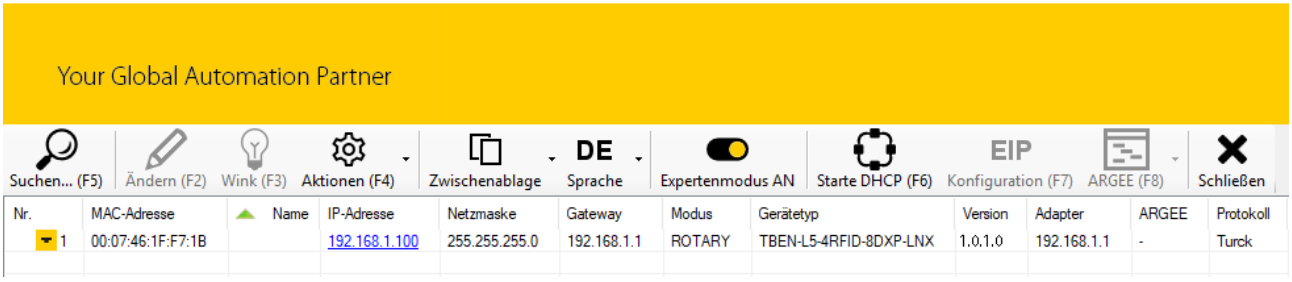

Abb. 21: Turck Service Tool – Gefundene Geräte

- Gewünschtes Gerät anklicken.
- **Ändern** klicken oder [F2] drücken.

HINWEIS
Ein Klick auf die IP-Adresse des Geräts öffnet den Webserver.

- IP-Adresse sowie ggf. Netzwerkmaske und Gateway ändern.
- Änderungen mit einem Klick auf **Im Gerät setzen** übernehmen.

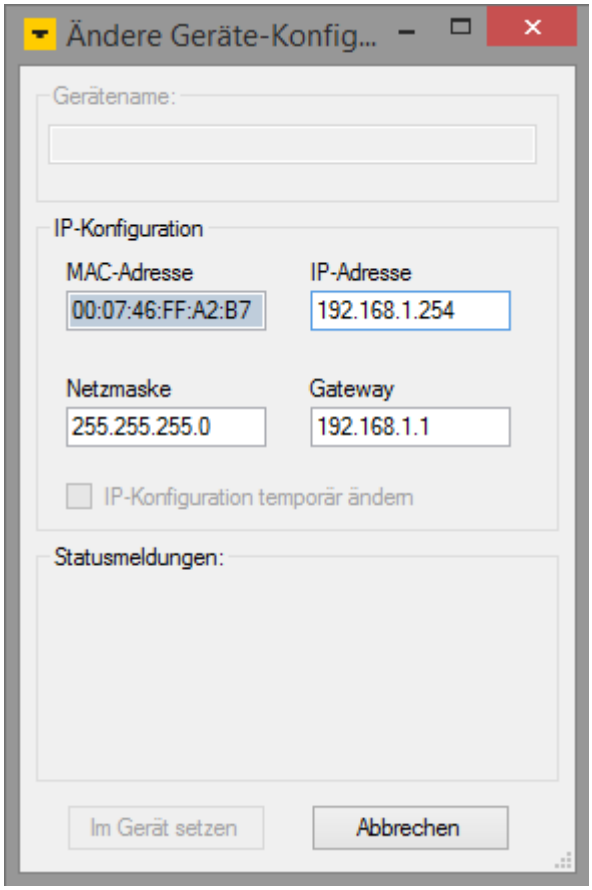

Abb. 22: Turck Service Tool – Geräte-Konfiguration ändern

7.2 RFID-Kanäle programmieren

Die RFID-Kanäle des Geräts sind als serielle RS485-Schnittstellen ausgeführt. Die Schnittstelle arbeitet im Halbduplex-Prinzip. Die Richtung des Kontrollsignals muss angepasst werden, um zwischen Senden und Empfangen umzustellen.

Zur Kommunikation mit angeschlossenen Geräten ist ein externes Peripheriegerät in das RFID-Interface integriert. Ein separater Slave-Controller dient zum Handling der zwischen Interface und Schreib-Lese-Kopf ausgetauschten Meldungen. Der Slave-Controller muss über ein Skript an die Linux-Einstellungen angepasst werden.

Die seriellen Schnittstellen sind wie folgt aufgebaut:

COM-Schnittstelle	TTY	Kanal
COM0	tty03	Ident 0
COM1	tty04	Ident 1
COM2	tty01	Ident 2
COM3	tty02	Ident 3

7.2.1 GPIOs der RFID-Kanäle – Übersicht

Die RFID-Kanäle können über die GPIOs programmiert werden. Die GPIOs sind unter folgendem Pfad verfügbar:
/sys/class/gpio/…

HINWEIS
Wenn Biasing eingeschaltet ist, muss auch die Polarität eingestellt ist. Dazu die GPIOs für den Polaritätswechsel nutzen.
Polaritätswechsel und Biasing müssen in den Linux-Einstellungen zugelassen werden.

Funktion	COM	GPIO	mögliche Werte
Power/AUX schalten	COM0	gpio493	0: ausschalten 1: einschalten
	COM1	gpio440	
	COM2	gpio441	
	COM3	gpio442	
Biasing A-GND_B-5V_1_1 invertieren	COM0_I	gpio461	0: ausschalten 1: einschalten
	COM1_I	gpio469	
	COM2_I	gpio450	
	COM3_I	gpio453	
Biasing A-5V_B-GND_1_1 in Normalmodus setzen	COM0_N	gpio462	0: ausschalten 1: einschalten
	COM1_N	gpio470	
	COM2_N	gpio449	
	COM3_N	gpio454	
Polaritätswechsel (RxD → TxD)	COM0	gpio456	0: normal 1: invertiert
	COM1	gpio464	
	COM2	gpio447	
	COM3	gpio451	
RS485-Busabschluss einschalten	COM0	gpio460	0: ausschalten 1: einschalten
	COM1	gpio468	
	COM2	gpio448	
	COM3	gpio452	

7.2.2 Slave-Controller über Skript anpassen

Zum Anpassen des Slave-Controllers ist auf dem Gerät ein Skript installiert. Das Skript ist unter folgendem Pfad verfügbar:
/TURCK/scripts/serial.sh

Das Skript kann mit der folgenden Syntax genutzt werden:

```
sh serial.sh device cmd [param]
```

Beispiel für die Nutzung des Skripts:

```
sh serial.sh COM0 send "Hello World!"
```

Die folgenden Parameterwerte können eingesetzt werden:

cmd	param
baud	Übertragungsrate, z. B. 9600
term	ON/OFF
bias	ON/OFF
swap	ON/OFF
send	Meldung
recv	
vaux	ON/OFF
sethw	
sets	

Skript sh serial.sh – Übersicht der Befehle

Befehl	Funktion
sh serial.sh COMx baud baudrate	Setzt die Übertragungsrate des Linux-Systems und des Slave-Controllers.
sh serial.sh COMx term ON/OFF	Schaltet den RS485-Busabschluss ein oder aus.
sh serial.sh COMx bias ON/OFF	Schaltet das Biasing ein oder aus.
sh serial.sh COMx swap ON/OFF	Schaltet die Swapping-Funktion ein oder aus.
sh serial.sh COMx send "message"	Sendet einen Datenstring an einen angeschlossenen Schreib-Lese-Kopf.
sh serial.sh COMx recv	Empfängt Meldungen eines angeschlossenen Schreib-Lese-Kopfs.
sh serial.sh COMx vaux ON/OFF	Schaltet die Hilfsspannung VAUX ein oder aus.
sh serial.sh COMx sethw	Slave-Controller an Linux-Einstellungen anpassen
sh serial.sh COMx seths [baudrate]k[databits][parity][stopbits]	Einstellungen für Slave-Controller und Linux-System anpassen

Beispiel: Einstellungen für Slave-Controller und Linux-System über Skript anpassen

Die Einstellungen für den Slave-Controller und das Linux-System können über die sets-Funktion des Skripts angepasst werden. Die Funktion kann mit folgender Syntax genutzt werden:

```
sh serial.sh COMx sets (Baudrate)k(databits)(parity)(stopbits)
```

Beispiel: Übertragungsrate von 115200 kbaud, 8 Bit, ohne Parität, 1 Stoppbit einstellen

```
serial.sh COM1 115.200k8n1
```

Die möglichen Werte für die einzelnen Parameter entnehmen Sie der folgenden Tabelle:

Übertragungsrate		**Bits**		**Parität**		**Stoppbit**	
Parameter	**Bedeutung**	**Parameter**	**Bedeutung**	**Parameter**	**Bedeutung**	**Parameter**	**Bedeutung**
9600	9600 kBaud	5	cs5	n	keine	1	1 Stoppbit
38400	38400 kBaud	6	cs6	e	gerade	2	2 Stoppbits
115200	115200 kBaud	7	cs7	o	ungerade		
		8	cs8				

7.2.3 RFID-Kanäle mit Python 3 programmieren

Die folgenden Beispiele zeigen die Programmierung der RFID-Schnittstelle mit Python 3.

Beispiel 1: Modul „pySerial“ verwenden

```
import serial # from module pySerial
from os import system as sh # for use of the sh-script

# open serial interface on port 0 and set a timout of 8 seconds
seri = serial.Serial("/dev/COM0", timeout=8)

# change settings
seri.baudrate = 115200 # set the baudrate of port COM0 to 115200
seri.parity ='N' # set no parity for port COM0
seri.bytesize = 7 # set the byte size for a sign to 7 for port
COM0
seri.stopbits = 1 # set stopbits to 1 for port COM0

sh('/TURCK/scripts/serial.sh COM0 sethw')

seri.write(bytearray.fromhex("aa 07 07 49 00 41 23")) # writes a
bytestream

print(seri.readline()) # reads incoming message as ascii
```

Beispiel 2: Modul „periphery“ verwenden

```
from periphery import Serial
from os import system as sh # for use of the sh-script

# Open /dev/COM1 with baudrate 115200, and defaults of 8N1, no
flow control
serial = Serial("/dev/COM1", 115200)

# write a bytestream serial.write(bytearray.fromhex("aa 07 07 49
00 41 23"))

# Read up to 128 bytes with 500ms timeout
buf = serial.read(128, 0.5)

print(buf)
print("read %d bytes: _%s_" % (len(buf), buf))

serial.close()
```

7.2.4 RFID-Kanäle über Node.js programmieren

Die folgenden Beispiele zeigen die Programmierung der RFID-Schnittstelle mit Node.js. Weitere Informationen zu Node.js und den Node.js-Packages finden Sie unter:

- https://nodejs.org
- https://www.npmjs.com/

```
# initialize the serialport-v5 box
var SerialPort=require('serialport-v5');

# initialize the shelljs box
var shell = require('shelljs');

# initialize COM1
com1 = new SerialPort('/dev/COM1');

# adjust hardware to the System settings
shell.exec('sh /TURCK/scripts/serial.sh COM0 sethw');

# read up to 6 bytes
com1.on('readable', function () {
console.log('\nincomming Data com1:', com1.read(6));
});

# write buffer to COM1
const buf1 = new Buffer([0x01, 0x02, 0x03, 0x04, 0x05, 0x11]);
com1.write(buf1, console.log('message written from com1: ' +
buf1.toString('hex')));

# read line as ascii

const readline = SerialPort.parsers.Readline;

const parser = new readline();

com1.pipe(parser);

parser.on('data', console.log);

# write ascii com2.write('Let us talk together...\n');
```

7.2.5 RFID-Kanäle über C oder C++ programmieren

Die folgenden Beispiele zeigen die Programmierung der RFID-Schnittstelle mit Ansi C/C++.

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>

// initialize function (use extern for C++)
ssize_t read (int __fd, void *__buf, size_t __nbytes) __wur;
ssize_t write (int __fd, const void *__buf, size_t __n) __wur;
int close (int __fd);

int main(void) {
     //choose Interface for connection
    const char *Path = "/dev/COM2";
    struct termios options;
    int fd, count, i ;
    unsigned char currentBuff[1];
    unsigned char InBuff[255];
    unsigned char *p_InBuff = InBuff;
    unsigned char Message[] = {0x48,0x65,0x6c,0x6c,0x6f};
    if((fd = open((Path), O_RDWR | O_NOCTTY)) != -1)
    {
         // Set serial Interface
        tcgetattr (fd, &options);

        cfsetspeed(&options, B9600);
        cfmakeraw (&options);
        options.c_cflag |= CS8;
        tcsetattr (fd, TCSANOW, &options);

        system("sh /TURCK/scripts/serial.sh COM2 sethw");
        // write to Interface COM2
        if ((write(fd, Message, sizeof(Message))) == -1)
                {
                        printf("not able to write...");
                 }

        // read from Interface COM2
        count = 0;
        do
        {
                 if ((count += read(fd, currentBuff, 1)) == -1)
                        printf("can not read...");
                *p_InBuff = currentBuff[0];
                p_InBuff++;
        }while(currentBuff[0] != 0xfe);

        // print:
         p_InBuff -= count;
        printf("\nData count: %i",count+1);
        printf("\nValues: \n");
         for(i = 0; i <= count ; i++)
        {
```

```
                printf("%.02x ", *p_InBuff ); //"\nEmpfangener
Wert %i: %.02x " p_InBuff++;
        }

        // close the Interface
        if((close(fd)) == -1)
        {
                printf("\n can not close interface");
        }
    }
    else
        printf("can not open interface\n");
     return EXIT_SUCCESS;
}
```

7.3 Digitale Kanäle (DXP) programmieren

7.3.1 GPIOs der DXP-Kanäle – Übersicht

Die digitalen I/O-Kanäle (DXP) können über die GPIOs als Eingänge oder Ausgänge programmiert werden. Die GPIOs sind unter folgendem Pfad verfügbar:
/sys/class/gpio/…

Kanal	Steckplatz	Typ	GPIO	Mögliche Werte
DXP8	C4	Eingang	110	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	12	0: Ausgang aus (0V) 1: Ausgang ein (24V)
DXP9		Eingang	111	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	13	0: Ausgang aus (0V) 1: Ausgang ein (24V)
DXP10	C5	Eingang	112	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	47	0: Ausgang aus (0V) 1: Ausgang ein (24V)
DXP11		Eingang	113	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	63	0: Ausgang aus (0V) 1: Ausgang ein (24V)
DXP12	C6	Eingang	114	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	86	0: Ausgang aus (0V) 1: Ausgang ein (24V)
DXP13		Eingang	116	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	87	0: Ausgang aus (0V) 1: Ausgang ein (24V)
DXP14	C7	Eingang	117	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	88	0: Ausgang aus (0V) 1: Ausgang ein (24V)
DXP15		Eingang	7	0: Eingang aus (0V) 1: Eingang ein (24V)
		Ausgang	89	0: Ausgang aus (0V) 1: Ausgang ein (24V)

Zuschaltbare Spannungsversorgung VAUX einstellen

Steckplatz	Typ	GPIO	Mögliche Werte
C4	Ausgang	495	0: VAUX aus 1: VAUX ein
C5	Ausgang	496	
C6	Ausgang	497	
C7	Ausgang	498	

Zuschaltbare Spannungsversorgung VAUX – Diagnose

Steckplatz	Typ	GPIO	Mögliche Werte
C4	Eingang	499	0: VAUX fehlerfrei 1: Fehler oder Über-spannung an VAUX
C5	Eingang	500	
C6	Eingang	501	
C7	Eingang	502	

7.3.2 DXP-Funktionen über Skript einstellen

Zum Einstellen der DXP-Kanäle ist auf dem Gerät ein Skript installiert. Das Skript ist unter folgendem Pfad verfügbar:
/TURCK/scripts/dxp.sh

Das Skript kann mit der folgenden Syntax genutzt werden:

```
sh dxp.sh DXPx [value]
```

Das folgende Beispiel setzt den Wert für den Kanal DXP8 auf „EIN“.

```
sh dxp.sh DXP8 1
```

Parameter	Mögliche Werte
DXP8…DXP15	1: Kanal einschalten 0: Kanal ausschalten

7.3.3 DXP-Kanäle mit Python 3 programmieren

HINWEIS
Die Geschwindigkeit der Datenübertragung ist abhängig von der konfigurierten Blockgröße und der eingestellten Übertragungsrate. Für zeitkritische Applikationen ist die Geschwindigkeit ggf. nicht ausreichend. Um eine schnellere Datenverarbeitung zu erreichen, kann der Prozess als Realtime-Prozess eingestellt werden.

Das folgende Beispiel zeigt die Programmierung der digitalen I/O-Kanäle mit Python 3.

```
import sys
#GPIOs-> OUT: IN:
ports = ["47","112"]

# write GPIO:
try:
    # set direction to write DXP
    fo = open("/sys/class/gpio/gpio" + ports[0] +"/direction",
"w")
    fo.write("out")
    fo.close()
    # write DXP:
    f = open("/sys/class/gpio/gpio" + ports[0] +"/value", "w")
    f.write("1")
    f.close()
except:
    # export gpio if not done as yet
    f1 = open("/sys/class/gpio/export", "w")
    f1.write(ports[0]) f1.close()
    # set direction to write DXP
    fo = open("/sys/class/gpio/gpio" + ports[0] +"/direction",
"w")
    fo.write("out")
    fo.close()
    # write DXP:
    fw = open("/sys/class/gpio/gpio" + ports[0] +"/value", "w")
    fw.write("1")
    fw.close()

# read GPIO:
try:
    # set direction to read DXP
    fo = open("/sys/class/gpio/gpio" + ports[1] +"/direction",
"w")
    fo.write("in")
    fo.close()
    # set active low to get the right value...
    fal = open("/sys/class/gpio/gpio" + ports[1] +"/active_low",
"w")
    fal.write("1")
    fal.close()
    # read DXP: fr = open("/sys/class/gpio/gpio" + ports[1] +"/va-
lue", "r")
    val=fr.read()
    fr.close()
```

```
    print(val)
except:
    # export gpio if not done as yet
    f1 = open("/sys/class/gpio/export", "w")
    f1.write(ports[1])
    f1.close()
    # set direction to read DXP
    fo = open("/sys/class/gpio/gpio" + ports[1] +"/direction",
"w")
    fo.write("in")
    fo.close()
    # set active low to get the right value...
    fal = open("/sys/class/gpio/gpio" + ports[1] +"/active_low",
"w")
    fal.write("1")
    fal.close()
    # read DXP:
    fr = open("/sys/class/gpio/gpio" + ports[1] +"/value", "r")
    val=fr.read()
    fr.close()
    print(val)
```

7.3.4 DXP-Kanäle über Node.js programmieren

HINWEIS
Die Geschwindigkeit der Datenübertragung ist abhängig von der konfigurierten Blockgröße und der eingestellten Übertragungsrate. Für zeitkritische Applikationen ist die Geschwindigkeit ggf. nicht ausreichend. Um eine schnellere Datenverarbeitung zu erreichen, kann der Prozess als Realtime-Prozess eingestellt werden.

Die folgenden Beispiele zeigen die Programmierung der digitalen I/O-Kanäle mit Node.js. Weitere Informationen zu Node.js und den Node.js-Packages finden Sie unter:
- **https://nodejs.org**
- **https://www.npmjs.com/**

```
// initialize the onoff box
const Gpio = require('onoff').Gpio;

function setGpioByInt(OUT, val) {
// switch from DXP to GPIO...
switch (OUT) {
    case 8:
        res = 12;
        break;
    case 9:
        res = 13;
        break;
    case 10:
        res = 47;
        break;
    case 11:
        res = 63;
        break;
    case 12:
        res = 86;
        break;
    case 13:
        res = 87;
        break;
    case 14:
        res = 88;
        break;
    case 15:
        res = 89;
}
        // initialize the GPIO just to write...
        const DXP_Write = new Gpio(res, "out");
        // write the GPIO / DXP...
        DXP_Write.writeSync(val);
        console.log('set Gpio '+ res + ' to ' + val);
}

function getGpio(IN) {
// switch from DXP to GPIO...
switch (IN) {
     case "8":
        res = 110;
```

```
            break;
        case "9":
            res = 111;
            break;
        case "10":
            res = 112;
            break;
        case "11":
            res = 113;
            break;
        case "12":
            res = 114;
            break;
        case "13":
            res = 116;
            break;
        case "14":
            res = 117;
            break;
        case "15":
            res = 7;
}
    // initialize the GPIO just to read...
    const DXP_Read = new Gpio(res, "in");
    // set active low to get the right value...
    DXP_Read.setActiveLow('true');
    // read the GPIO / DXP...
    var res = DXP_Read.readSync();
    console.log('Gpio '+ r_Pin + ' is: ' + res);
    return res;
}
```

7.3.5 DXP-Kanäle über C oder C++ programmieren

Das folgende Beispiel zeigt die Programmierung der digitalen I/O-Kanäle mit Ansi C/C++.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

// initialize function (use extern for C++)
int access(const char *pfad, int modus);

int main(void) {
     //choose DXP / GPIO for connection

    char GPIO_IN_FILE[] = "/sys/class/gpio/gpio114";
    char GPIO_OUT_FILE[] = "/sys/class/gpio/gpio86";
    char input[2]; FILE *fh;
/*
    ================================================================
==============
    READ:

================================================================
==========*/
    if( access( GPIO_IN_FILE, F_OK ) == -1 )
    {
        // file doesn't exist!
            // export gpio...
            if((fh = fopen("/sys/class/gpio/export", "w")) != 0)
            {
                fputs("114",fh);
                fclose(fh);
            }
            else
            {
                printf("failed on export to read...\n");
                printf("result: %i \n", (int)fh);
                return -1;
            }
    }
    // set direction to read...
    if((fh =fopen("/sys/class/gpio/gpio114/direction", "w")) != 0)
    {
        fputs("in",fh);
        fclose(fh);
    }
    else
    {
         printf("failed on setting direction to read...\n");
        return -1;
    }

    // set active low to read...
    if((fh = fopen("/sys/class/gpio/gpio114/active_low", "w")) !=
0)
    {
```

```
        fputs("1",fh);
        fclose(fh);
    }
    else
    {
        printf("failed on setting active low ...\n");
        return -1;
    }
    // read GPIO...
    if((fh = fopen("/sys/class/gpio/gpio114/value", "r")) != 0)
    {
        fgets(input,2,fh);
        fclose(fh);
        printf("Value: %c\n", input[0]);
    }
     else
    {
        printf("failed on reading ...\n");
        return -1;
    }
/*

==========================================================================
==========
    WRITE:
    ==========================================================================
==============*/
    if( access( GPIO_OUT_FILE, F_OK ) == -1 )
    {
        // file doesn't exist
            // export gpio...
            if((fh = fopen("/sys/class/gpio/export", "w")) != 0)
            {
                fputs("86",fh);
                fclose(fh);
            }
            else
            {
                 printf("failed on export to write...\n");
                printf("result: %i \n", (int)fh);
                return -1;
        }
    }
    // set direction to read...
    if((fh = fopen("/sys/class/gpio/gpio86/direction", "w")) != 0)
    {
            fputs("out",fh);
            fclose(fh);
    }
    else
    {
         printf("failed on setting direction to write...\n");
        return -1;
    }
```

```
    // write GPIO...
    if((fh = fopen("/sys/class/gpio/gpio86/value", "w")) != 0)
    {
        fputs((const char*)"1",fh);
        fclose(fh);
    }
     else
    {
        printf("failed on writing ...\n");
        return -1;
    }
    return EXIT_SUCCESS;
}
```

7.4 LED-Funktionen programmieren

7.4.1 LEDs – Übersicht

Das Gerät verfügt über drei frei programmierbare LEDs. Über Lese- und Schreibbefehle können die LEDs einzeln programmiert werden. Die LEDs sind auf dem System unter dem folgendem Pfad gemappt: „/sys/class/leds/...“

LED	Farbe	Systemname
APPL	grün	appl_green
	rot	appl_red
ERR	grün	err_green
	rot	err_red
RUN	grün	run_green
	rot	run_red

Wenn die Farben rot und grün einer LED gleichzeitig eingeschaltet sind, leuchtet die LED orange.

HINWEIS
Bei einem laufenden Firmware-Update wird die RUN-LED vom System genutzt.

7.4.2 LED-Funktionen über Skript einstellen

Zum Einstellen der LEDs ist auf dem Gerät ein Skript installiert. Das Skript ist unter folgendem Pfad verfügbar:
/TURCK/scripts/led.sh

Das Skript kann mit der folgenden Syntax genutzt werden:
```
sh led.h led color [value]
```

Das folgende Beispiel schaltet die rote LED „APPL“ ein.
```
sh led.sh appl red 1
```

LED	Mögliche Farbeinstellung	Mögliche Values
ERR	green/red	1: LED einschalten 0: LED ausschalten
RUN	green/red	1: LED einschalten 0: LED ausschalten
APPL	green/red	1: LED einschalten 0: LED ausschalten

7.4.3 LED-Funktionen mit Python 3 programmieren

Das folgende Beispiel zeigt die Programmierung der LED-Funktionen mit Python 3:

```
import sys
import time
# write red LEDs:
fw = open("/sys/class/leds/run_red/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/appl_red/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/err_red/brightness", "w")
fw.write("1")
fw.close()

# Wait for 5 seconds
time.sleep(5)

# write green LEDs:
fw = open("/sys/class/leds/appl_green/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/err_green/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/run_green/brightness", "w")
fw.write("1")
fw.close()

# Wait for 5 seconds
time.sleep(5)

# clean red LEDs:
fw = open("/sys/class/leds/run_red/brightness", "w")
fw.write("0")
fw.close()

fw = open("/sys/class/leds/appl_red/brightness", "w")
fw.write("0")
fw.close()

fw = open("/sys/class/leds/err_red/brightness", "w")
fw.write("0")
fw.close()

# Wait for 5 seconds
time.sleep(5)

# clean green LEDs:
fw = open("/sys/class/leds/appl_green/brightness", "w")
```

```
fw.write("0")
fw.close()

fw = open("/sys/class/leds/err_green/brightness", "w")
fw.write("0")
fw.close()

fw = open("/sys/class/leds/run_green/brightness", "w")
fw.write("0")
fw.close()
```

7.4.4 LED-Funktionen über Node.js programmieren

Die folgenden Beispiele zeigen die Programmierung der LED-Funktionen mit Node.js. Weitere Informationen zu Node.js und den Node.js-Packages finden Sie unter:

- **https://nodejs.org**
- **https://www.npmjs.com/**

```
// initialize the onoff box
const Gpio = require('onoff').Gpio;

//initialize the leds which are free for the user

appl_green_led = new LED('appl_green');
appl_red_led = new LED('appl_red');
error_green_led = new LED('err_green');
error_red_led = new LED('err_red');
run_green_led = new LED('run_green');
run_red_led = new LED('run_red');
```

7.4.5 LED-Funktionen über C oder C++ programmieren

Das folgende Beispiel zeigt die Programmierung der LED-Funktionen mit Ansi C/C++.

```
#include <stdio.h>
#include <stdlib.h>

// initialize function (use extern for C++)
char* strcpy(char* Ziel, const char* Quelle);
char* strcat(char* s1, const char* s2);

int main(void) {
   // LEDs for the customer:
   char *appl_green_led = "appl_green";
   char *appl_red_led = "appl_red";
   char *error_green_led = "err_green";
   char *error_red_led = "err_red";
   char *run_green_led = "run_green";
   char *run_red_led = "run_red";
   char *LED_FILE = "/sys/class/leds/";
   char *brightness = "/brightness";
   FILE *fh;

   char cur_Str[50] = {0};
   strcpy(cur_Str, LED_FILE);
   // take LED which will shine:
   strcat(cur_Str, run_red_led);
   strcat(cur_Str, brightness);

   //WRITE:
   printf("string to led: %s\n", cur_Str);

   // write LED...
   if((fh = fopen(cur_Str, "w")) != 0)
   {
      // write "1" to switch on and "0" to switch of LED
      fputs((const char*)"0",fh);
      fclose(fh);
   }
   else
   {
      printf("failed on writing ...\n");
      return -1;
   }
   return EXIT_SUCCESS;
}
```

7.5 C-Applikation erstellen

Voraussetzungen

Um eine C-Applikation erstellen zu können, sind folgende Komponenten erforderlich:
- Toolchain für Cortex A8
- C-Programm

Toolchain herunterladen

Um ein C-Programm cross-kompilieren zu können, benötigen Sie die folgende Toolchain für den Cortex A8-Prozessor:
- OSELAS.toolchain-2014.12.0-arm-cortexa8-linux-gnueabihf-gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized

Die Toolchain ist unter **http://debian.pengutronix.de/debian** zum Download erhältlich.

Beispiel: C-Programm erstellen

▶ Datei „hello.c“ erstellen.

▶ Folgenden Text in die Datei kopieren:

```
// hello.c
#include <stdio.h>

int main() {
   printf("Hello World!\n");
   return 0;
}
```

▶ Ausführbare Datei über den folgenden Toolchain-Befehl erstellen:

```
/opt/OSELAS.Toolchain-2014.12.0/arm-cortexa8-linux-gnueabihf/
gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized/bin/
arm-cortexa8-linux-gnueabihf-gcc -o helloExample hello.c
```

Beispiel: C-Programm über ein Makefile erstellen

Das Dienstprogramm „make“ automatisiert das Erstellen von ausführbaren Dateien aus Quellcode. Über „make“ lassen sich C-Programme kompilieren. Dazu wird ein Makefile verwendet, das Regeln zur Erstellung von ausführbaren Dateien enthält.

Das folgende Beispiel zeigt ein einfaches Makefile:

```
all: helloExample

helloExample: hello.o
    /opt/OSELAS.Toolchain-2014.12.0/arm-cortexa8-linux-gnueabihf/
gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized/bin/arm-
cortexa8-linux-gnueabihf-gcc -o helloExample hello.o

hello.o: hello.c
   /opt/OSELAS.Toolchain-2014.12.0/arm-cortexa8-linux-gnueabihf/
gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized/bin/arm-
cortexa8-linux-gnueabihf-gcc -c hello.c

clean:
   rm hello.o helloExample
}
```

- ▶ Makefile erstellen.
- ▶ Makefile im selben Ordner wie die C-Applikation speichern.
- ▶ Makefile mit dem Befehl „make“ ausführen.
- ⇨ Das C-Programm wird installiert.

7.6 Applikation automatisch starten (Autostart)

Eine Applikation kann mit der Autostart-Funktion nach dem Start des RFID-Interfaces automatisch ausgeführt werden. Dazu muss eine Konfigurationsdatei (Unit-Datei) erstellt, in das Gerät geschrieben und aktiviert werden

7.6.1 Autostart – Konfigurationsdatei (Unit-Datei) erstellen

- Unit-Datei mit der Endung „.service“ erstellen.

Beispiel: Die Unit-Datei „.setdxp.service“ startet eine Node.js-Applikation, mit der die DXP-Kanäle bei jedem Neustart des Geräts getriggert werden.

- Über „ExecStart“ die Applikation aufrufen, die bei jedem Neustart des Interfaces aufgerufen werden soll:
  `ExecStart=path_to_programm app/file`

Optional kann ein Parameter übertragen werden. Beispiel: `ExecStart=path_to_programm app/file parameter`

Weitere Informationen zu Unit-Konfigurationsdateien finden Sie unter:
https://www.freedesktop.org/software/systemd/man/systemd.service.html

Beispiel: Autostart einer Applikation mit Parameter-Transfer

```
ExecStart=/usr/bin/node /home/user/ hello_GPIO.js webactive
```

7.6.2 Beispiel: Unit-Datei nutzen

Im folgenden Beispiel wird die Node.js-Datei „hello_GPIO.js“ heruntergeladen und unter „/home/user“ gespeichert:

```
[Unit]
Description= trigger the DXPs
#After=Service_that_must_run_before.service

[Service] Type=simpleExecStart=/usr/bin/node /home/user/ hello_GPIO.js

[Install]
WantedBy=multi-user.target
```

- Beispiel-Datei in „./etc/systemd/system/“ erstellen:
  `sudo touch /etc/systemd/system/setdxp.service`
- Erstellte Datei öffnen:
  `sudo nano /etc/systemd/system/setdxp.service`
- Oben aufgeführten Quelltext in die geöffnete Datei einfügen.

7.6.3 Unit-Datei aktivieren

Nach dem Erstellen muss die Unit-Datei über den systemctl-Befehl aktiviert werden. Zum Aktivieren sind Zugriffsrechte auf das Root-Verzeichnis erforderlich. Die Angabe der Dateiendung .services ist optional und kann entfallen.

▶ Unit-Datei über folgenden Befehl aktivieren:
`sudo systemctl enable setdxp.service`

Der erstellte Symlink lautet:
`/etc/systemd/system/multi-user.target.wants/setdxp.service â /etc/systemd/system/setdxp.service`

Unit-Datei deaktivieren

▶ Unit-Datei über folgenden Befehl deaktivieren:
`sudo systemctl disable setdxp.service`

7.7 Zugriffsrechte verwalten

Das Gerät unterstützt die Standard-Linux-Benutzerverwaltung. Die Zugriffsrechte können über die folgenden Standard-Linux-Tools verwaltet werden:

- adduser
- addgroup
- passwd

Benutzer	Rechte	Passwort
root	Systemadministrator (alle Zugriffsrechte)	turck
user	eingeschränkte Zugriffsrechte und Konsolenrechte	password
sftpuser	Zugriffsrechte, SFTP-Rechte im Verzeichnis /home	password

7.8 Python-Packages installieren

Module, Bibliotheken und andere Software können über das BSP (Board Support Package) mit dem Distributionstool PTXdist konfiguriert und auf das Gerät geladen werden. Wenn Pakete in eine bestehende Firmware integriert werden sollen, müssen sie zuvor mit PTXdist erstellt werden. PTXdist ist unter **https://www.pengutronix.de/de/software/ptxdist.html** zum Download erhältlich.

Auf dem Gerät ist zum Integrieren von Software-Paketen der ipkg-Paketmanager (Itsy Package Management System) installiert. Mit dem ipkg-Paketmanager können auch Python-Module nachträglich installiert werden.

7.8.1 Beispiel: Python-Modul installieren

Das folgende Beispiel erläutert das Vorgehen bei der Installation des Python-Moduls sh. Das Python-Modul wird dabei nachträglich in eine bestehende Firmware integriert.

Voraussetzungen

- PTXdist ist auf dem Linux-Hostsystem installiert.
- Das erforderliche Python-Modul wurde heruntergeladen (Beispiel: **https://amoffat.github.io/sh/**).

Beispiel: Python-Modul sh installieren

Um das Python-Modul sh mit PTXdist erstellen zu können, muss zunächst ein Rule File erstellt werden.

- Rule File mit folgendem Befehl erstellen:
  ```
  $ ptxdist newpackage target
  ```
- Interaktiv-Angaben zum Paket erstellen:
  ```
  Output:
  ptxdist: creating a new 'target' package:
  ptxdist: enter package name...........:
  sh ptxdist: enter version number......:
  1.12.13 ptxdist: enter URL of basedir.:https://github.com/
  amoffat/sh/archive/
  ptxdist: enter suffix.................: tar.gz
  ptxdist: enter package author.........: Your Name <E-Mail>
  ptxdist: enter package section........: Python3


  generating rules/sh.make
  generating rules/sh.in
  ```

Die Files sh.make und sh.in werden automatisch erstellt.

- Wenn bekannt, den MD5-Schlüssel des Pakets als Parameter SH_MD5 im File sh.make eintragen.
- Parameter SH_CONF_TOOL im File sh.make auf das entsprechende Tool einstellen (hier: Python 3).
  ```
  SH_CONF_TOOL :=python3
  ```
- Wenn ein Python-Modul über einen separaten Unterordner verfügt: Unterordner im Zielverzeichnis erstellen (in diesem Beispiel nicht erforderlich):
  ```
  @$(call install_copy, Modul, 0, 0, 0755, $(PYTHON3_SITEPACKA-
  GES)/foldername)
  ```
- Im Bereich #Target-Install den Installationsort des Python-Moduls im Zielsystem angeben (Beispiel: sh-Modul):
  ```
  @for file in `find $(SH_PKGDIR)/usr/lib/python$(PYTHON3_MA-
  JORMINOR)/site-packages \
      ! -type d ! -name "*.py" -printf "%P\n"`; do \
    $(call install_copy, sh, 0, 0, 0644, -, \
      /usr/lib/python$(PYTHON3_MAJORMINOR)/site-packages/$$fi-
  le); \
  done
  ```

Im File sh.in können Abhängigkeiten eingetragen werden. Im folgenden Beispiel muss Python 3 vorhanden sein, um Python-Module installieren zu können. Auf dem Hostsystem muss das Modul „setuptools“ vorhanden sein.

- Abhängigkeiten wie folgt eintragen:
  ```
  ## SECTION=python3
  config PYTHON_SH
    tristate
    select PYTHON3                  # Python 3 must be installed
    select HOST_PYTHON3_SETUPTOOLS # Setuptools must be instal-
  led on the host
    prompt "sh"
    help
      FIXME
  ```

- Kompilieren.

Damit das sh-Modul beim nächsten Build mit erzeugt wird, muss das Modul in „menuconfig“ ausgewählt werden:

- „menuconfig“ über folgenden Befehl öffnen:
  `ptxdist menuconfig`
- Über „Scripting Languages“ → „python3 Extra Modules“ zu den Python 3-Modulen navigieren.
- sh-Modul auswählen.
- Konfiguration speichern.

Abb. 23: PTXdist – „Python 3 Extra Modules“

- ipkg-Pakete über den folgenden Befehl erzeugen:
  `ptxdist go`
- ⇨ Wenn keine Fehler aufgetreten sind, ist das Paket mit dem sh-Modul unter „platform-tben-lx-linux/packages/“ zu finden:
  ```
  $ ls platform-tben-lx-linux/packages/
  ...
  python3_3.5.0_armhf.ipk
  sh_1.12.14_armhf.ipk
  ...
  ```
- ipk-File auf das TBEN-Gerät kopieren (z. B. mit scp):
  ```
  scp ~/turck/TBEN-Lx-4RFID-8DXP-LNX/platform-tben-lx-linux/
  packages/sh_1.12.14_armhf.ipk
  root@Target-IP:/directory/of/your/choice/
  ```
- Auf dem TBEN-Gerät einloggen, um das Paket „sh_1.12.14_armhf.ipk“zu installieren.
- ipkg-Manager aufrufen, um das Python-Modul zu installieren:
  `ipkg -force-depends install sh_1.12.14_armhf.ipk`
- ⇨ Das Modul ist im Python Interpreter verfügbar.

8 Einstellen

Die Schreib-Lese-Köpfe müssen über das Schreib-Lese-Kopf-Protokoll parametriert werden. Dazu muss das Schreib-Lese-Kopf-Protokoll auf dem TBEN-Gerät implementiert sein.

9 Betreiben

9.1 LED-Anzeigen

Die Geräte verfügen über frei programmierbare Mehrfarben-LEDs.

DXP-LEDs (Digitale Kanäle, LEDs DXP0…3)		
LED grün	**LED rot**	**Bedeutung**
aus	aus	kein I/O-Signal vorhanden
leuchtet	aus	I/O-Signal vorhanden
aus	leuchtet	Überlast am Ausgang
blinkt	blinkt	Überlast der Hilfsversorgung

LED APPL	Bedeutung
blinkt weiß	Wink-Kommando aktiv

9.2 Gerät zurücksetzen (Reset)

Das Gerät kann über die Drehcodierschalter und das Turck Service Tool über die F_Reset-Funktion auf die Werkseinstellungen zurückgesetzt werden. Im Fehlerfall lässt sich das Gerät über einen Neustart (Reboot) oder den Reset-Befehl zurücksetzen. Wenn ein Neustart durchgeführt oder das Gerät uber den Reset-Befehl zurückgesetzt wurde, bleiben die Einstellungen erhalten.

10 Störungen beseitigen

Sollte das Gerät nicht wie erwartet funktionieren, überprüfen Sie zunächst, ob Umgebungsstörungen vorliegen. Sind keine umgebungsbedingten Störungen vorhanden, überprüfen Sie die Anschlüsse des Geräts auf Fehler.
Ist kein Fehler vorhanden, liegt eine Gerätestörung vor. In diesem Fall nehmen Sie das Gerät außer Betrieb und ersetzen Sie es durch ein neues Gerät des gleichen Typs.

11 Instand halten

11.1 Firmware-Update über die USB-Schnittstelle durchführen

- ▶ Auf einem USB-Stick den Ordner „FW_UPDATE“ anlegen.
- ▶ Firmware als bin-Datei im Ordner „FW_UPDATE“ ablegen.
- ▶ USB-Stick in das Gerät stecken.
- ⇨ Die RUN-LED blinkt grün mit 0,5 Hz.
- ▶ Innerhalb von 30 Sekunden den Set-Taster für mindestens 3 Sekunden gedrückt halten.
- ⇨ Die RUN-LED blinkt in der Abfolge 3 × grün - Pause (1 Hz) - 3 × grün - Pause (1 Hz) - ….
- ⇨ Die Daten werden in das Gerät geladen.
- ⇨ Wenn die RUN-LED orange mit 1 Hz blinkt, ist das Firmware-Update abgeschlossen.
- ▶ USB-Stick entfernen.
- ▶ Spannungsreset durchführen.
- ⇨ Das Gerät startet neu.

11.2 Firmware-Update über die Konsole durchführen

Das Firmware-Update kann entweder über ein geeignetes Tool (z. B. WinSCP oder FileZilla) oder als „Secure Copy“ auf das Gerät übertragen werden.

- ▶ Update-Datei (z. B. TBEN-Lx-4RFID-8DXP-LNX_V1010.bin) über ein geeignetes Tool (z. B. WinSCP) auf das Gerät laden (siehe „Beispiel: Firmware-Update über WinSCP und PuTTY durchführen).
  Alternativ Update-Datei als „Secure Copy“ auf das Gerät übertragen:
  `scp Path/To/Your/File/root.ubifs user@ip_of_your_board:/run/Update:`
- ▶ Update auf dem Gerät mit dem folgenden Befehl ausführen:
  `sudo update system /path/To/The/Updatefile/updatefile`
- ⇨ Die Firmware wurde erfolgreich installiert, wenn keine Fehlermeldungen erscheinen.

11.2.1 Beispiel: Firmware-Update über WinSCP und PuTTY durchführen

Im folgenden Beispiel wird ein Firmware-Update mit Hilfe der Tools WinSCP und PuTTY durchgeführt.

Voraussetzungen

- WinSCP ist installiert.
- PuTTY ist installiert.
- Die Update-Datei liegt als .bin-Datei auf einem lokalen Rechner vor.

Firmware-Datei mit WinSCP übertragen

▶ In WinSCP mit folgenden Angaben auf dem Gerät anmelden:
Übertragungsprotokoll: FTP
Rechnername: IP-Adresse des Geräts (hier: 192.168.1.100)
Port: 21
Benutzername: root
Passwort: turck

HINWEIS
Alternativ ist auch die Anmeldung über SFTP und Port 22 möglich.

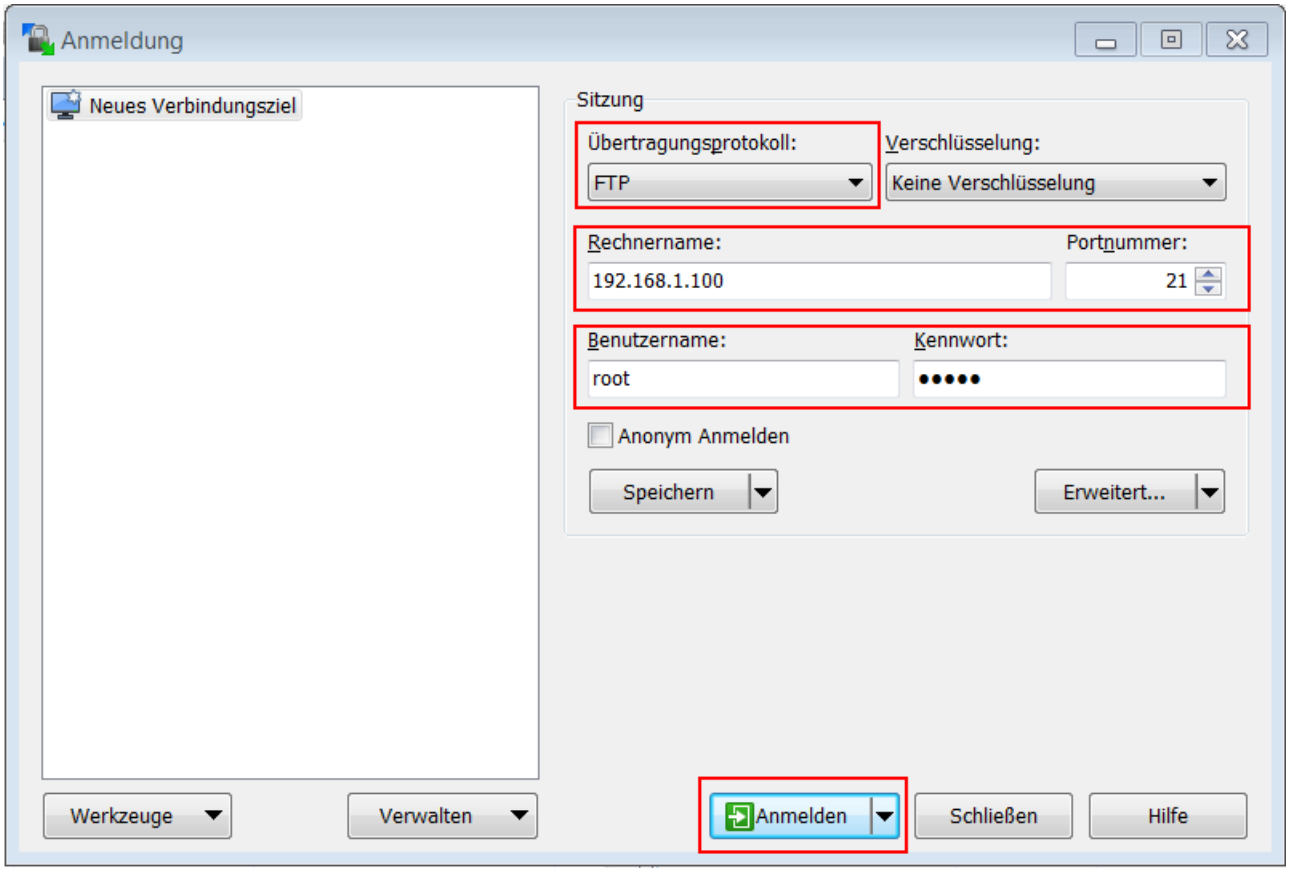

Abb. 24: WinSCP – Login

▶ In WinSCP zum Speicherort der Update-Datei auf dem Host-PC navigieren.

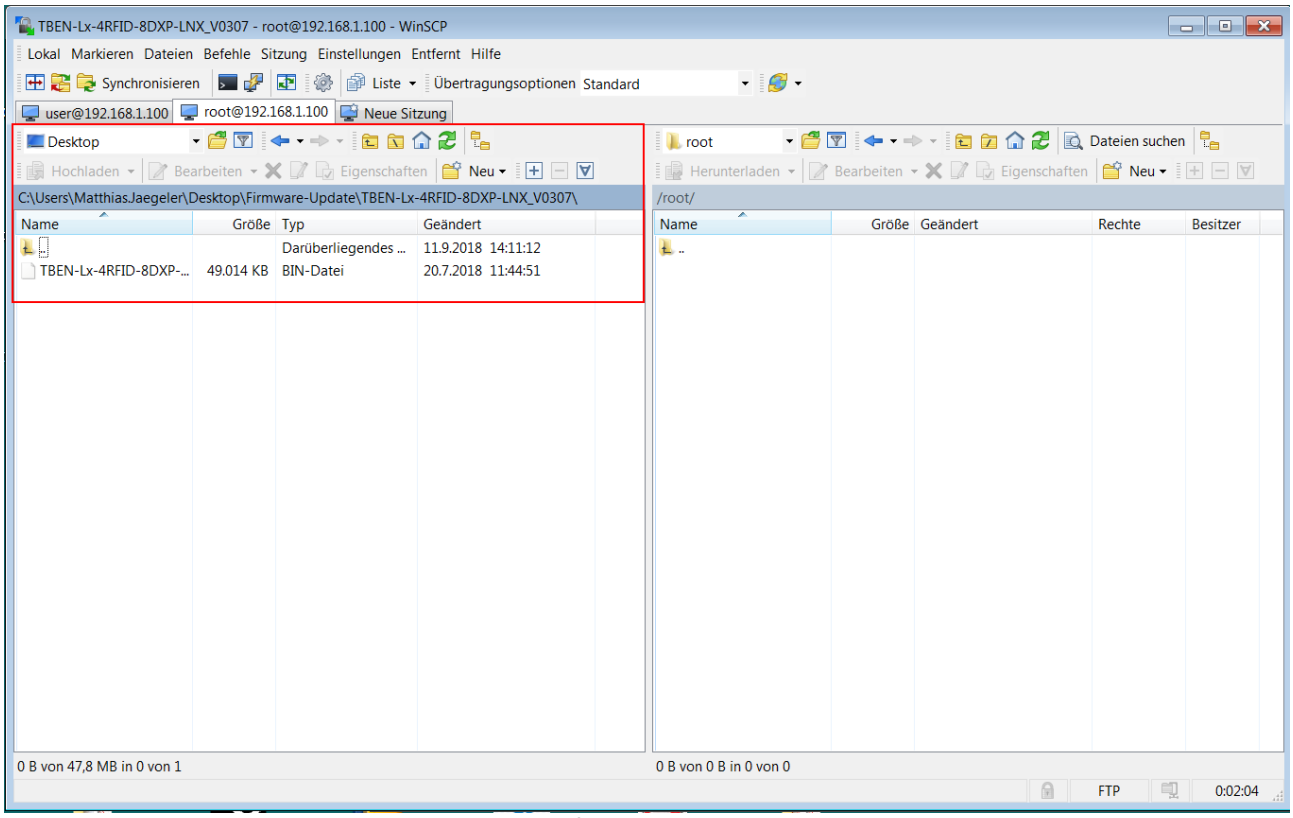

Abb. 25: WinSCP – Speicherort der Update-Datei auf dem Host-PC

▶ Auf dem Gerät zum Run-Verzeichnis navigieren.

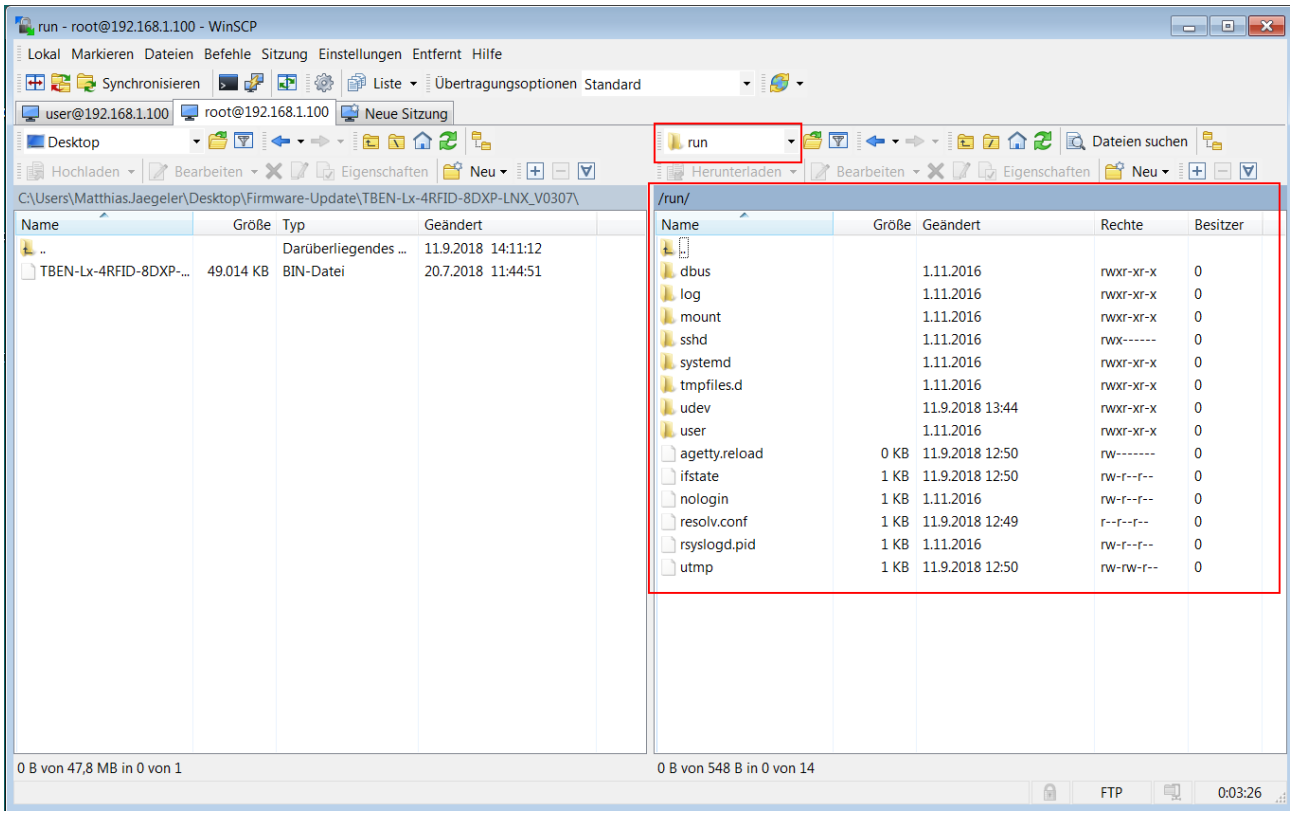

Abb. 26: WinSCP – RUN-Verzeichnis auf dem Gerät

- Update-Datei per Drag-and-Drop oder per Klick auf **Hochladen** im run-Verzeichnis des Geräts ablegen.
- Folgende Abfrage mit **OK** bestätigen.

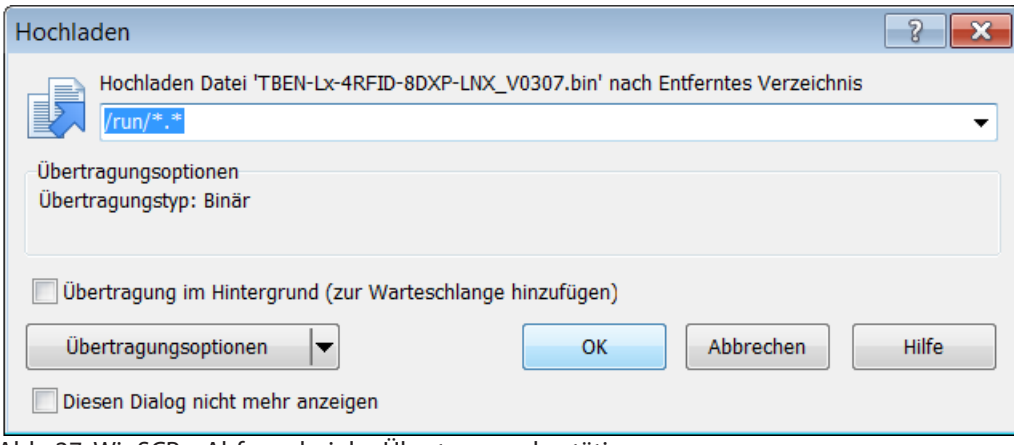

Abb. 27: WinSCP – Abfrage bei der Übertragung bestätigen

Die Übertragung der Update-Datei wird von WinSCP wie folgt angezeigt:

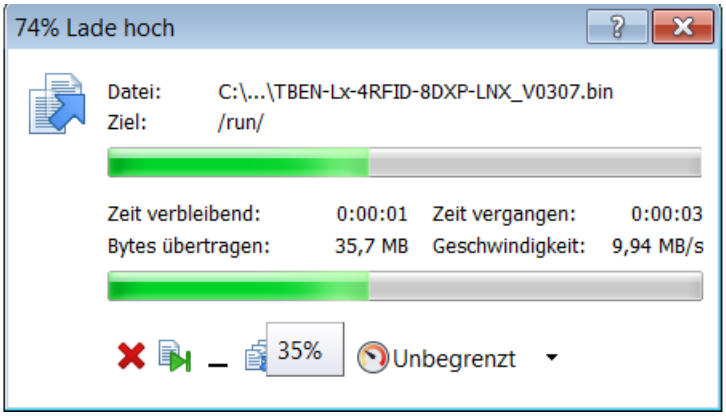

Abb. 28: WinSCP – Datei-Übertragung

Firmware-Update mit PuTTY durchführen

- ▶ PuTTY öffnen.
- ▶ In PuTTY folgende Einstellungen eintragen:

- Host Name: IP-Adresse des Geräts
- Port: 22

- ▶ Optional: Namen für die aktuelle Session vergeben (hier: TBEN-Lx_LNX). Bei späteren Wiederholungen kann die Session über **Load** geladen werden.
- ▶ Bei gespeicherten Sessions: TBEN-Lx_LNX auswählen und mit **Load** bestätigen.
- ▶ **Open** klicken.

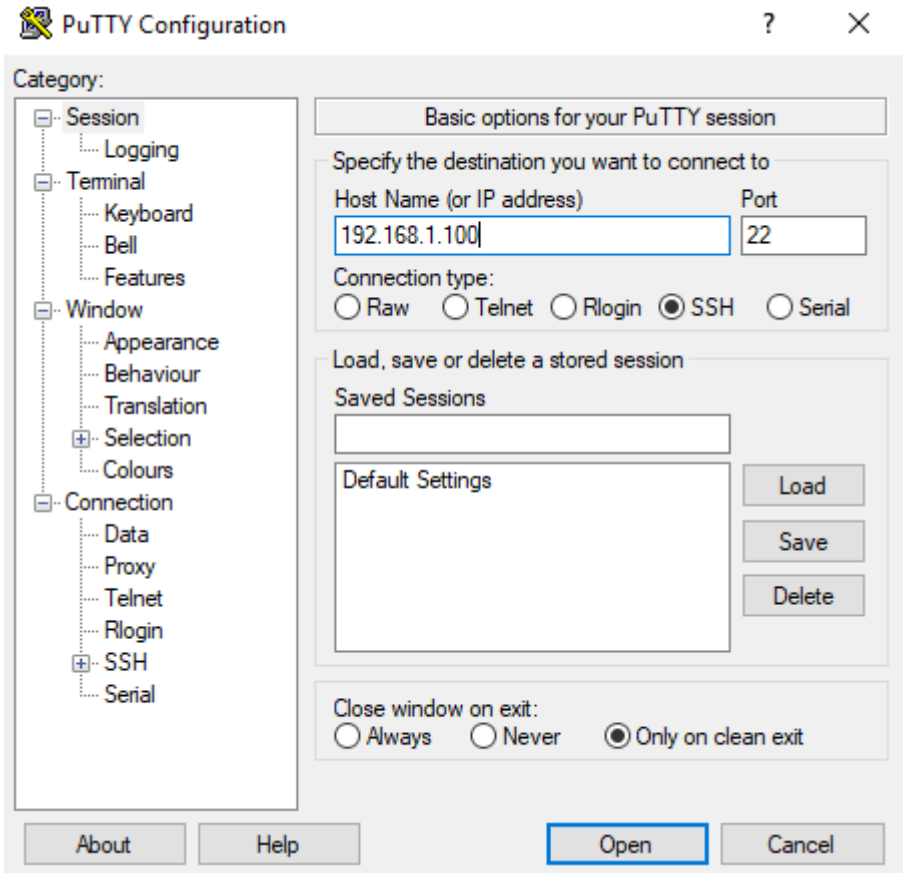

Abb. 29: PuTTY-Konfiguration

- Mit dem Benutzernamen „root“ auf dem Gerät anmelden (Passwort: „turck“). Die Eingabe des Passworts wird in PuTTY nicht angezeigt.
- Update mit dem Befehl `sudo update system /run/[Dateiname].bin` ausführen.
  Beispiel: `sudo update system /run/TBEN-Lx-4RFID-8DXP-LNX_01520038_V1.0.1.0.bin`

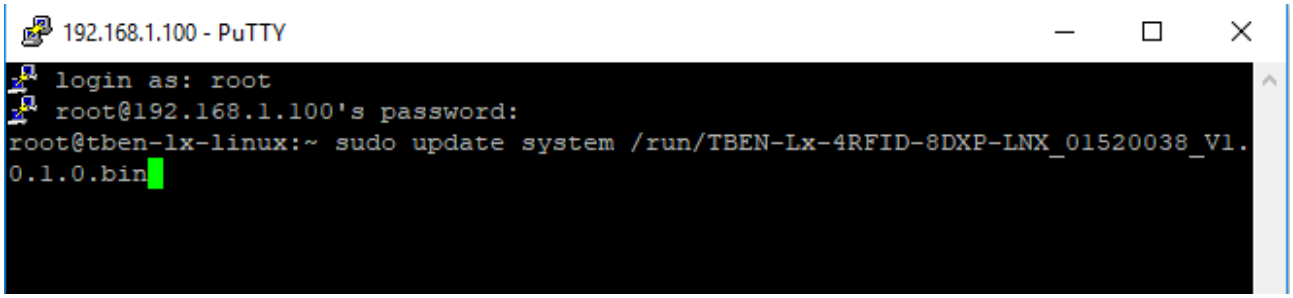

Abb. 30: PuTTY – Firmware-Update starten

- Abwarten, bis die Meldung `updating system finished` angezeigt wird.

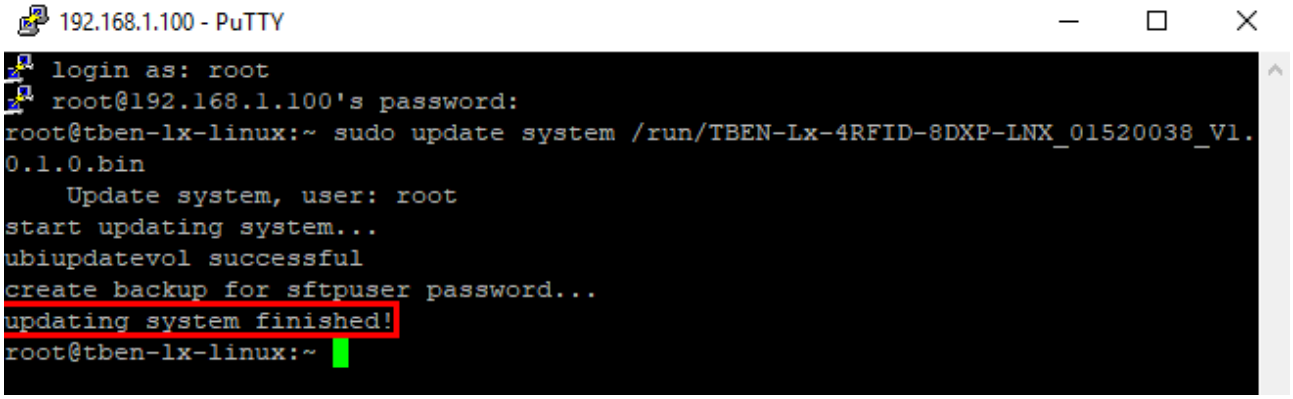

Abb. 31: PuTTY – Update erfolgreich

- Gerät mit dem Befehl `reboot` neu starten.

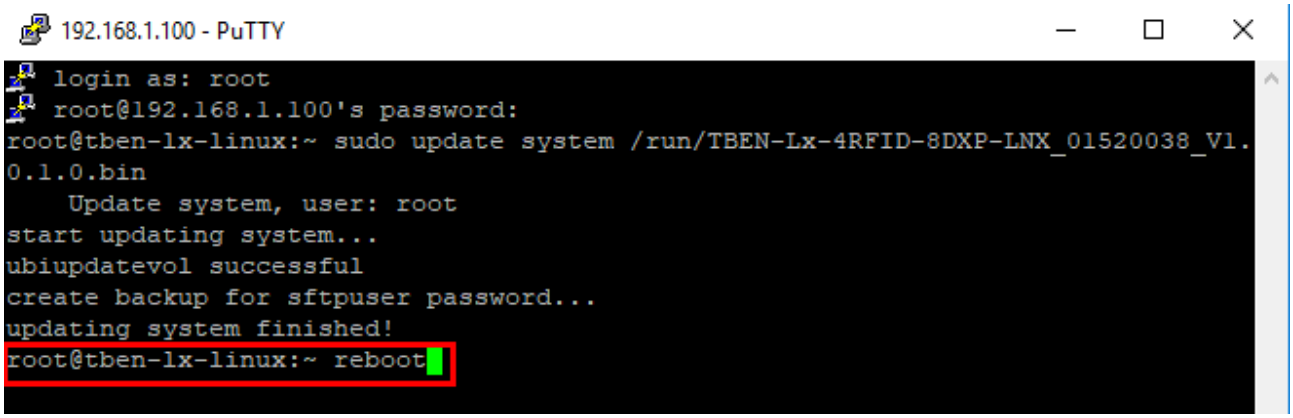

Abb. 32: PuTTY – Gerät neu starten

- Aktuellen Firmware-Stand überprüfen, z. B. über das Turck Service Tool: Der aktuelle Firmware-Stand ist unter **Version** zu finden.

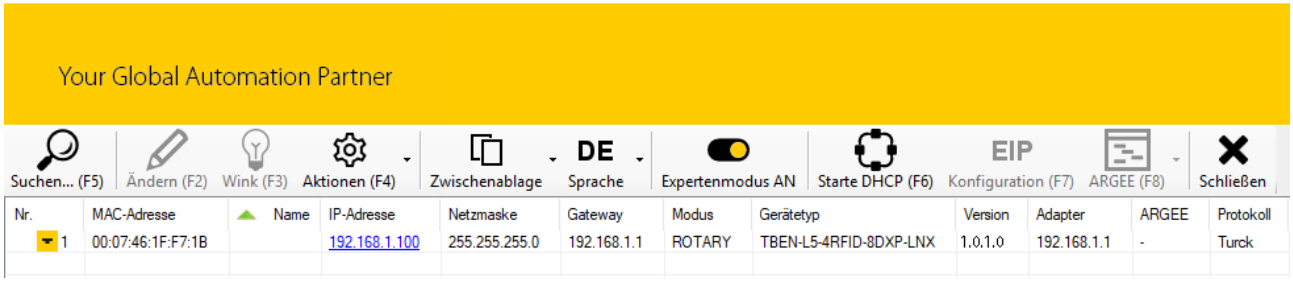

Abb. 33: Turck Service Tool – Firmware-Stand überprüfen

12 Reparieren

Das Gerät ist nicht zur Reparatur durch den Benutzer vorgesehen. Sollte das Gerät defekt sein, nehmen Sie es außer Betrieb. Bei Rücksendung an Turck beachten Sie unsere Rücknahmebedingungen.

12.1 Geräte zurücksenden

Rücksendungen an Turck können nur entgegengenommen werden, wenn dem Gerät eine Dekontaminationserklärung beiliegt. Die Erklärung steht unter
http://www.turck.de/de/produkt-retoure-6079.php
zur Verfügung und muss vollständig ausgefüllt, wetter- und transportsicher an der Außenseite der Verpackung angebracht sein.

13 Entsorgen

Die Geräte müssen fachgerecht entsorgt werden und gehören nicht in den normalen Hausmüll.

14 Technische Daten

Technische Daten	
Versorgung	
Versorgungsspannung	24 VDC
Zulässiger Bereich	18…30 VDC
Gesamtstrom	V1 max. 8 A, V2 max. 9 A bei 70 °C pro Modul
RFID-Versorgung	2 A pro Kanal bei 70 °C
Sensor-/Aktuatorversorgung	2 A pro Steckplatz bei 70 °C
Potenzialtrennung	galvanische Trennung von V1- und V2-Spannungsgruppe
Spannungsfestigkeit	bis 500 VDC V1 und V2 gegenüber Ethernet
Verlustleistung	typisch ≤ 5 W
Systembeschreibung	
Prozessor	Cortex A8, 800 MHz
Speicher	512 MB Flash ROM; 512 MB DDR3 RAM
Erweiterungsspeicher	1 × USB-Host-Port
Echtzeituhr	ja
Betriebssystem	Linux
Systemdaten	
Übertragungsrate	Ethernet 10 Mbit/s/100 Mbit/s
Anschlusstechnik	2 × M12, 4-polig, D-codiert
RFID	
Kanalanzahl	4
Anschlusstechnik	M12
Versorgung	2 A pro Kanal bei 70 °C, kurzschlussfest
Leitungslänge	max. 50 m
Digitale Eingänge	
Kanalanzahl	8
Anschlusstechnik	M12, 5-polig
Eingangstyp	PNP
Art der Eingangsdiagnose	Kanaldiagnose
Schaltschwelle	EN 61131-2 Typ 3, pnp
Signalspannung Low-Pegel	< 5 V
Signalspannung High-Pegel	> 11 V
Signalstrom Low-Pegel	<1,5 mA
Signalstrom High-Pegel	> 2 mA
Potenzialtrennung	galvanische Trennung zu P1/P2
Spannungsfestigkeit	bis 500 VDC (V1 und V1 gegenüber Ethernet)
Digitale Ausgänge	
Kanalanzahl	8
Anschlusstechnik Ausgänge	M12, 5-polig
Ausgangstyp	PNP
Art der Ausgangsdiagnose	Kanaldiagnose

Technische Daten	
Ausgangsspannung	24 VDC aus Potenzialgruppe
Ausgangsstrom pro Kanal	2,0 A, kurzschlussfest, max. 4,0 A pro Steckplatz
Gleichzeitigkeitsfaktor	0,56
Lastart	EN 60947-5-1: DC-13
Kurzschlussschutz	ja
Potenzialtrennung	galvanische Trennung zu P1/P2
Spannungsfestigkeit	bis 500 VDC (V1 und V1 gegenüber Ethernet)
Norm-/Richtlinienkonformität	
Schwingungsprüfung	gemäß EN 60068-2-6
Beschleunigung	bis 20 g
Schockprüfung	gemäß EN 60068-2-27
Kippfallen und Umstürzen	gemäß IEC 60068-2-31/IEC 60068-2-32
Elektromagnetische Verträglichkeit	gemäß EN 61131-2
Zulassungen und Zertifikate	CE
UL Kond.	cULus LISTED 21 W2, Encl.Type 1 IND.CONT.EQ., Encl. Type 1 -40…+55 °C
Allgemeine Information	
Abmessungen (B × L × H)	60,4 × 230,4 × 39 mm
Betriebstemperatur	-40…+70 °C
Lagertemperatur	-40…+85 °C
Einsatzhöhe	max. 5000 m
Schutzart	IP65/IP67/IP69K
MTTF	75 Jahre nach SN 29500 (Ed. 99) 20 °C
Gehäusematerial	PA6-GF30
Gehäusefarbe	schwarz
Material Fenster	Lexan
Material Schraube	303 Edelstahl
Material Label	Polycarbonat
Halogenfrei	ja
Montage	2 Befestigungslöcher, Ø 6,3 mm

15 Anhang: EU-Konformitätserklärung

EU-Konformitätserklärung Nr.: 5035-2M
EU Declaration of Conformity No.:

Wir/ We: HANS TURCK GMBH & CO KG
WITZLEBENSTR. 7, 45472 MÜLHEIM A.D. RUHR

erklären in alleiniger Verantwortung, dass die Produkte
declare under our sole responsibility that the products

Kompakte I/O Module in IP20/IP67:
Compact I/O modules in IP20/IP67:
Typen / types: FDN20-*, FDNL-*, FDNP-*, FDP20-*, FGDP-*, FGEN-*, FLDP-*, FLIB-*, FXEN-*, SDPX-*, TBDP-*, TBEN-*, TBIL-*, TBPN-*

auf die sich die Erklärung bezieht, den Anforderungen der folgenden EU-Richtlinien durch Einhaltung der folgenden Normen genügen:
to which this declaration relates are in conformity with the requirements of the following EU-directives by compliance with the following standards:

EMV - Richtlinie /EMC Directive EN 61131-2:2007 (Abschnitte / section 8, 9, 10)	2014 / 30 / EU	26.02.2014
RoHS – Richtlinie /RoHS Directive	2011 / 65 / EU	08.06.2011

Weitere Normen, Bemerkungen:
additional standards, remarks:

Zusätzliche Informationen:
Supplementary infomation:

Mülheim, den 13.07.2018
Ort und Datum der Ausstellung /
Place and date of issue

i.V. Dr. M. Linde, Leiter Zulassungen /Manager Approvals
Name, Funktion und Unterschrift des Befugten /
Name, function and signature of authorized person

Hans Turck GmbH & Co. KG | T +49 208 4952-0 | F +49 208 4952-264 | more@turck.com | www.turck.com

16 Anhang: Beispiel – „HelloGPIO“ für Node.js

```
// This script show how to use the DXP's of the TBEN-4RFID-8DXP-LNX
// Start the script with "webactive" option will make the I/O useable via webbrowser
// Strat the script without options  will toggle the I/O's

const Gpio = require('onoff').Gpio;

if (process.argv[2] == 'webactive')
{
    runserver();
}
else
{
    //toggle the dxp's...
    time0 =new Date().getTime();
    for(var state = 1; state > -1; state--)
    {
        var index=8;
        while(index < 16 )
        {
            var timenow = new Date().getTime();
            if (timenow - time0 > 1000)
            {
                // func:
                setGpioByInt(index, state);
                index++;
                time0=timenow;
            }
        }
    }
}


// Make the dxp switchable via webbrowser
// http://ip_of_the_board:8080/?dxp=13&value=1
function runserver()
{
    var http = require('http');
    var url = require('url');
    http.createServer(function (req, res){
        res.writeHead(200, {'Contend-Type': 'text/plain'});
        var q = url.parse(req.url, true).query;

        dxp_port= q.dxp;
        dxp_value= q.value;
        //const DXP_Writer = new Gpio(12, "out");
        //DXP_Writer.writeSync(0);

        if (dxp_port > 7 && dxp_port < 16)
        {
            res.write('Would like read DXP' + dxp_port + ' or set to ' + dxp_value );
            if (dxp_value == '1')
            {
                setGpio(dxp_port, 1);
            }
            else if(dxp_value == '0'){
                setGpio(dxp_port, 0);
            }
            res.write('\nDXP'+ dxp_port +' ist: ' + getGpio(dxp_port) + ' now\n');

        }
        else{
            res.write('DXP ports are between 8 and 16 available only');
        }
        res.end('\nsee you...\n');
    }).listen(8080, '192.168.1.81');
    console.log('Server running at http://192.168.1.81:8080/');
}
```

```
function setGpio(w_Pin, val) {

    switch (w_Pin) {
    case "8":
        res = 12;
        break;
    case "9":
        res = 13;
        break;
    case "10":
        res = 47;
        break;
    case "11":
        res = 63;
        break;
    case "12":
        res = 86;
        break;
    case "13":
        res = 87;
        break;
    case "14":
        res = 88;
        break;
    case "15":
        res = 89;
    }
    const DXP_Write = new Gpio(res, "out");
    DXP_Write.writeSync(val);
    console.log('set Gpio '+ res + ' to ' + val);
}

function setGpioByInt(w_Pin, val) {

    switch (w_Pin) {
    case 8:
        res = 12;
        break;
    case 9:
        res = 13;
        break;
    case 10:
        res = 47;
        break;
    case 11:
        res = 63;
        break;
    case 12:
        res = 86;
        break;
    case 13:
        res = 87;
        break;
    case 14:
        res = 88;
        break;
    case 15:
        res = 89;
    }
    const DXP_Write = new Gpio(res, "out");
    DXP_Write.writeSync(val);
    console.log('set Gpio '+ res + ' to ' + val);
}

function getGpio(r_Pin) {

    switch (r_Pin) {
```

```
    case "8":
        res = 110;
        break;
    case "9":
        res = 111;
        break;
    case "10":
        res = 112;
        break;
    case "11":
        res = 113;
        break;
    case "12":
        res = 114;
        break;
    case "13":
        res = 116;
        break;
    case "14":
        res = 117;
        break;
    case "15":
        res = 7;
    }
    const DXP_Read = new Gpio(res, "in");
    DXP_Read.setActiveLow('true');
    var res = DXP_Read.readSync();
    console.log('Gpio '+ r_Pin + ' is: ' + res);
    return res;
}
```

TURCK

Over 30 subsidiaries and over
60 representations worldwide!

100002513 | 2019/05

www.turck.com