

Analoge und digitale Ein- und Ausgänge mit einer USB-Schnittstelle

Version 2-10



Bedienungsanleitung

Dokument-Version A, erstellt am 29.05.2017

Inhalt

Technische Daten.....	3
Charakteristik	3
USB-Schnittstelle	3
Anschlüsse.....	3
digitale Ein- und Ausgänge.....	3
analoge Ein- und Ausgänge.....	3
Spannungsversorgung.....	4
Allgemein	4
Lieferumfang	4
Anschlüsse und Konfigurationselemente.....	5
USB-Schnittstelle	6
Inbetriebnahme.....	8
Digitale Ein- und Ausgänge	10
Analoger Ein- und Ausgang.....	11
Treiberinstallation	14
Installation des virtuellen Ports für die USB-Schnittstelle	14
Softwareschnittstelle.....	15
Funktion der Softwareschnittstelle	15
Steuerung der Kommunikation	20
Steuerung der analogen Ein- und Ausgänge.....	22
Steuerung der digitalen Ein- und Ausgänge	25
Automatischer Datentransfer	28
Fehlerbehandlung	30
Eigendiagnose des Moduls.....	32
Informationen über das Modul	34
Beispiel eines Kommunikationsprogramms.....	37

Technische Daten

Charakteristik

- zwei Analogeingänge, ein Analogausgang
- vier Digitaleingänge, vier Digitalausgänge
- USB-Schnittstelle
- 16-Bit RISC Mikrocontroller
- Stromversorgung über die USB-Schnittstelle
- Hutschienengehäuse nach DIN EN 50022

USB-Schnittstelle

- Standard: USB 2.0
- Datenübertragungsrate: bis zu 12 MBit/s (*Full Speed*)
- Parameter der virtuellen Schnittstelle:
230,4 kBaud, 8 Datenbits, 1 Stopbit, gerade Parität

Anschlüsse

- Ein- und Ausgänge: Schraubenklemmen mit 12 und 9 Kontakten,
Anschlussvermögen:
Leitergrößen: 0,2..2,5 mm²
flexible Leitergrößen mit Aderendhülsen: 0,25..1,5 mm²
- USB-Anschluss: USB-Buchse Typ B

digitale Ein- und Ausgänge

- polarisierte Eingangsrelaisspulen
- nominale Nennspannung der Eingänge: 24 V=
- erlaubter Bereich der Eingangsspannung: 18..48 V= (bei 40°C max.)
- Stromverbrauch der Eingänge: 5,3 mA bei 24 V=
- Ausgangsrelaiskontakte: Schließer
- Schaltleistung der Ausgänge: 50 V= / 50 V~, 1 A, 30 W / 60 VA

analoge Ein- und Ausgänge

- Nennspannung der Eingänge: +10 V
- erlaubter Bereich der Eingangsspannung: -2..+13 V
- Eingangsimpedanz: 400 kΩ
- Spannungsbereich des Ausgangs: 0..+10 V
- maximaler Ausgangsstrom: 5 mA

Spannungsversorgung

- Mikrocontroller und Ausgangsrelais: 5 V= aus der USB-Schnittstelle
- Hilfsspannung 24 V: Versorgung des Analogausgangs
- Stromverbrauch: ca. 13 mA im Ruhezustand
- erlaubter Bereich der Versorgungsspannung: 12..30 V

Allgemein

- Gehäuse:
 - Schienemontage nach DIN EN 50022
 - Material: Kunststoff, Oberteil grau, Unterteil schwarz
 - Abmessungen: 70 x 86 x 60 mm³ (Breite × Höhe × Tiefe)
- Gewicht: ca. 120 g

Lieferumfang

- komplett getestetes und kalibriertes Modul USB-DIO4+4-AIO1+1
- Diagnosesoftware (online)
- Bedienungsanleitung (online)
- optional:
 - USB-Anschlusskabel A-Stecker auf B-Stecker
 - USB-Schnittstellenkarte

Anschlüsse und Konfigurationselemente

Das Modul verfügt über zwei Schraubenklemmen mit 12 und 9 Kontakten (CN1 und CN4 in Abb. 1) zum Anschluss der analogen und digitalen Ein- und Ausgänge. Weiterhin ist ein USB-Steckverbinder (CN3) für die Daten-Schnittstelle und die Stromversorgung vorhanden.

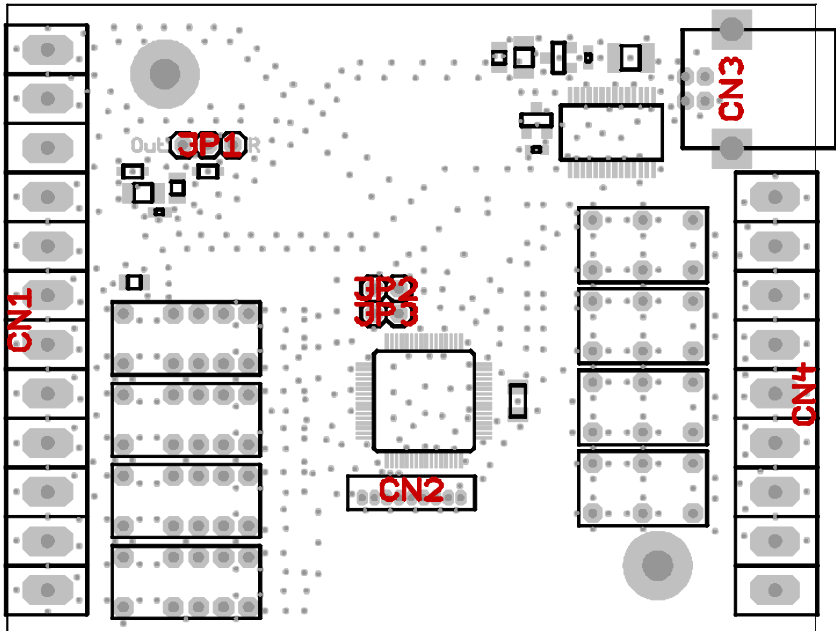


Abb. 1. Anschlüsse des Moduls.

CN1, CN4: Schraubenklemmen für die analogen und digitalen Ein- und Ausgänge, CN3: USB-Anschluss, JP1: Kurzschlussbrücke zur Konfiguration des Analogausgangs AIO1, JP2, JP3: Kurzschlussbrücken zum Programmieren des Mikrocontrollers, CN2: optionaler Programmierstecker.

Nach dem Öffnen des Moduls ist ein Programmierstecker und drei Kurzschlussbrücken zugänglich, mit zweien von denen das Hochla-

den der Firmware ermöglicht wird.[†] Die Kurzschlussbrücke JP1 wird zur Konfiguration des Anschlusses AIO1 benutzt. Wird die Kurzschlussbrücke auf der durch "Out" markierten Seite (links in Abb. 1) eingesetzt, wird der Anschluss AIO1 als Ausgang konfiguriert. Die Messung seiner Spannung kann dann zur Kontrolle der Ausgangsspannung verwendet werden (siehe Abschnitt "Steuerung der analogen Ein- und Ausgänge"). Wird die Kurzschlussbrücke entfernt oder auf der durch "R" markierten Seite (rechts in Abb. 1) eingesetzt, wird der Anschluss AIO1 als Eingang konfiguriert und kann zur Messung einer externen an den Anschluss AIO1 angelegten Spannung benutzt werden. Die Kurzschlussbrücke auf der durch "R" markierten Seite kann zudem einen optionalen Pullup-Widerstand aktivieren, welcher einen definierten Strom in den Anschluss AIO1 liefern kann. Dieser Widerstand kann zum Betrieb bestimmter Sensoren wie der Temperaturfühler Pt1000 eingesetzt werden. Der Pullup-Widerstand wird nur auf Anfrage bestückt, so dass die beiden zuletzt genannten Varianten der Kurzschlussbrücke JP1 (keine Verbindung oder Verbindung rechts) die selbe Funktion erfüllen.

! **Vorsicht:** Das Modul ist ein elektronisches Gerät, das empfindlich auf statische Elektrizität ist. Bei der Manipulation mit dem Modul, besonders beim geöffneten Gehäuse, müssen die ESD-Maßnahmen (*Electro-Static Discharge*) beachtet werden. Es muss unbedingt dafür gesorgt werden, dass bevor ein beliebiges Teil des Moduls angefasst wird, weder die Hände noch das Werkzeug elektrostatisch aufgeladen sind.

! **Vorsicht:** Beim Entfernen des Gehäusedeckels ist äußerste Vorsicht geboten. Die Laschen zur Halterung des Deckels können leicht beschädigt werden.

USB-Schnittstelle

Das Modul verfügt über eine USB-Schnittstelle zur Datenübertragung zwischen dem Modul und einem Steuerrechner (üblicherweise einem PC). Die Datenübertragungsrate beträgt 230,4 kBaud, wodurch kurze Antwortzeiten gesichert sind.

[†] Das Hochladen einer neuen Firmware in das Modul soll nicht direkt vom Kunden durchgeführt werden, wird daher in dieser Bedienungsanleitung nicht beschrieben. Die Kurzschlussbrücken dürfen nicht eingesetzt werden, denn bei einer falschen Bedienung kann das Modul beschädigt werden.

Die USB-Schnittstelle erfordert einen kurzen Abstand zwischen dem Steuerrechner und dem Modul, durch die Datenkabel wird der Steuerrechner mit dem Modul galvanisch verbunden. Durch die galvanische Kopplung können Störsignale vom Steuerrechner übertragen werden oder können Störungen von Außen die Funktion des Steuerrechners beeinflussen.

Für die Funktion der USB-Schnittstelle ist auf der PC-Seite ein virtueller Porttreiber erforderlich, welcher einen virtuellen COM-Port erstellt (siehe Abschnitt "USB-Schnittstelle"). Die Kommunikation mit dem Modul erfolgt über diesen Port, der Schnittstellenadapter kommuniziert intern mit dem Mikrocontroller mit der o.g. Datenübertragungsrate von 230,4 kBaud.

Die USB-Schnittstelle ist mit der USB-Version 2.0 kompatibel, sie nutzt den *Full-Speed Mode*, d.h. dass die Übertragung über USB mit der Datenrate von maximal 12 MBit/s erfolgt. Der Anschluss ist der B-Steckverbinder (CN3 in Abb. 1), für die Verbindung mit einem USB-Host (dem Steuerrechner) ist ein genormtes Kabel mit A- und B-Steckern erforderlich, seine Länge sollte 5 m nicht überschreiten. Die USB-Schnittstelle nutzt die vom Host bereitgestellte Stromversorgung, um den Einschaltstrom auf den von der USB-Norm geforderten Wert zu reduzieren, ist in dem Modul ein Strombegrenzer eingebaut.

Inbetriebnahme

Das Modul USB-DIO4+4-AIO1+1 benötigt für seine Funktion eine funktionierende USB-Schnittstelle. Für den Anschluss ist ein passendes Datenkabel mit der maximalen Länge von 5 m erforderlich.

Um die Funktion des Moduls zu überprüfen, gehen Sie wie folgt vor:

- Schließen Sie das Modul mit Hilfe eines Datenkabels an eine USB-Schnittstelle des PCs an. Besitzt der PC keine (freie) USB-Schnittstelle, muss eine Schnittstellenkarte installiert werden. Auf der PC-Seite muss weiterhin ein virtueller Porttreiber installiert werden (siehe Abschnitt "Treiberinstallation").
- Öffnen Sie ein Kommandozeilenfenster, wechseln Sie in das Verzeichnis mit dem Diagnoseprogramm `USB-ADIO-Test.exe` (Verzeichnis "Program" des Softwarepakets) und starten Sie es. Das Programm zeigt nach dem Start und dem Drücken der Taste '?' die verfügbaren Optionen. Für eine erfolgreiche Ausführung muss mindestens die Nummer der virtuellen seriellen Schnittstelle angegeben werden, an welche das Modul angeschlossen ist. Ist das Modul beispielsweise an die virtuelle Schnittstelle `COM5` angeschlossen, muss das Diagnoseprogramm durch die folgende Anweisung gestartet werden: `"USB-ADIO-Test 5"`. Wird die angegebene Schnittstelle nicht gefunden, endet das Programm mit einer Fehlermeldung. In einem solchen Fall muss im Gerätemanager von Windows überprüft werden, ob die virtuelle Schnittstelle existiert und ob sie funktionsfähig ist. Die Konfiguration der Schnittstelle ist dabei unerheblich, denn sie wird durch das Diagnoseprogramm selbst eingestellt.
- Konnte das Diagnoseprogramm erfolgreich gestartet werden, drücken Sie die Taste 'P' auf der Tastatur des PCs. Das Programm liest aus dem angeschlossenen Modul USB-DIO4+4-AIO1+1 die Produktbezeichnung aus. Sollte anstelle davon ein Fehler gemeldet werden, wurde die Datenübertragung gestört. Wenn die Übertragung auch bei einer Wiederholung nicht korrekt funktioniert, versuchen Sie durch Drücken der Taste 'Z' die Kommunikationspuffer zu löschen und/oder die Kommunikation neu zu starten. Gelingt es auch dann nicht eine fehlerfreie Antwort von dem Modul zu erhalten, liegt ein grundlegender Fehler vor. Überprüfen Sie das Anschlusskabel und die Funktion der benutzten USB-Schnittstelle des PCs.
- Funktioniert alles fehlerfrei, können Sie durch Drücken der Tasten 'N', 'D' und 'V' weitere Produktinformationen abfragen. Gegebenfalls

können auch weitere Funktionen des Moduls getestet werden, siehe dazu die Online-Hilfe zu dem Diagnoseprogramm.

Konnten die Tests fehlerfrei durchgeführt werden, ist das Modul für den Einsatz mit der Benutzersoftware betriebsbereit.

Digitale Ein- und Ausgänge

Die digitalen Ein- und Ausgänge sind an die Schraubenklemmen angeschlossen, die Tabelle 1 zeigt die Pinbelegung. Die digitalen Ein- und Ausgänge sind von der Gerätemasse, d.h. der USB-Schnittstelle und demzufolge auch dem Steuerrechner galvanisch getrennt.

Abb. 2 zeigt die Beschaltung der Ein- und Ausgänge. Die Ausgänge sind an die Kontakte (Schließer) des jeweiligen Relais angeschlossen, das durch das Modul angesteuert wird. Wird ein Kanal aktiviert, schließen die Kontakte. Im Ruhezustand oder ohne externe Stromversorgung sind die Kontakte geöffnet.



Abb. 2. Beschaltung der digitalen Eingänge (links) und der digitalen Ausgänge (rechts).

Die Eingangsanschlüsse steuern direkt die Spule des jeweiligen Relais und das Modul ermittelt den Schaltzustand anhand der Kontakte des Relais. Die Eingänge werden durch das Anlegen der entsprechenden Steuerspannung aktiviert, im stromlosen Zustand sind die Eingänge inaktiv. Bei der Beschaltung muss die Polarität der Steuerspannung beachtet werden.

Da beim Ausschalten der Eingänge durch die in der Induktivität der Relais-Spule akkumulierte Energie hohe Spannungen entstehen können (Selbstinduktion), wurden die Eingänge mit einem aus jeweils einer Diode (siehe Abb. 2) bestehenden Überspannungsschutz ausgestattet. Dadurch werden die angeschlossenen Schaltkreise ausreichend geschützt, weitere Schutzmaßnahmen sind auch bei der Verwendung von Halbleiter-Schaltern (beispielsweise von Näherungssensoren) nicht erforderlich.

! Vorsicht: Wird eine Steuerspannung falscher Polarität oder eine Steuerspannung höher als die nominale angelegt, kann dies zu einer dauerhaften Beschädigung des jeweiligen Eingangs oder sogar des gesamten Moduls führen.

Analoge Ein- und Ausgang

Der analoge Ein- und Ausgang ist zusammen mit der Gerätemasse an die Schraubenklemme angeschlossen (siehe Tabelle 1 und Abb. 3). Die Gerätemasse ist das Referenzpotential beider Anschlüsse, d.h. die Spannungen werden zwischen dem jeweiligen Anschluss und der Gerätemasse gemessen.

Die Analogeingänge A_{IN0} und A_{IO1} können positive Spannungen im Bereich von 0 bis 10 V messen. Der Innenwiderstand liegt bei 400 k Ω , so dass der Eingangsstrom bei der maximalen Eingangsspannung von 10 V nur 25 μ A beträgt. Der erlaubte Bereich der Eingangsspannung beträgt –2 bis +13 V, liegt die Spannung aber außerhalb des o.g. Bereichs von 0 bis 10 V, werden durch das Modul USB-DIO4+4-AIO1+1 falsche Spannungswerte ermittelt.

Der Analogausgang A_{IO1} liefert ebenfalls positive Spannungen im Bereich von 0 bis 10 V, die Voraussetzung dafür ist jedoch, dass die Kurzschlussbrücke JP1 auf der durch "Out" markierten Seite eingesetzt wird (siehe Abschnitt "Anschlüsse und Konfigurationselemente").

Um Spannungen höher als die 5 V-Versorgung erzeugen zu können, welche durch die USB-Schnittstelle bereitgestellt wird, muss das Modul USB-DIO4+4-AIO1+1 mit einer externen Hilfsversorgung von 12 bis 30 V gespeist werden. Wird die Hilfsversorgung nicht angeschlossen, werden die internen Bauteile mit ca. 4,5 V versorgt, so dass der Analogausgang nur Spannungen bis zu dieser Grenze erzeugen kann.

Die Belastbarkeit des Analogausgangs A_{IO1} beträgt maximal 5 mA. Bei höheren Ausgangsströmen sinkt durch etwaige Spannungsabfälle auf den Zuleitungen die Genauigkeit, so dass die Belastung des Analogausgangs stets möglichst klein gehalten werden soll.

Tab. 1. Pinbelegung der Schraubenklemmen

Pin	Bezeichnung	Funktion
1	AIO1	analoger Ausgang/Eingang
2	AIN0	analoger Eingang
3	Gnd	Gerätemasse
4	+24V	Hilfsversorgung +12V bis +30V
5	DOUT0A	digitaler Ausgang 0, Anschluss a
6	DOUT0B	digitaler Ausgang 0, Anschluss b
7	DOUT1A	digitaler Ausgang 1, Anschluss a
8	DOUT1B	digitaler Ausgang 1, Anschluss b
9	DOUT2A	digitaler Ausgang 2, Anschluss a
10	DOUT2B	digitaler Ausgang 2, Anschluss b
11	DOUT3A	digitaler Ausgang 3, Anschluss a
12	DOUT3B	digitaler Ausgang 3, Anschluss b
13	USB	USB-Anschluss, Buchse Typ B
14		
15		
16	DIN3N	digitaler Eingang 3, negativ
17	DIN3P	digitaler Eingang 3, positiv
18	DIN2N	digitaler Eingang 2, negativ
19	DIN2P	digitaler Eingang 2, positiv
20	DIN1N	digitaler Eingang 1, negativ
21	DIN1P	digitaler Eingang 1, positiv
22	DIN0N	digitaler Eingang 0, negativ
23	DIN0P	digitaler Eingang 0, positiv
24	Gnd	Gerätemasse



USB				D13– 16	D13+ 17	D12– 18	D12+ 19	D11– 20	D11+ 21	D10– 22	D10+ 23	Gnd 24
				USB–DIO4+4–AIO1+1								
				91909								
				23/2017								
1	AIO1											
2	AIO0											
3	Gnd											
4	+24V											
5	DO0a											
6	DO0b											
7	DO1a											
8	DO1b											
9	DO2a											
10	DO2b											
11	DO3a											
12	DO3b											

Abb. 3. Beschriftung in der Abdeckung des Moduls.

Treiberinstallation

Installation des virtuellen Ports für die USB-Schnittstelle

Der virtuelle Porttreiber ist für den Anschluss des Moduls an eine USB-Schnittstelle erforderlich. Die Installationsdateien befinden sich im Verzeichnis "USB" des beiliegenden Softwarepakets. Beachten Sie dabei folgendes:

- Alternativ zu den Treibern aus der beiliegenden CD können Standard-Windows-Treiber oder aktuelle Treiber von der Homepage des Herstellers des USB-Adapters heruntergeladen werden. Die Treiber werden unter der Adresse <http://www.ftdichip.com/Drivers/VCP.htm> zur Verfügung gestellt, es soll die zum Betriebssystem passende Datei heruntergeladen werden.
- Zum Installieren der Treiber benötigen Sie Administrationsrechte.
- Die Installation ist in einer der sich in dem o.g. Verzeichnis befindlichen pdf-Datei detailliert beschrieben. Lesen Sie diese Beschreibung sorgfältig durch.
- Nach der Installation kann die Nummer der virtuellen Schnittstelle eingestellt werden. Dies wird im Gerätemanager bei dem Gerät *USB Serial Port* durchgeführt, es sind dazu Administrationsrechte erforderlich. Die Umstellung der Portnummer geschieht sofort nach der Bestätigung, der Rechner muss dabei nicht neu gestartet werden.

Softwareschnittstelle

Die Softwareschnittstelle zu dem Modul USB-DIO4+4-AIO1+1 besteht aus einer dynamischen Linkbibliothek `USB-ADIO.dll`. Sie befindet sich im Verzeichnis "Program" des Softwarepakets. Die Softwareschnittstelle stellt ein selbständiges Softwarepaket dar, für ihre Funktion sind keine weiteren Treiber oder Bibliotheken erforderlich.

Die Benutzerfunktionen in der dynamischen Linkbibliothek `USB-ADIO.dll` können aus allen gängigen Programmiersprachen aufgerufen werden. Die Details entnehmen Sie dem Benutzerhandbuch Ihres Compilers. Die Definitionen der Benutzerfunktionen sind in der Datei `USB-ADIO.h` enthalten. Sollte Ihr Compiler keine Importbibliothek erstellen können, steht Ihnen die Datei `USB-ADIO.lib` zur Verfügung.

Die folgenden Beispiele sind für den Compiler Borland C++ Version 5.0 vorgesehen.

Funktion der Softwareschnittstelle

Die Softwareschnittstelle kann insgesamt 8 Kommunikationskanäle für die Datenübertragung verwalten. Jeder benutzte Kanal muss durch das Öffnen einer seriellen Schnittstelle zugeordnet werden. Ein geöffneter Kanal kann wieder geschlossen werden und danach durch das erneute Öffnen entweder der gleichen oder einer anderen Schnittstelle zugeordnet werden.

Das Öffnen eines Kommunikationskanals konfiguriert die zugeordnete serielle Schnittstelle und löscht ihre Puffer. Direkt nach dem Öffnen kann jede beliebige Funktion zur Datenübertragung aufgerufen werden. D.h. es kann der Zustand der Eingänge abgefragt werden oder können ausgewählte Ausgänge angesteuert werden.

Alle Funktionen erfordern als einen Parameter die Nummer des Kommunikationskanals `PortNumber` (siehe weiter), auf dem die gewünschte Operation ausgeführt werden soll. Es handelt sich um eine vorzeichenlose 16 Bit-Zahl (`WORD`). Die Nummer 0 bezeichnet den ersten Kanal, die Nummer 7 den letzten.

Der Rückgabewert der meisten Funktionen enthält die Aussage über den Erfolg der jeweiligen Operation. Es handelt sich um eine 32 Bit-Zahl mit Vorzeichen (`int`). Der letzte Rückgabewert kann durch die

Funktion `USB_ADIO_State` neu abgefragt werden. Tabelle 2 listet die möglichen Rückgabewerte zusammen mit den Fehlermeldungen auf, wie sie durch die Funktion `USB_ADIO_ErrorMessage` zurückgegeben werden. Ist ein Fehler bei der Datenübertragung entstanden, kann die Ursache durch Aufrufen der Funktionen `USB_ADIO_IO_State` und `USB_ADIO_IO_ErrorMessage` herausgefunden werden. Die erste gibt den letzten I/O-Fehler zurück, die zweite die entsprechende Fehlermeldung (siehe Tab. 3.).

Um die Integration in Pascal-Softwarepakete zu vereinfachen, verwenden alle Funktionen der Softwareschnittstelle die Pascal-Aufrufkonvention.

! **Vorsicht:** Wird bei der Deklaration der Funktionen eine andere Aufrufkonvention angegeben, wird die Softwareschnittstelle nicht richtig funktionieren und es können unvorsehbare und nur schwer diagnostizierbare Fehler entstehen.

Tab. 2. Rückgabewerte der Funktionen

Rückgabewert	Fehlermeldung	Beschreibung
0	No error	Die Funktion endete erfolgreich.
1	No data is available	Es wurden keine Daten empfangen, diesen Rückgabewert liefert nur die Funktion <code>USB_ADIO_CheckAutoInput</code> zurück.
-1	PortNumber out of range	Nummer des Kommunikationskanals <code>PortNumber</code> war größer als maximal zulässig.
-2	Memory for the port data could not be allocated	Bei der Initialisierung konnte dem Kommunikationskanal kein Speicher zugeteilt werden.
-3	Error opening the port	Die Schnittstelle konnte nicht geöffnet werden. Für mögliche Ursachen siehe Tab. 3.
-4	Error closing the port	Die Schnittstelle konnte nicht geschlossen werden. Für mögliche Ursachen siehe Tab. 3.
-5	Error purging the port	Die Puffer der Schnittstelle konnten nicht gelöscht werden. Für mögliche Ursachen siehe Tab. 3.
-6	Error setting the port control lines	Die Statusleitungen der Schnittstelle konnten nicht angesteuert werden.
-7	Error reading the port status lines	Die Statusleitungen der Schnittstelle konnten nicht abgefragt werden.
-8	Error sending command	Die Datenübertragung zum Modul ist fehlgeschlagen. Für mögliche Ursachen siehe Tab. 3.
-9	Error sending data	
-10	Error sending termination character	

Rückgabewert	Fehlermeldung	Beschreibung
-11	Error receiving command	Die Datenübertragung vom Modul ist fehlgeschlagen. Für mögliche Ursachen siehe Tab. 3.
-12	Error receiving data	
-13	Error receiving termination character	
-14	Wrong command received	Es wurde eine falsche Antwort vom Modul erhalten.
-15	Wrong argument received	Die vom Modul empfangene Antwort enthält falsche Daten.
-16	Wrong argument passed to the function	Der an die Funktion übergebene Parameter befand sich nicht im erlaubten Bereich.
-100	Device not connected	Die Statusleitungen der Schnittstelle zeigen, dass das Modul nicht angeschlossen ist.
-101	Device not ready	Die Statusleitungen der Schnittstelle zeigen, dass das Modul für die Kommunikation nicht bereit ist.
-102	Device state could not be set to not ready	Das Modul reagiert nicht korrekt. Versuchen Sie die Kommunikation zurückzusetzen oder das Modul neu zu starten, indem Sie die USB-Verbindung kurzzeitig unterbrechen.

Tab. 3. I/O-Fehler

Rückgabewert	Fehlermeldung	Beschreibung
0	No error	Die Funktion endete erfolgreich.
1	Port has not been opened yet	Es wurde versucht, einen Kommunikationskanal zu verwenden, der noch nicht geöffnet wurde.
2	Cannot open the port	Die spezifizierte Schnittstelle konnte nicht geöffnet werden. Die Schnittstelle existiert entweder nicht oder wird von einem anderen Programm verwendet.
3	Cannot get the state of the port	Der Zustand der Schnittstelle konnte nicht ermittelt werden.
4	Cannot set the state of the port	Der Zustand der Schnittstelle konnte nicht gesetzt werden.
5	Cannot set the timeouts for the port	Das Timeout für die Schnittstelle konnte nicht gesetzt werden.
6	Cannot clear the port	Die Puffer der Schnittstelle konnten nicht gelöscht werden.
7	Error reading data from the port	Der Empfang der Daten ist fehlgeschlagen.
8	Error writing data to the port	Das Senden der Daten ist fehlgeschlagen.
9	Wrong data amount written to the port	Es konnte nicht die richtige Anzahl an Zeichen gesendet werden.
10	Error setting the control lines of the port	Die Steuerleitungen der Schnittstelle konnten nicht angesteuert werden.
11	Error reading the status lines of the port	Der Zustand der Statusleitungen der Schnittstelle konnte nicht ermittelt werden.
12	Device is busy	Das Modul ist nicht bereit für die Kommunikation.

Steuerung der Kommunikation

Funktion USB_ADIO_Open

```
int pascal USB_ADIO_Open  
(WORD PortNumber, WORD ComNumber);
```

Öffnet die angegebene Schnittstelle für den Kommunikationskanal. Sie muss die erste Funktion sein, die während der Kommunikation mit dem jeweiligen Kommunikationskanal ausgeführt wird. Endet diese Funktion mit einem Fehler, ist keine Kommunikation mit dem Kanal möglich. Die Zahl `ComNumber` bezeichnet die Nummer der virtuellen COM-Schnittstelle, die dem Kommunikationskanal zugeordnet werden soll. `ComNumber=1` steht beispielsweise für die Schnittstelle COM1.

Funktion USB_ADIO_Close

```
int pascal USB_ADIO_Close (WORD PortNumber);
```

Schließt den Kommunikationskanal. Sie kann verwendet werden, um den Kommunikationskanal freizugeben, um ihn später durch Öffnen der gleichen oder einen anderen Schnittstelle zuzuordnen.

Wird das Programm beendet, das die Softwareschnittstelle `USB-ADIO.dll` benutzt, werden alle geöffneten Kommunikationskanäle automatisch geschlossen. Der Programmierer muss keine Sorge dafür tragen.

Funktion USB_ADIO_Purge

```
int pascal USB_ADIO_Purge (WORD PortNumber);
```

Löscht die Schnittstellenpuffer und die Kommunikationspuffer des Moduls `USB-DIO4+4-AIO1+1`. Sie kann verwendet werden, um eine gestörte Kommunikation wiederherzustellen.

Funktion USB_ADIO_Buffer_State

```
int pascal USB_ADIO_Buffer_State  
(WORD PortNumber, BOOL & empty);
```

Liest den Zustand des Eingangspuffers des Moduls `USB-DIO4+4-AIO1+1` in die Variable `empty`. Der Rückgabewert ist `TRUE`, wenn der Eingangspuffer leer ist. Wird ein Wert `FALSE` zurückgegeben, enthält

der Eingangspuffer unverarbeitete Daten, was auf eine gestörte Kommunikation hindeutet.

Funktion USB_ADIO_Ret_Buffer_State

```
BOOL pascal USB_ADIO_Ret_Buffer_State  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_Buffer_State`. Sie liefert den Wert der Variable `empty` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_Buffer_State` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion USB_ADIO_Device_Purge

```
int pascal USB_ADIO_Device_Purge  
(WORD PortNumber, BOOL & empty);
```

Löscht den Eingangspuffer des Moduls und liest den Zustand des Eingangspuffers in die Variable `empty`. Der Rückgabewert ist wie bei der Funktion `USB_ADIO_Buffer_State` und sollte nie den Wert `FALSE` besitzen.

Funktion USB_ADIO_Ret_Device_Purge

```
BOOL pascal USB_ADIO_Ret_Device_Purge  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_Device_Purge`. Sie liefert den Wert der Variable `empty` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_Device_Purge` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Steuerung der analogen Ein- und Ausgänge

Funktion USB_ADIO_GetAnalogInput

```
int pascal USB_ADIO_GetAnalogInput  
  (WORD PortNumber, BOOL SecondaryIO,  
   float & Voltage);
```

Liest den zuletzt erfassten Wert der Spannung am Analogeingang des Moduls USB-DIO4+4-AIO1+1 in die Variable `Voltage`. Die Variable enthält nach dem Aufruf der Funktion die gemessene Spannung in Volt. Die Variable `SecondaryIO` spezifiziert dabei, welcher der beiden Analogeingänge benutzt wird. Ist ihr Wert `FALSE`, wird die Spannung am Anschluss `AIN0` ermittelt, beim Wert `TRUE`, wird die Spannung am Anschluss `AIO1` gemessen.

Wird diese Funktion aufgerufen, wird die automatische Datenübermittlung aus dem Modul für den spezifizierten Anschluss unterbrochen, welche durch die Funktion `USB_ADIO_AutoAnalogInput` gestartet wurde.

Funktion USB_ADIO_RetAnalogInput

```
float pascal USB_ADIO_RetAnalogInput  
  (WORD PortNumber, BOOL SecondaryIO);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_GetAnalogInput`. Sie liefert den Wert der Variable `Voltage` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_GetAnalogInput` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion USB_ADIO_AutoAnalogInput

```
int pascal USB_ADIO_AutoAnalogInput  
  (WORD PortNumber, BOOL SecondaryIO);
```

Startet die automatische Übermittlung der Messwerte des Analogeingangs aus dem Modul USB-DIO4+4-AIO1+1. Für die Beschreibung der Variable `SecondaryIO` siehe die Funktion `USB_ADIO_GetAnalogInput`.

Wird die Funktion `USB_ADIO_AutoAnalogInput` aufgerufen, werden bei jeder Erfassung des Analogeingangs durch den Mikrokontroller des Moduls die neuen Daten des spezifizierten Anschlusses automatisch gesendet. Um die Daten zu empfangen, soll periodisch die Funktion `USB_ADIO_CheckAutoInput` aufgerufen werden.

Funktion USB_ADIO_GetLastAnalogInput

```
int pascal USB_ADIO_GetLastAnalogInput  
(WORD PortNumber, BOOL SecondaryIO,  
float & Voltage);
```

Liest den zuletzt automatisch übermittelten Wert der Spannung am Analogeingang des Moduls USB-DIO4+4-AIO1+1 in die Variable `Voltage`. Die Variable enthält nach dem Aufruf der Funktion die gemessene Spannung in Volt. Für die Beschreibung der Variable `SecondaryIO` siehe die Funktion `USB_ADIO_GetAnalogInput`.

Die Funktion soll dann aufgerufen werden, wenn die Funktion `USB_ADIO_CheckAutoInput` den automatischen Datentransfer ermittelt hat, d.h. in der Variable `Mask` das Bit `USB_ADIO_AUTO_AIN` gesetzt wurde.

Funktion USB_ADIO_RetLastAnalogInput

```
float pascal USB_ADIO_RetLastAnalogInput  
(WORD PortNumber, BOOL SecondaryIO);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_GetLastAnalogInput`. Sie liefert den Wert der Variable `Voltage` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_GetLastAnalogInput` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion USB_ADIO_SetAnalogOutput

```
int pascal USB_ADIO_SetAnalogOutput  
(WORD PortNumber, float Voltage);
```

Steuert den Analogausgang `AIO1` anhand der Variable `Voltage` an. Die Variable enthält die erwünschte Spannung in Volt und ihr Wert darf zwischen 0 und 10 liegen.

- ! Beachten Sie, dass wenn Spannungen höher als 4,5 V ausgegeben werden sollen, muss ebenfalls die Hilfsversorgung von 12 bis 24 V anliegen.

Funktion USB_ADIO_GetAnalogOutput

```
int pascal USB_ADIO_GetAnalogOutput  
(WORD PortNumber, float & Voltage);
```

Liest den zuletzt eingestellten Spannungswert des Analogausgangs AIO1 in die Variable *Voltage*. Die Variable enthält nach dem Aufruf der Funktion die Spannung in Volt.

- ! Beachten Sie, dass die Funktion nur den zuvor durch die Funktion *USB_ADIO_SetAnalogOutput* übertragenen Wert zurückgibt. Soll die Messung der tatsächlichen Ausgangsspannung durchgeführt werden, muss dazu die Funktion *USB_ADIO_GetAnalogInput* mit dem Parameter *SecondaryIO = TRUE* benutzt werden.

Funktion USB_ADIO_RetAnalogOutput

```
float pascal USB_ADIO_RetAnalogOutput  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion *USB_ADIO_GetAnalogOutput*. Sie liefert den Wert der Variable *Voltage* zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion *USB_ADIO_GetAnalogOutput* anschließend durch die Funktion *USB_ADIO_State* ermittelt werden.

Steuerung der digitalen Ein- und Ausgänge

Funktion USB_ADIO_GetDigitalInput

```
int pascal USB_ADIO_GetDigitalInput  
(WORD PortNumber, WORD & Data);
```

Liest die digitalen Eingänge aus und speichert den Zustand in die Variable *Data*. Die vier niedrigsten Bits entsprechen dabei jeweils jedem der Eingänge. Dem Zustand des Eingangs In0 entspricht Bit 0, Bit 3 widerspiegelt den Zustand des Eingangs In3. Ist das Eingangsrelais inaktiv also spannungslos, wird das jeweilige Bit gelöscht (0). Ist das Eingangsrelais aktiv also an eine Spannung angeschlossen, wird das jeweilige Bit gesetzt (1).

Wird diese Funktion aufgerufen, wird die automatische Datenübermittlung aus dem Modul unterbrochen, welche durch die Funktion *USB_ADIO_AutoDigitalInput* gestartet wurde.

Funktion USB_ADIO_RetDigitalInput

```
WORD pascal USB_ADIO_RetDigitalInput  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion *USB_ADIO_GetDigitalInput*. Sie liefert den Wert der Variable *Data* zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion *USB_ADIO_GetDigitalInput* anschließend durch die Funktion *USB_ADIO_State* ermittelt werden.

Funktion USB_ADIO_AutoDigitalInput

```
int pascal USB_ADIO_AutoDigitalInput  
(WORD PortNumber);
```

Startet die automatische Datenübermittlung der Zustände der digitalen Eingänge aus dem Modul USB-DIO4+4-AIO1+1.

Wird diese Funktion aufgerufen, werden bei jeder Änderung des Zustandes der digitalen Eingänge die neuen Daten automatisch von dem Modul gesendet. Um die Daten zu empfangen, soll periodisch die Funktion *USB_ADIO_CheckAutoInput* aufgerufen werden.

Funktion USB_ADIO_GetLastDigitalInput

```
int pascal USB_ADIO_GetLastDigitalInput  
(WORD PortNumber, WORD & Data);
```

Liest den zuletzt automatisch übermittelten Zustand der digitalen Eingänge aus dem Modul USB-DIO4+4-AIO1+1 in die Variable `Data`. Für die Beschreibung der Werte siehe die Funktion `USB_ADIO_GetDigitalInput`.

Die Funktion soll dann aufgerufen werden, wenn die Funktion `USB_ADIO_CheckAutoInput` den automatischen Datentransfer ermittelt hat, d.h. in der Variable `Mask` das Bit `USB_ADIO_AUTO_DIN` gesetzt wurde.

Funktion USB_ADIO_RetLastDigitalInput

```
WORD pascal USB_ADIO_RetLastDigitalInput  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_GetLastDigitalInput`. Sie liefert den Wert der Variable `Data` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_GetLastDigitalInput` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion USB_ADIO_SetDigitalOutput

```
int pascal USB_ADIO_SetDigitalOutput  
(WORD PortNumber, WORD Data);
```

Steuert die digitalen Ausgänge anhand der Variable `Data` an. Die vier niedrigsten Bits entsprechen dabei jeweils jedem der Ausgänge. Dem Zustand des Ausgangs `Out0` entspricht Bit 0, Bit 3 widerspiegelt den Zustand des Ausgangs `Out3`. Ist das jeweilige Bit gelöscht (0), wird das Ausgangsrelais ausgeschaltet und die Kontakte des Ausgangs geöffnet. Ist das jeweilige Bit gesetzt (1), wird das Ausgangsrelais aktiviert und die Kontakte des Ausgangs geschlossen.

Funktion USB_ADIO_GetDigitalOutput

```
int pascal USB_ADIO_GetDigitalOutput  
  (WORD PortNumber, WORD & Data);
```

Liest den Zustand der digitalen Eingänge aus und speichert ihn in die Variable `Data`. Die Bits der Variable `Data` haben die gleiche Bedeutung wie bei der Funktion `USB_ADIO_SetDigitalOutput`.

Die Funktion `USB_ADIO_GetDigitalOutput` kann verwendet werden, um den früher gesetzten Zustand der Ausgänge abzufragen oder den aktuellen Zustand des Moduls USB-DIO4+4-AIO1+1 zu ermitteln, wenn die Software neu gestartet wird.

Funktion USB_ADIO_RetDigitalOutput

```
WORD pascal USB_ADIO_RetDigitalOutput  
  (WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_GetDigitalOutput`. Sie liefert den Wert der Variable `Data` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_GetDigitalOutput` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Automatischer Datentransfer

Funktion USB_ADIO_CheckAutoInput

```
int pascal USB_ADIO_CheckAutoInput  
(WORD PortNumber, WORD & Mask);
```

Prüft, ob automatisch übermittelte Daten aus dem Modul USB-DIO4+4-AIO1+1 empfangen wurden, und gibt den ermittelten Zustand zurück. Die Funktion liefert einen der folgenden Rückgabewerte:

Rückgabewert	Bedeutung
0	Es wurden Daten empfangen, sie wurden gespeichert und stehen bereit
1	Die Schnittstelle hat keine Daten empfangen
< 0	Fehler, siehe Tab. 2

Die Funktion liest in die Variable `Mask` den Status der automatisch empfangen Daten. Die Variable ist eine Bitmaske, die eine Kombination folgender Werte sein kann:

Mask	Bedeutung
USB_ADIO_AUTO_AIN = 1 (Bit 0)	Es stehen Daten über den Analogeingang AIN0 bereit, sie können durch die Funktion <code>USB_ADIO_GetLastAnalogInput</code> ermittelt werden.
USB_ADIO_AUTO_AIO = 4 (Bit 2)	Es stehen Daten über den Analogeingang AIO1 bereit, sie können durch die Funktion <code>USB_ADIO_GetLastAnalogInput</code> ermittelt werden.
USB_ADIO_AUTO_DIN = 2 (Bit 1)	Es stehen Daten über die Digitaleingänge bereit, sie können durch die Funktion <code>USB_ADIO_GetLastDigitalInput</code> ermittelt werden.

Die Funktion `USB_ADIO_CheckAutoInput` soll periodisch aufgerufen werden, wenn das Modul USB-DIO4+4-AIO1+1 Daten automatisch sendet. Die gesendeten Daten werden nach dem Versenden in dem Zwischenspeicher der virtuellen seriellen Schnittstelle im Steuerrechner gespeichert. Die Funktion `USB_ADIO_CheckAutoInput` soll so oft ausgeführt werden, dass der Zwischenspeicher nicht überlaufen kann und dass die erwünschte zeitliche Auflösung der Ereignisse gewährleistet wird.

Die automatische Datenübermittlung aus dem Modul wird durch die Funktionen `USB_ADIO_AutoAnalogInput` und `USB_ADIO_AutoDigitalInput` gestartet und durch die Funktionen `USB_ADIO_AnalogInput` und `USB_ADIO_DigitalInput` wieder unterbrochen.

Funktion `USB_ADIO_RetAutoInput`

```
WORD pascal USB_ADIO_RetAutoInput  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_CheckAutoInput`. Sie liefert den Wert der Variable `Mask` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_CheckAutoInput` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion `USB_ADIO_GetAutoMask`

```
int pascal USB_ADIO_GetAutoMask  
(WORD PortNumber, WORD & Mask);
```

Liest den Status der automatisch empfangen Daten in die Variable `Mask`. Die Funktion ermittelt lediglich den aktuellen Zustand und hat keinen Einfluss auf die Kommunikation. Für die Rückgabewerte siehe die Funktion `USB_ADIO_CheckAutoInput`.

Funktion `USB_ADIO_RetAutoMask`

```
WORD pascal USB_ADIO_RetAutoMask  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_GetAutoMask`. Sie liefert den Wert der Variable `Mask` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_GetAutoMask` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Fehlerbehandlung

Funktion USB_ADIO_State

```
int pascal USB_ADIO_State (WORD PortNumber);
```

Liefert einen Fehlercode nach Tab. 2. Sie kann verwendet werden, um das Ergebnis der letzten Operation erneut abzufragen. Die Funktion hat keinen Einfluss auf die Kommunikation und kann zu jeder Zeit aufgerufen werden.

Funktion USB_ADIO_ErrorMessage

```
const char * pascal USB_ADIO_ErrorMessage  
(WORD PortNumber);
```

Liefert eine Fehlermeldung zu dem Ergebnis der letzten Operation. Der Rückgabewert ist ein Zeiger auf eine null-terminierte Zeichenkette, der Inhalt der Fehlermeldung kann Tab. 2 entnommen werden. Die Funktion hat keinen Einfluss auf die Kommunikation und kann zu jeder Zeit aufgerufen werden.

Funktion USB_ADIO_IO_State

```
int pascal USB_ADIO_IO_State  
(WORD PortNumber);
```

Liefert einen I/O-Fehlercode nach Tab. 3. Sie kann verwendet werden, um das Ergebnis der letzten I/O-Operation erneut abzufragen. Die Funktion hat keinen Einfluss auf die Kommunikation und kann zu jeder Zeit aufgerufen werden.

Funktion USB_ADIO_IO_ErrorMessage

```
const char * pascal USB_ADIO_IO_ErrorMessage  
(WORD PortNumber);
```

Liefert eine Fehlermeldung zu dem Ergebnis der letzten I/O-Operation. Der Rückgabewert ist ein Zeiger auf eine null-terminierte Zeichenkette. Der Inhalt der Fehlermeldung kann Tab. 3 entnommen werden. Die Funktion hat keinen Einfluss auf die Kommunikation und kann zu jeder Zeit aufgerufen werden.

Funktion USB_ADIO_OpenDebugFile

```
int pascal USB_ADIO_OpenDebugFile  
(WORD PortNumber, char * FileName);
```

Öffnet eine Textdatei zur Fehlersuche in der Software. Die Variable `FileName` spezifiziert den Pfadnamen der Datei, die Funktion liefert einen Fehlercode nach Tab. 2 oder 1, wenn die Datei nicht geöffnet werden kann. Existiert die Datei bereits vor dem Aufruf der Funktion, wird sie ohne Warnung überschreiben.

In die spezifizierte Textdatei werden alle Zugriffe auf den spezifizierten Kommunikationskanal inkl. genaue Zeitangaben protokolliert. Dies bietet eine Hilfe bei der Softwareentwicklung, insbesondere wenn es unerklärbare Fehler bei der Kommunikation mit dem Modul USB-DIO4+4-AIO1+1 gibt. Um den Inhalt der Datei im Detail zu entziffern, mag es allerdings erforderlich sein, den Hersteller des Moduls zu kontaktieren. In diesem Falle bitte eine möglichst genaue Beschreibung des Applikationssoftware beilegen

Funktion USB_ADIO_CloseDebugFile

```
int pascal USB_ADIO_CloseDebugFile  
(WORD PortNumber);
```

Schließt die Textdatei zur Fehlersuche in der Software, welche vorher durch die Funktion `USB_ADIO_OpenDebugFile` geöffnet wurde.

Durch den Aufruf der Funktion `USB_ADIO_CloseDebugFile` wird die Protokollierung gestoppt. Eine Fortsetzung der Protokollierung ist jederzeit möglich, die vorher abgespeicherte Datei wird aber dabei überschreiben, wenn die neue Datei denselben Namen besitzen sollte.

Eigendiagnose des Moduls

Funktion USB_ADIO_GetCPUVoltage

```
int pascal USB_ADIO_GetCPUVoltage  
(WORD PortNumber, float & Voltage);
```

Liest den zuletzt erfassten Wert der Versorgungsspannung des Mikrocontrollers des Moduls USB-DIO4+4-AIO1+1 in die Variable `Voltage`. Die Variable enthält nach dem Aufruf der Funktion die gemessene Spannung in Volt.

Funktion USB_ADIO_RetAnalogInput

```
float pascal USB_ADIO_RetCPUVoltage  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_GetCPUVoltage`. Sie liefert den Wert der Variable `Voltage` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_GetCPUVoltage` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion USB_ADIO_GetCPUTemperature

```
int pascal USB_ADIO_GetCPUTemperature  
(WORD PortNumber, float & Temperature);
```

Liest den zuletzt erfassten Wert der Temperatur des Mikrocontrollers des Moduls USB-DIO4+4-AIO1+1 in die Variable `Temperature`. Die Variable enthält nach dem Aufruf der Funktion die gemessene Temperatur in °C.

Funktion USB_ADIO_RetCPUTemperature

```
float pascal USB_ADIO_RetCPUTemperature  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_GetCPUTemperature`. Sie liefert den Wert der Variable `Temperature` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion

USB_ADIO_GetCPUtemperature anschließend durch die Funktion USB_ADIO_State ermittelt werden.

Funktion USB_ADIO_GetUptime

```
int pascal USB_ADIO_GetUptime  
(WORD PortNumber, DWORD & Seconds,  
WORD & Milliseconds);
```

Liest die Zeit in die Variablen `Seconds` und `Milliseconds`, welche seit dem Einschalten, bzw. seit dem letzten Neustart des Moduls USB-DIO4+4-AIO1+1 verstrichen ist. Die Variablen enthalten nach dem Aufruf der Funktion die abgelaufene Zeit in Sekunden und Millisekunden, die Zeit in Sekunden im Fließkomma-Format kann als $(1E-3 * \text{Milliseconds} + \text{Seconds})$ berechnet werden. Die Angabe über die Zeit kann verwendet werden, um festzustellen, ob das Modul einen ungeplanten Neustart durchgeführt hat. Die Zeitauflösung beträgt 1/64 s.

Informationen über das Modul

Funktion USB_ADIO_Version

```
WORD pascal USB_ADIO_Version();
```

Liefert die Version der Softwareschnittstelle (der dynamischen Linkbibliothek USB-ADIO.dll). Sie soll verwendet werden, um festzustellen, ob eine Softwareschnittstelle mit der passenden Version verwendet wird. Die Funktion soll aufgerufen werden, bevor die restlichen Funktionen der Softwareschnittstelle verwendet werden.

Der Rückgabewert ist eine vorzeichenlose 16 Bit-Zahl (WORD), in dem höheren Byte wird die Hauptversionsnummer zurückgegeben, das niedrigere Byte bezeichnet die Reihenfolge innerhalb der Hauptversion. Eine identische Hauptversionsnummer zweier Bibliotheken USB-ADIO.dll bedeutet, dass diese gleiche Funktionen implementieren und dass lediglich interne Korrekturen vorgenommen wurden. Sind die Hauptversionsnummern unterschiedlich, besteht keine Garantie, dass die Bibliothek eingesetzt werden kann, ohne dass das Programm neu kompiliert oder sogar modifiziert werden muss.

Funktion USB_ADIO_FW_Ver

```
int pascal USB_ADIO_FW_Ver  
(WORD PortNumber, WORD & Version);
```

Liefert die Version der Firmware des Moduls USB-DIO4+4-AIO1+1. Sie soll verwendet werden, um festzustellen, ob die Softwareschnittstelle eine passende Version zu der Firmware hat.

Der Rückgabewert ist wie bei der Funktion `USB_ADIO_Version` eine vorzeichenlose 16 Bit-Zahl `Version`. Die dynamische Linkbibliothek USB-ADIO.dll und die Firmware des Moduls sind miteinander kompatibel, wenn die Hauptversionsnummern identisch sind.

Funktion USB_ADIO_Ret_FW_Ver

```
WORD pascal USB_ADIO_Ret_FW_Ver  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_FW_Ver`. Sie liefert den Wert der Variable `Version` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg

des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_FW_Ver` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion `USB_ADIO_FW_Date`

```
int pascal USB_ADIO_FW_Date  
(WORD PortNumber, char * DateString);
```

Liefert das Kompilationsdatum der Firmware des Moduls USB-DIO4+4-AIO1+1. Der Rückgabewert ist eine null-terminierte Zeichenkette, sie wird in den Puffer `DateString` kopiert. Der Puffer `DateString` muss vor dem Aufruf der Funktion definiert werden und muss groß genug sein, um 11 Zeichen des Rückgabewerts (beispielsweise 10 JUN 2014) und das Abschlusszeichen aufnehmen zu können.

Funktion `USB_ADIO_prod_No`

```
int pascal USB_ADIO_prod_No  
(WORD PortNumber, DWORD & Number);
```

Liefert die Produktnummer des Moduls USB-DIO4+4-AIO1+1. Der Rückgabewert `Number` ist eine vorzeichenlose Zahl.

Die Produktnummer ist eine einzigartige vom Hersteller vergebene Nummer, die zur eindeutigen Identifizierung jedes Gerätes benutzt werden kann.

Funktion `USB_ADIO_Ret_prod_No`

```
DWORD pascal USB_ADIO_Ret_prod_No  
(WORD PortNumber);
```

Die Funktion ist die Alternative zu der vorherigen Funktion `USB_ADIO_prod_No`. Sie liefert den Wert der Variable `Number` zurück, so dass diese nicht als Referenz übergeben muss. Den Erfolg des Aufrufs muss im Gegensatz zu der Funktion `USB_ADIO_prod_No` anschließend durch die Funktion `USB_ADIO_State` ermittelt werden.

Funktion USB_ADIO_prod_ID

```
int pascal USB_ADIO_prod_ID  
    (WORD PortNumber, char * Identification);
```

Liefert den Identifikationstext des Moduls USB-DIO4+4-AIO1+1. Der Rückgabewert ist eine null-terminierte Zeichenkette, sie wird in den Puffer `Identification` kopiert. Der Puffer `Identification` muss vor dem Aufruf der Funktion definiert werden und muss groß genug sein, um alle Zeichen des Rückgabewerts aufnehmen zu können. Empfohlen ist eine Pufferlänge von mindestens 32 Zeichen.

Beispiel eines Kommunikationsprogramms

```

#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "USB-ADIO.h"

int main()
{
    // define the port number:
    unsigned short PortNum = 0;

    // open the port, attach it to COM1:
    if (USB_ADIO_Open (PortNum, 1))
    {
        printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));
        return 1; // handle a possible error
    }

    printf ("Press 'i' to get identification data, "
           "or 'Esc' to stop\n");

    // Set automatic transmission of analog input voltage:
    if (USB_ADIO_AutoAnalogInput (PortNum, false))
        printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));

    // Set automatic transmission of digital input state:
    if (USB_ADIO_AutoDigitalInput (PortNum))
        printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));

    // main program loop:
    while (true)
    {
        WORD Mask;
        // check for data:
        const int Result = USB_ADIO_CheckAutoInput (PortNum, Mask);

        if (Result < 0) // handle a possible error
            printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));

        if (Result == 0) // automatic data ready?
        {
            if (Mask & USB_ADIO_AUTO_AIN)
            {
                float Volt = 0;
                if (USB_ADIO_GetLastAnalogInput (PortNum, false, Volt))
                    printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));
                else
                    printf ("Analog input: %.3fV\n", Volt);
            }
            if (Mask & USB_ADIO_AUTO_DIN)
            {
                WORD Data = 0;
                if (USB_ADIO_GetLastDigitalInput (PortNum, Data))
                    printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));
                else

```

```

        printf ("Digital input: %d (%02Xh)\n",
               Data, Data);
    }
}

if (kbhit() == 0) continue; // repeat if no key is pressed
const int Key = getch();    // read the pressed key

// get identification data:
if (Key == 'i')
{
    // get firmware version:
    WORD Version;
    if (USB_ADIO_FW_Ver (PortNum, Version))
        printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));
    else
        printf ("Firmware version: %d.%02d\n",
               Version / 0x100, Version & 0xFF);

    // get firmware date:
    char DateString [20];
    if (USB_ADIO_FW_Date (PortNum, DateString))
        printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));
    else
        printf ("Firmware date: %s\n", DateString);

    // get product identification:
    char IDString [20];
    if (USB_ADIO_prod_ID (PortNum, IDString))
        printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));
    else
        printf ("Product identification: %s\n", IDString);

    // get product number:
    DWORD Number;
    if (USB_ADIO_prod_No (PortNum, Number))
        printf ("%s\n", USB_ADIO_ErrorMessage (PortNum));
    else
        printf ("Product number: %d\n", Number);
}

// stop on 'Esc':
if (Key == 27)
    break;
}

//finish the program:
return 0;
}

```