Features

- **Modbus-Interface**
  - Weit verbreitetes Mess- und Automatisierungs-protokoll
- **gpio.NET *Core* Interface**
  - Einheitlicher Register- und Befehlssatz
  - Unterstützt Modbus RTU und Modbus ASCII über RS232/RS485
  - Unterstützt Modbus ASCII über USB
  - Gateway-Funktionalität mit Protokollkonverter zwischen RS232/RS485 (RTU) und USB (ASCII)
  - Konfiguration wird bei Systemstart aus dem EEPROM geladen
- **RS232/RS485**
  - Einstellbare Baudraten zwischen 1200 Baud und 1MBit
  - Frei konfigurierbare Schnittstellenparameter (Datenbreite, Parität, Stoppbits)
  - Wählbarer Betriebsmodus (RS232 oder RS485)
- **USB**
  - USB 2.0 CDC-Device
- **Eindeutige Seriennummer**
  - Modbus-Addresse ist per Modbus-Befehl über die Seriennummer einstellbar
- **1 KByte EEPROM**
  - Trennung in Config-Section und User-Section
  - Beide Bereiche sind unabhängig voneinander schreibschützbar
- **8 analoge Eingänge**
  - Strom- oder Spannungseingänge
  - 0-4,096V oder 0-24mA
  - bis 1 kHz

gpio.NET

AI

gpio.NET analoge Input-Karte

gpio.AI
U-Serie

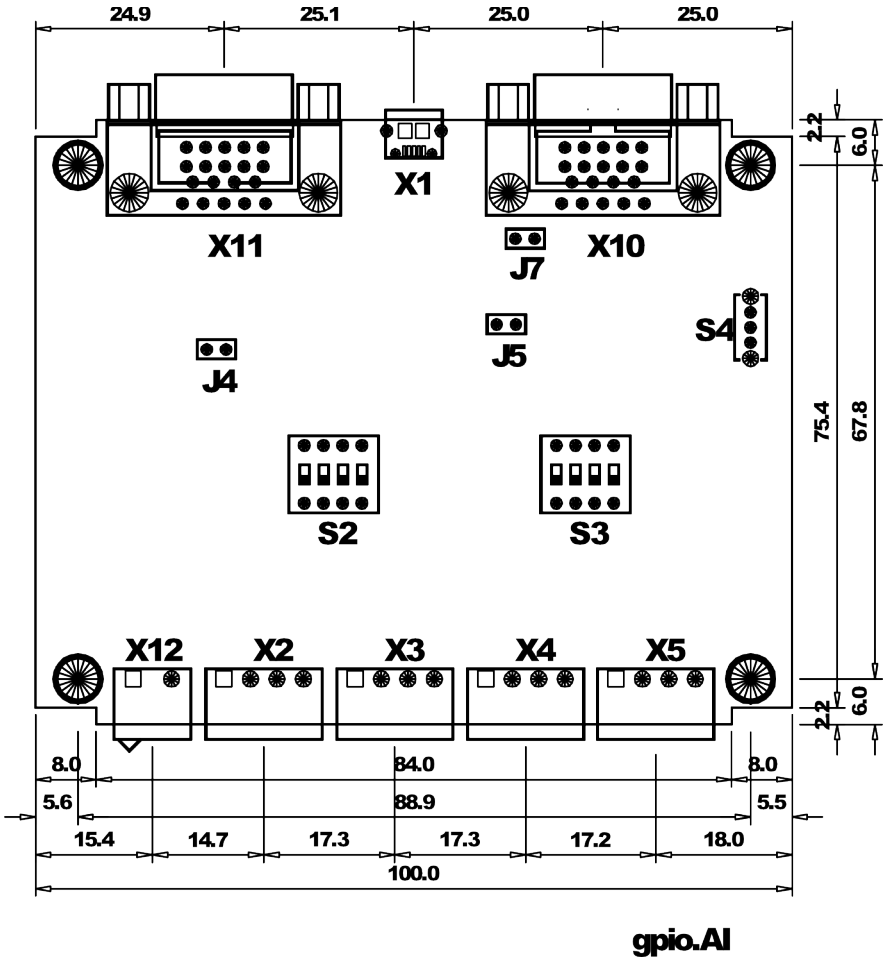
gpio.AI
R-Serie

20140127gai

1 Überblick

Die gpio.AI-Module bieten acht analoge Eingänge, die wahlweise Spannungen oder Ströme messen können. Alle Module lassen sich problemlos in bestehende Modbus-Systeme[1] integrieren und können über eine individuelle Seriennummer sogar während des Betriebs identifiziert und konfiguriert werden. Darüber hinaus bieten sie eine, über die gesamte gpio.NET-Familie einheitliche, Registerschnittstelle.

1.1 Skizze



Zeichnung 1: Abmaße und Beschriftung

1.2 Schnittstellen

Zur Kommunikation verfügen gpio.NET-Module über ein oder zwei Modbus-Schnittstellen. Sofern zwei Schnittstellen vorhanden sind (R-Variante), findet zwischen ihnen ein Routing statt. DSUB-Buchse (`X11`) und DSUB-Stecker (`X10`) finden sich nur auf der R-Variante.

1.2.1 USB

Der USB-Deviceport (`X1`) gehört zur Standardausstattung der gpio.NET-Karten.

1 Modbus ist ein weit verbreiteter industrieller Feldbus. Eine Beschreibung findet sich auf http://modbus.org/tech.php.

Über diese Schnittstelle ist das Modul gängigen PCs oder Laptops zugänglich. Um dem und daraus resultierenden Gegebenheiten Rechnung zu tragen, wird über die USB-Schnittstelle ausschließlich per Modbus-ASCII kommuniziert.[2] Die gpio.NET-Karte wird vom Host als virtuelle serielle Schnittstelle erkannt und unterscheidet sich somit für ein Programm nicht von einer realen Schnittstelle.

Der USB-Deviceport wird in der Regel beim Direktanschluss der Karte an den Host-Rechner als Spannungsversorgung genutzt. Alternativ kann diesen Zweck auch `PIN9` an `X10` oder `X11` erfüllen.[3] Dabei ist darauf zu achten, dass die zugeführte, geregelte Spannung 5 Volt beträgt.

1.2.2 RS232

Die serielle Schnittstelle mit RS232-Pegeln ist auf der R-Variante des gpio.NET-Moduls an der DSUB-Buchse `X11` verfügbar. Ausser den beiden Datenleitungen und der Signalmasse werden keine Steuerleitungen benutzt.[4] Über RS232 lassen sich sowohl Modbus-ASCII als auch Modbus-RTU verwenden. Dabei sind Baudraten bis zu 250k Baud möglich. Durch die gleichzeitige Verwendung von USB und RS232 lässt sich eine weitere, über RS232 angeschlossene Karte, vom Host-Rechner (PC/Laptop) per USB ansprechen.

Pin	X11
1	
2	RXD
3	TXD
4	
5	GND
6	
7	
8	
9	VBUS

Tabelle 1: Belegung RS232

2 Der von Modbus standardmäßig verwendete Modus RTU setzt die exakte Bestimmung des Zeitpunktes, an dem ein Zeichen empfangen wurde, voraus. Dieses Kriterium lässt sich auf einem Multitasking-System wie einem Standard-PC jedoch nicht ohne spezielle Treibersoftware garantieren, wodurch Modbus RTU störanfällig wird. Durch die Eigenschaften des USB wird dieser Effekt zusätzlich verstärkt. Modbus-ASCII unterliegt diesen Kriterien nicht und bietet so eine zuverlässige Grundlage bei der Nutzung von Standardhardware und -betriebssystemen.

3 Die Ringindikatoren (`PIN9`) beider DSUB sind mit der USB-Betriebsspannung (`VBUS`) verbunden (gilt für `X10` nur bei gestecktem Jumper `J7`). Auf diesem Wege lässt sich die Betriebsspannung von einer gpio.NET-Karte auf die nächste beziehungsweise an alle Teilnehmer eines RS485-Busses führen.

4 Beim Anschluss an einen PC per RS232 ist auf die Belegung der RS485-Schnittstelle zu achten, da diese Steuerleitungen der standard RS232-Schnittstelle als Datenleitungen benutzt. Diese Pins (`DTR`, `DSR`) sollten **nicht** mit einem PC verbunden werden.

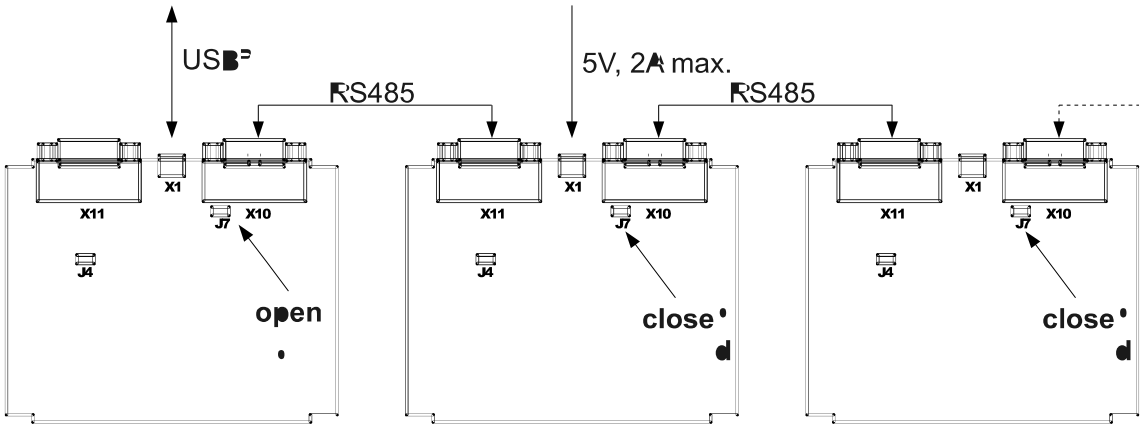
1.2.3 RS485

Die R-Variante bietet die Möglichkeit der Vernetzung über einen RS485-Bus. Zu diesem Zweck sind beide Potentialleitungen sowohl auf `X10` als auch auf `X11` verfügbar.[5] Auf diese Weise lässt sich ein RS485-Bus leicht mittels RS232-Kabeln realisieren. Wie auch bei RS232 lassen sich sowohl Modbus-ASCII als auch Modbus-RTU verwenden. Die maximale Bitrate beträgt 1 Mbit. Ein 110 Ohm Widerstand kann als Terminierung durch Jumper `J4` zugeschaltet werden.

Pin	X10	X11
1		
2		
3		
4	RS485+	RS485+
5	GND	GND
6	RS485-	RS485-
7		
8		
9	VBUS	VBUS

Tabelle 2: Belegung RS485

Über RS485 verbundene Karten werden per USB zum Hostrechner geroutet. So ist der Zugriff auf den RS485-Bus ohne spezielle PC-Hardware möglich.



Zeichnung 2: RS485-Bus an USB mit zusätzlicher Spannungsversorgung

Teile eines analog zur Skizze aufgebauten Busses lassen sich an beliebigen Stellen durch ein eigenes Netzteil versorgen. Hier ist darauf zu achten, dass `VBUS` der so entstandenen Gruppen tatsächlich voneinander getrennt ist. Das Beispiel zeigt dies anhand der, am Host-PC per USB angeschlossenen, ersten Karte und dem Rest-Bus, der durch ein separates USB-Netzteil versorgt wird. Der geöffnete Jumper `J7` unterbricht die Verbindung von `VBUS` über `PIN9` der DSUB.

5 Die Potentialleitungen nutzen die Pins der RS232-Steuersignale `DTR` und `DSR`. Ein direkter Anschluss dieser Signale an einen PC is zu **vermeiden**.

2 Inbetriebnahme

Dieser Schnelleinstieg in die gpio.NET-Familie illustriert die Verwendung der Baugruppe an einem Standard-PC. Stromversorgung und Kommunikation finden direkt über USB (X1) statt. Nachdem die Baugruppe mit Spannung versorgt wurde, läuft ein interner Setupprozess ab. An dieser Stelle wird die Modbusadresse des Moduls ermittelt und die Konfiguration aus dem EEPROM geladen. Anschließend ist das Gerät unter der konfigurierten Adresse ansprechbar.

2.1 Treiberinstallation

Die gpio.NET-Karten erscheinen am PC als serieller Schnittstellenadapter, wenn sie per USB angeschlossen werden. Damit Windows seinen Treiber (usbser.sys) verwendet, muss dem System dies mitgeteilt werden. Zu diesem Zweck stellt Taskit eine entsprechende Konfigurationsdatei zur Verfügung (gpio.NET.inf). Die Verbindung zur Karte kann dann über den neuen Comport hergestellt werden (zum Beispiel //./COM12).

Ubuntu 11.04 erkennt gpio.NET-Module automatisch. Hier kann die Device-Datei /dev/ttyACM0 verwendet werden. Bei anderen Linux-Distributionen kann es nötig sein, den USB-Treiber von Hand zu laden.

```
# modprobe usbserial vendor=0x03EB product=0x2044
```

Die Device-Datei lautet dann für gewöhnlich zum Beispiel /dev/ttyUSB0 oder /dev/usb0. Durch das Hinzufügen folgender Zeile in die Datei /etc/modules wird der Treiber auf Linux-Rechnern automatisch beim Hochfahren geladen:

```
usbserial vendor=0x03EB product=0x2044
```

2.2 Setzen der Adresse

Die Modbus-Device-Adresse wird beim Starten aus dem EEPROM-Register HREG_MODBUS_ADDR gelesen (siehe **Register-Layout**, **Persistente EEPROM-Konfiguration**). Beinhaltet dieses Register einen gültigen Wert (1 – 253) wird diese Addresse benutzt. Anderenfalls verwendet das Modul die Addresse 1. Es muss jedoch jede Addresse im Busbetrieb einzigartig sein. Alternativ kann die Addresse automatisch über die Steckposition im gpio.NET-Rack ermittelt werden.

Geändert werden kann die Device-Addresse entweder über das direkte Schreiben des EEPROM-Registers HREG_MODBUS_ADDR oder per Kommando WRITE_BY_SERIAL. Die zweite Vorgehensweise wird im folgenden Abschnitt demonstriert.

2.3 Zugriff über Simple Client

Simple Client ist ein Tool, das eine Shell auf Basis der *taskit-modbus-lib* zum Zugriff auf Modbus-Komponenten bietet. Das Programm ist Teil dieser Modbus-Bibliothek und sowohl für Windows und Linux in kompilierter Form als auch im Quelltext verfügbar.

An dieser Stelle soll ein kleines Tutorial das Setzen der Modbus-Addresse sowie das Nutzen einer gpio.NET-Karte als USB-Gateway zu einem RS485-Modbus-

System veranschaulichen.

Das Beispiel geht von der Annahme aus, dass die gpio.NET-Karte `A` am virtuellen Comport 12 hängt und sich im Auslieferungszustand befindet. Eine weitere Karte `B` – ebenfalls im Ausliefungszustand – ist per RS485 mit `A` verbunden.

1. **Konfiguration der Schnittstelle (USB, Windows)**

```
# open //./COM12 19200 7 even 2
# set-mode ascii
```

Die Schnittstelle `COM12` wird mit 19200 Baud und den Parametern `7E2` geöffnet. Dies ist der Standard für Modbus-ASCII. Baudrate und Parameter finden bei USB jedoch keine Verwendung und werden daher ignoriert. Der zweite Befehl stellt die Modbus-Bibliothek auf Modbus-ASCII um.

2. **Setzen der Device-Addresse**

```
# report-id 1
count:   53
id:      0x01
running: 1
data:    GPIO.NET:01.00:card type :000001AEB664:www.gpio.net
```

In der Antwort des Kommandos befindet sich unter anderem die Seriennummer des gpio.NET-Moduls `A` (Device-Addresse `1`). Hier ist dies `000001AEB664`. Mit Hilfe ihrer Hilfe kann nun die Device-Addresse verändert werden.

```
# write-by-serial 000001AEB664 5
```

Der Befehl wird per Broadcast (Device-Addresse `0`) an alle Module, die über Karte `A` (inklusive Karte `A`) per RS485 erreichbar sind gesendet. Das Modul, dessen Seriennummer übereinstimmt, übernimmt die neue Addresse (hier `5`). Auf dieses Kommando erfolgt keine Antwort.

3. **Lesen von Holding-Registern**

```
# readh 5 0 16
0x0000  0x0000  0x0004  0x0008  0x0002  0x0000  0x0001  0x0001
0x0000  0x0000  0x0000  0x0000  0x0000  0x0000  0x0000  0x0000
```

Liest vom Modul – unter Verwendung der neuen Device-Addresse `5` – 16 Holding-Register ab Addresse `0x0000`. Dies sind die *Core*-Register, die im Kapitel **Das gpio.NET *Core* Interface** beschrieben sind.

Über die USB-Verbindung können auch Geräte mit anderen Device-Addressen, die sich gemeinsam mit dem USB-Modul in einem RS485-Netz befinden, angesprochen werden. So wird in unserem Beispiel unter der Device-Addresse `1` nun nicht mehr das Modul `A` sondern `B` gefunden (Auslieferungszustand -> Device-Addresse `1`).

3 Das gpio.NET *Core* Interface

Das gpio.NET *Core* Interface erlaubt die Vernetzung über RS232, RS485 und USB. Dank der Unterstützung beider Modbus-Protokolle (Modbus-RTU und Modbus-ASCII) kann ein gpio.NET-Modul als Gateway zwischen USB und RS485-Bus dienen und dabei Konverter zwischen beiden Protokollen sein. So ist es einfacher ein Modbus-System über einen normalen Desktop-PC oder Laptop zu kontrollieren.

Die folgenden Modbusbefehle werden von gpio.NET unterstützt:

Opcode	Name
0x01	Read Coils
0x02	Read Discrete Inputs
0x03	Read Holding Registers
0x04	Read Input Register
0x05	Write Single Coil
0x06	Write Single Register
0x0F	Write Multiple Coils
0x10	Write Multiple Registers
0x11	Report Slave ID[6]
0x41	Write By Serial[7]

Tabelle 3: Unterstützte Befehle

6 Standardbefehl; liefert u.a. einen CSV-Identifikationsstring der Form `GPIO.NET:01.00:card type :000001AEB664:www.gpio.net`, dessen dritte Position die zwölfstellige Seriennummer des gpio.NET-Moduls enthält. Trennzeichen ist der Doppelpunkt (`:`).

7 Taskit-spezifisch; erwartet zwei Argumente – die zwölfstellige Seriennummer eines gpio.NET-Moduls und eine Modbus-Device-ID. Dieser Befehl ist als Broadcast-Befehl gedacht, um einem Gerät, das nur durch seine Seriennummer bekannt ist, während des Bus-Betriebs eine definierte Addresse zu geben. Das Gerät, dem die gesendete Seriennummer inhärent ist, übernimmt das zweite Argument als Modbus-Device-Addresse. Eine Beschreibung findet sich in der Dokumentation der *taskit-modbus-lib*.

3.1 Register-Layout

Der gemeinsame Registersatz vereinfacht das Arbeiten mit unterschiedlichen gpio.NET-Modulen. Geräteidentifikation, Konfiguration der Schnittstellen und das Grundverhalten ist auf allen Modulen einheitlich.

Das gpio.Net *Core* Interface definiert vier Bereiche im Adressraum der *Holding Register*.[8] Bereich `0x0000-0x00FF` enthält allgemeine Konfigurationsregister – die *Core*-Register. Alle gpio.AI-spezifischen Register belegen den Bereich der *Application*-Register `0x0100-0x0FFF` an den sich das EEPROM anschließt. Register `0x1000-0x107F` dienen der Konfiguration des Start-up-Verhaltens der Baugruppe. Diese Gruppe enthält im Wesentlichen Defaultwerte der *Core*- und *Application*-Register. Der Bereich `0x2000-0x237F` steht dem Anwender als persistenter Speicher zur freien Verfügung.

Die EEPROM-Bereiche `0x1000-0x107F` und `0x2000-0x237F` sind einzeln vor versehentlichem Überschreiben schützbar.

Alle Register sind – wie bei Modbus üblich – 16 Bit breit.

Holding Regs	
0x0000	HREG_CTRL
0x0001	HREG_PM
0x0002	HREG_BAUDSEL
0x0003	HREG_DBITS
0x0004	HREG_PARITY
0x0005	HREG_STOP
0x0006	HREG_SERIAL_MODE
0x0007	HREG_MODBUS_MODE
…	Core-Register frei
0x0100	Application-Register
…	
0x1000	unbenutzt
0x1001	HREG_PM
0x1002	HREG_BAUDSEL
0x1003	HREG_DBITS
0x1004	HREG_PARITY
0x1005	HREG_STOP
0x1006	HREG_SERIAL_MODE
0x1007	HREG_MODBUS_MODE
…	unbenutzt
0x100E	HREG_CARD_TYPE
0x100F	HREG_MODBUS_ADDR
…	Config-EEPROM frei
0x107F	
…	unbenutzt
0x2000	User-EEPROM
…	
0x2380	

Tabelle 4: Registerübersicht

8 Das Modbus-Protokoll unterscheidet vier Addressbereiche – *Inputregister*, *Holdingregister*, *Inputs* und *Coils*. Die beiden Reisterarten sind immer 16bit breit; *Inputs* und *Coils* bieten bitweisen Zugriff auf die Ressourcen. Während *Holdingregister* und *Coils* sowohl schreib- als auch lesbar sind, kann auf die anderen beiden Typen nur lesend zugegriffen werden. Jeder Typ besitzt einen 16bit-Addressraum, der sich logisch mit den Addressräumen anderer Typen überschneiden darf.

3.1.1 Control-Register – HREG_CTRL

Hier handelt es sich um ein Spezialregister, das der Konfiguration des gpio.Net *Core* dient. Auf Grund seiner Funktion, ist es gegen unbeabsichtigtes Schreiben durch einen Schlüssel geschützt.

HREG_CTRL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTRL_ACCESS_KEY - 0x8E								reserviert			CTRL_CONN0_INIT	CTRL_EEPROM_WREN	CTRL_CONFIG_WREN	CTRL_DEFAULT	CTRL_RESET

Tabelle 5: Kontrollregister

3.1.1.1 CTRL_ACCESS_KEY

Jegliche Änderungen an den anderen Flags dieses Registers werden nur dann wirksam, wenn gleichzeitig `CTRL_ACCESS_KEY` den gültigen Schlüssel (`0x8E`) enthält. Der Zugriff auf die unteren acht Bit erlischt nach diesem Schreibzugriff – `CTRL_ACCESS_KEY` muss folglich bei jedem Zugriff auf das Konfigurationsregister explizit gesetzt werden.

3.1.1.2 CTRL_CONN0_INIT

Das Setzen dieses Bits auf `1` bewirkt eine Initialisierung der seriellen Schnittstelle (RS485 / RS232) mit den entsprechenden Parametern. Dieses Flag dient auch der Konfiguration des, auf dieser Schnittstelle aktiven, Modbus-Modus'. Die Änderungen der Schnittstelle werden erst nach dem Beantworten des Modbus-Requests aktiv.

3.1.1.3 CTRL_EEPROM_WREN

Eine `0` sperrt Schreibzugriffe auf das User-EEPROM. Dieses Flag ändert seinen Zustand nach einem Schreibzugriff auf den Bereich `0x2000-0x237F` **nicht**. Das heißt, der Schreibzugriff bleibt nach dem Setzen auf `1` solange möglich, bis das Flag auf `0` zurückgesetzt wird (durch Schreibzugriff auf `HREG_CTRL` oder Reset). Initialwert des Flags ist `0`.

3.1.1.4 CTRL_CONFIG_WREN

Dieses Flag verhält sich analog zu `CTRL_EEPROM_WREN` mit dem Unterschied,

dass es die persistenten Konfigurations-Register `0x1000-0x107F` schützt. Der Initialwert ist `0`.

3.1.1.5 CTRL_DEFAULT

Das Aktivieren dieses Flags (Setzen auf `1`) löst eine Re-Initialisierung der persitenten Konfigurations-Register auf den Auslieferungszustand aus. Dieser Vorgang ändert jedoch nicht den Zustand der aktiven Konfiguration. Zum Übernehmen der Änderungen ist ein Reset nötig.

3.1.1.6 CTRL_RESET

Dieses Flag löst einen Neustart des Moduls aus, wenn es den Wert `1` annimmt. Zwischen dem Setzen des Flags und dem Reset vergehen mindestens 16ms. Das Setzen der Default-Konfiguration wird vor dem Neustart durchgeführt, wenn `CTRL_DEFAULT` entsprechend den Wert `1` besitzt.

3.1.2 Power-Management-Register – HREG_PM

Dieses Register ist zur Zeit unbenutzt. Es sollte mit Blick auf zukünftige Module nicht verwendet werden.

3.1.3 Baudraten-Konfiguration – HREG_BAUDSEL

Für die serielle Schnittstelle (RS485 / RS232) können Baudraten zwischen 1200 Baud bis 1 Mbit gewählt werden. Da `HREG_BAUDSEL` 16 Bit breit ist, wird anstelle der direkten Baudrate ihr Index in die nebestehende Tabelle verwendet. Indizies außerhalb des zulässigen Bereiches resultieren in der Standardbaudrate – 19200 Baud (Index `4`).

Die Konfiguration tritt erst durch Setzen des Flags `CTRL_CONN0_INIT` im Register `HREG_CTRL` in Kraft.

HREG_BAUDSEL

Value	Baudrate
0	1200
1	2400
2	4800
3	9600
4	19200
5	38400
6	57600
7	115200
8	230400
9	250000
10	500000
11	1000000

Tabelle 6: Baudraten

3.1.4 Datenbreite – HREG_DBITS

Die verwendete Bitbreite der, über die serielle Schnittstelle transferierten, Bytes wird durch dieses Register bestimmt. Gültige Werte sind `7` und `8` Bit. Das Register deckt somit die für Modbus nötigen Bitbreiten ab. Werte ausserhalb dieses Bereiches führen zur Verwendung der Standardeinstellung – `8` Bit.

Die neue Datenbreite wird erst nach dem Reset der seriellen Verbindung verwendet.

3.1.5 Parität – HREG_PARITY

Ob und welche Art Paritätsprüfung auf der seriellen Verbindung Verwendung findet entscheidet der Inhalt des Registers `HREG_PARITY`. Gültige Werte finden sich in der nebenstehenden Tabelle, der Defaultwert ist `2` (even).

HREG_PARITY

Value	Parity
0	none
1	odd
2	even

Tabelle 7: Parität

Es ist ein Reset der seriellen Verbindung nötig, um Änderungen der Paritätsprüfung zu aktivieren.

3.1.6 Anzahl der Stoppbits – HREG_STOP

Jedes Datenbyte wird auf der seriellen Schnittstelle (RS232 / RS485) mit mindestens einem Stoppbit beendet. Eine Verlängerung dieser Pause um maximal ein weiteres Stoppbit ist möglich. Das Register `HREG_STOP` enthält zu diesem Zweck die Anzahl der **zusätzlichen** Stoppbits. Zum Übernehmen der Änderung ist eine Reinitialisierung der Schnittstelle nötig. Der Standardwert dieses Registers ist `0`.

3.1.7 Wahl des Schnittstellenmodus' – HREG_SERIAL_MODE

Die serielle Schnittstelle kann entweder als RS232 oder als RS485 verwendet werden. `HREG_SERIAL_MODE` dient der Wahl dieses Modus'. Standard ist `1` (RS485). Der Modus wird nach dem Reset der Schnittstelle aktiv.

HREG_SERIAL_MODE

Value	Mode
0	RS232
1	RS485

Tabelle 8: Schnittstellen-modus

3.1.8 Wahl des Modbus-Modus' - HREG_MODBUS_MODE

Modbus kennt zwei grundlegende Modi – Modbus-RTU und Modbus-ASCII. Der Pflichtmodus RTU (`1`) ist bei der gpio.NET-Familie der Standardfall. Alternativ steht der ASCII-Modus (`0`) zur Verfügung.

HREG_MODBUS_MODE

Value	Mode
0	ASCII
1	RTU

Tabelle 9: Modbus-Modus

Modbus-ASCII bietet bei höheren Übertragungsraten und kurzen Paketen mehr Performance und ist darüber hinaus besser für Debugging und Logging geeignet. Ein Reset der Schnittstelle ist zum Übernehmen des neuen Modus nötig.

3.2 Persistente EEPROM-Konfiguration

Die *Core*-Register besitzen, mit Ausnahme von `HREG_CTRL`, frei konfigurierbare Defaultwerte im EEPROM (`0x1000-0x100F`). Jedes Register läd von seiner Addresse plus dem Offset `0x1000` nach dem Reset seine Initialkonfiguration. Ist das EEPROM unkonfiguriert, werden Produktionsdefaults geladen. Über

HREG_CTRL können diese Produktionsdefaults in die entsprechenden Register und deren EEPROM-Pendants geschrieben werden.

Zwei Spezialregister – HREG_CARD_TYPE und HREG_MODBUS_ADDR – existieren nur im EEPROM. HREG_CARD_TYPE gibt Auskunft über die Bestückung der gpio.NET-Karte und bewirkt, dass sich die Firmware gegebenenfalls entsprechend verhält. Ob und wie sich dieses Register auswirkt findet sich applikationsspezifischen Teil dieser Dokumentation. Die Wirkungsweise von HREG_MODBUS_ADDR ist im Kapitel **Inbetriebnahme** erläutert.

3.3 Setzen der Device-Addresse – Write By Serial

Die gpio.NET-Module bieten die Möglichkeit während des Busbetriebs eine neue Modbusadresse einzustellen. Hierfür ist allein die Seriennummer der entsprechenden Karte nötig. Diese Funktion stellt eine Erweiterung des Modbus-Standards dar.

Byte	Name	Example
0x00	Function code	0x41
0x01 – 0x0A	Serial number (ASCII)	000034F30C10
0x0B – 0x0C	New device address	0x0001 – 0x00F7

Tabelle 10: Write-By-Serial - Modbus Frame

4 Beschreibung gpio.AI

Holding Regs

0x0100	Samplerate
0x0101	Sample EXP
0x0102	Triggerfrequenz

Input Regs

0x0000	Daten – Kanal 0
0x0001	Daten – Kanal 1
0x0002	Daten – Kanal 2
0x0003	Daten – Kanal 3
0x0004	Daten – Kanal 4
0x0005	Daten – Kanal 5
0x0006	Daten – Kanal 6
0x0007	Daten – Kanal 7
0x0008	Ringbuffer RB_START
0x0009	Ringbuffer RB_CNT
0x000A	Ringbuffer RB_LOOP
0x000B	Ringbuffer DATA
…	
0x0082	Ringbuffer DATA

Tabelle 11: Registerlayout gpio.AI

4.1 Normaler Modus

Im Normalbetrieb wird durch das Lesen eines Datenregisters $Data_n$ (Inputregister `0x0000-0x0007`) der Samplevorgang angestoßen. Der gesampelte Wert wird dann als $Data_n$ ausgelesen. Über das Holdingregister `0x0100` lässt sich Samplerate in Hz einstellen. In diesem Modus hat die Samplerate hat direkten Einfluss auf die Antwortzeit des Moduls.

4.2 Automatische Mittelung

Holdingregister `0x0101` (Sample_EXP) bietet die Möglichkeit mehrere Samples zu mitteln bevor die Ausgabe über $Data_n$ erfolgt. Die Anzahl der zur Mittelung herangezogenen Samples beträgt 2^{Sample_EXP}. Defaultwert des Registers ist 0 (1 Sample). Im Normalbetrieb hat die Mittlungstiefe direkten Einfluss auf die Reaktionszeit des Moduls. Das heißt, es werden erst alle Samples mit der konfigurierten Samplerate erfasst, dann gemittelt und schließlich wird das Ergebnis über $Data_n$ zurückgeliefert.

4.3 Autotrigger

Die Messung einer Sequenz zeitlich äquidistanter Werte lässt sich über den Autotrigger realisieren. Holdingregister `0x0102` gibt dafür die verwendete Triggerfrequenz in Hz an. Ein Wert von 0 (default) schaltet den Autotrigger ab.

In diesem Modus stößt ein interner Timer den Messvorgang an. Daten werden analog zum Normalbetrieb unter Verwendung der Register Samplerate und Sample_EXP erzeugt, jedoch nicht in den Inputregistern $Data_0$ bis $Data_7$ gespeichert. Stattdessen werden die erfassten (und gemittelten) Daten im Datenbereich (Inputregister `0x000B-0x0082`) eines internen Ringbuffers

abgelegt. Der Ringbuffer verfügt außerdem über drei Counter (RB_START, RB_CNT, RB_LOOP), die Auskunft über den aktuellen Zustand des Buffers geben. RB_CNT (Inputregister `0x0009`) wird mit jedem Triggerevent inkrementiert. Überschreitet RB_CNT die Maximalgröße des Buffers (15*8 = 120 Werte), verschiebt sich RB_START (Inputregister `0x0008`) und alte Daten werden überschrieben. Das Register RB_LOOP (Inputregister `0x000A`) zählt die Anzahl voller Buffer (RB_START erreicht wieder 0) und kann somit als Indikator für Datenverlust dienen.

Im Autotrigger-Modus werden immer alle acht Kanäle gesampelt. Die Größe des Ringbuffers ist so gewählt, dass alle Daten inklusive der drei Counter in einem Modbuspaket Platz finden. Dadurch kann der Buffer immer komplett gelesen und Synchronisierungsfehler vermieden werden. Der Buffer enthält maximal die letzten 15 Messwerte für jeden Kanal (120 Werte). Die Werte werden dabei sequentiell abgelegt. Ein Messwert sei als $X_{y;z}$ definiert. Hiebei steht y für dea Triggerereignis und z für den Eingangskanal. Im Buffer findet sich entsprechend folgende Sequenz:

$$\mathbf{[X_{0;0}, X_{0;1} \ldots X_{0;7}], [X_{1;0}, X_{1;1} \ldots X_{1;7}] \ldots [X_{14;0}, X_{14;1} \ldots X_{14;7}]}$$

In diesem Modus ändert sich das Verhalten der Inputregister $Data_0$ – $Data_7$. Diese stoßen nun keine Messung an, sondern liefern beim Auslesen den Mittelwert der im Rinbuffer befindlichen Daten für den jeweiligen Kanal.

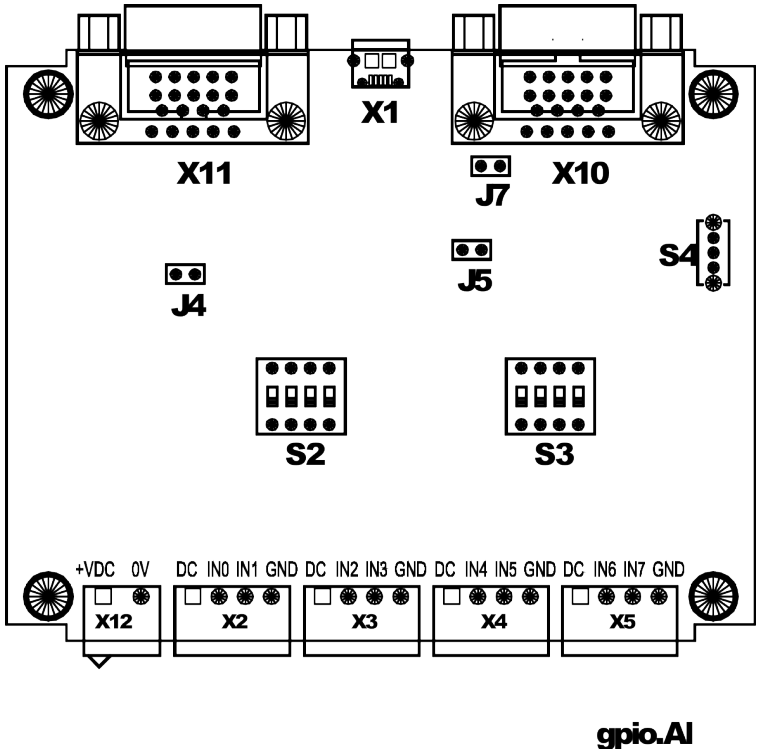
5 Steckerbelegung

Kommunikation:

X1: Micro-USB Device

DSub-9 (nur R-Version)

Pin	X11 (female)	X12 (male)
1	Future Appl.	Future Appl.
2	TXD	
3	RXD	
4	RS485+	RS485+
5	GND	GND
6	RS485-	RS485-
7	(RTS)	
8	(CTS)	
9	VBUS	VBUS



Opionale Spannungsversorgung (X12, nur R-Version)

+VDC	+7,5V....24V
0V	0V / GND

Analogeingänge:

Stecker X2

DC	Output +24V / +VDC
IN0	0...4V / 0...20mA
IN1	0...4V / 0...20mA
GND	Common-

Stecker X3

DC	Output +24V / +VDC
IN2	0...4V / 0...20mA
IN3	0...4V / 0...20mA
GND	Common-

Stecker X4

DC	Output +24V / +VDC
IN4	0...4V / 0...20mA
IN5	0...4V / 0...20mA
GND	Common-

Stecker X5

DC	Output +24V / +VDC
IN6	0...4V / 0...20mA
IN7	0...4V / 0...20mA
GND	Common-

6 Technische Daten

Allgemein

Maße (ohne Stecker, ohne optionale Frontplatte): 80x100x13 mm (halbe Europakarte)
Zulässige Betriebstemperatur: -10°C bis +70°C (kurzfrstig -40°C bis +85°C)
Maximal zulässige relative Luftfeuchtigkeit: 75%

Betriebsspannung über USB, VBUS (Schalter S4 auf "5V"): 4,5 Volt – 5,5 Volt

Stromaufnahme über USB/VBUS (Eigenverbrauch): ca. 70 mA

Die Stromaufnahme über USB/VBUS ist abhängig von der zusätzlichen Gesamtlast an den DC-Ausgängen. Bei reinem Betrieb über USB sollten 500 mA nicht überschritten werden (USB-Spezifiaktion). Gemischter Betrieb ist möglich.

Maximaler Verbrauch über USB/VBUS bei 600mA Last an DC: 1500 mA

DC-Ausgang (X2, X3, X4, X5): +24 ±0.5 Volt
$I_{max(X2)} = I_{max(X3)} = I_{max(X4)} = I_{max(X5)}$: 140 mA, U_{Out} 24 ±0.5 Volt
$I_{max\ (X2+X3+X4+X5)}$: 140 mA, U_{Out} 24 ±0.5 Volt
$I_{max\ (X2+X3+X4+X5)}$: 560 mA, 9 Volt ÷ $U_{Droupout}$ = 15 Volt

Betriebsspannung über +VDC (Schalter S4 auf "24V"): 7,5V....28V (max.30V)

Maximaler Eigenverbrauch bei +VDC = 7,5V: 60mA
Maximaler Eigenverbrauch bei +VDC = 24V: 20mA

$VBUS_{OUT}$: 5V ±0,2V
Maximale Belastbarkeit bei +VDC=7,5V: 0,5A (I_{VDC}=0,52 A)
Maximale Belastbarkeit bei +VDC=9V: 0,75A (I_{VDC}=0,64 A)
Maximale Belastbarkeit bei +VDC=12V: 1,0A (I_{VDC}=0,62 A)
Maximale Belastbarkeit bei +VDC=15V: 1,0A (I_{VDC}=0,48 A)
Maximale Belastbarkeit bei +VDC=24V: 1,5A (I_{VDC}=0,44 A)

Analoge Eingänge

Kanäle: 8 (mux), einzeln konfigurierbar für Spannungs- oder Strommessung
Auflösung: 16 Bit
Typische Ungenauigkeit: ± 1.5 LSB (1 Kanal, Samplefrequenz 1 Hz)
Maximale Ungenauigkeit: ± 10 LSB (8 Kanäle, max. Samplefrequenz)

Messbereich Spannung: 0...+4.096 Volt
Eingangswiderstand: > 5 MOhm
Übersteuerungsfestigkeit: max. 50 Volt

Messbereich Strom: 0....+20mA (20.48mA)
Lastwiderstand: 200 Ohm, ± 0.1% max.
Übersteuerungsfestigkeit: max. 30 mA (kontinuierlich)
max 100 mA (< 1s , duty cycle 1:10)

7 Konformitätserklärungen und Herstellerdaten

Die **taskit GmbH** bestätigt die für die Produkte **gpio.AI-R** und **gpio.AI-U** die EG-Konformität gemäß der EMV-Richtlinie (2004/108/EG).

Die t**askit GmbH** erklärt für die Produkte **gpio.AI-R** und **gpio.AI-U**, dass diese gemäß der EU-Richtline 2002/95/EG (RoHS) hergestellt wurden und insbesondere frei von folgenden Inhaltsstoffen sind:

1. Blei und seine Verbindungen
2. Quecksilber und seine Verbindungen
3. Kadmium und seine Verbindungen
4. Polybromierte Biphenyle (PBB)
5. Polybromierte Diphenylether (PBDE)
6. Chrom VI-Verbindungen

taskit GmbH, Groß-Berliner Damm 37, 12847 Berlin, Germany
Tel.:+49(0)30 6112950, Fax: +49(0)30 61129510