BECKHOFF New Automation Technology

Handbuch | DE

TS6255-0030

TwinCAT 2 | Modbus RTU BC

Supplement | Communication

09.05.2022 | Version: 1.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT®

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

Die TwinCAT SPS Bibliothek Modbus RTU bietet Funktionsbausteine zur seriellen Kommunikation mit Modbus-Endgeräten.

Typische Endgeräte sind Bedienterminals, die über eine serielle RS-232, RS-422 oder RS-485 Schnittstelle an eine TwinCAT Steuerung angeschlossen werden und über einen Modbus-Treiber verfügen. In diesem Fall ist die TwinCAT-SPS ein Modbus-Slave und das Bedienterminal ein Modbus-Master. Der Programmieraufwand auf der SPS-Seite ist in dieser Konfiguration sehr gering.

Alternativ stehen in der Bibliothek Modbus-Master-Funktionen zur Verfügung, mit denen die SPS ein oder mehrere Modbus-Slaves ansprechen kann. Diese Konfiguration ist seltener anzutreffen und auch weniger zu empfehlen, weil der Programmieraufwand höher ist.

Unterstützte TwinCAT Steuerungen

- TwinCAT PC
- CX1000 Serie
- Buscontroller BC und BX

Unterstützte Schnittstellen

- serielle Schnittstelle (COM-Port) eines PC oder CX
- serielle Schnittstelle eines BX Controllers
- serielle Busklemmen KL6001, KL6011, KL6021, KL6031 oder KL6041, entsprechende EL-Klemmen

Randbedingungen

Das Modbus-Protokoll definiert ein exaktes Timing, so sollen z. B. alle Zeichen eines Telegramms lückenlos übertragen werden. Da die Kommunikation Modbus RTU auf einer SPS-Steuerung realisiert wird, kann wegen der zyklischen Abarbeitung des SPS-Programms, dieses exakte Timing nicht garantiert werden. Die meisten Endgeräte sind sehr tolerant und verhalten sich problemlos, falls kurze Zeitlücken zwischen den Zeichen auftreten. Im Einzelfall muss das Verhalten des Endgerätes überprüft werden.

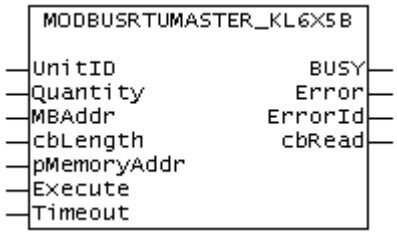
Bei einer EL60x2 ist der zweite Kanal nicht für ModbusRTU-Kommunikation geeignet, da dieser Low Prior bearbeitet wird und dadurch die Frames mit Lücken versendet, was wiederum die Gegenstelle als Frame Error detektieren könnte.

Hinweis : Bei einigen seriellen Schnittstellen Klemmen kann ein Interner Buffer vor dem Senden gefüllt werden (Option *kontinuierliches Senden*). Die Bibliothek ModbusRTU kann diese Funktion verwenden, wenn dies in der entsprechenden seriellen Klemme eingestellt ist. Zum Beispiel kann bei der KL6031 mit dem Konfigurationsbaustein *KL6configuration* der ContinuousMode aktiviert werden (Register 34 Bit 6). Damit werden dann bis zu 128 Byte in den internen Buffer der Busklemme gelegt und kontinuierlich gesendet.

Weiterführende Dokumentation

http://www.modicon.com/
http://www.modbus.org

3 ModbusRtuMaster_KL6x5B



Der Funktionsbaustein *ModbusRtuMaster_KL6x5B* realisiert einen Modbus-Master, der über eine serielle Busklemme KL6001, KL6011 oder KL6021 kommuniziert. Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Ein Beispielprogramm für einen https://infosys.beckhoff.com/content/1031/tcplclibmodbusrtubc/Resources/zip/455297291.zip verdeutlicht die Funktionsweise.

Unterstüzte Modbus-Funktionen (Aktionen

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*
  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden im komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*
  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden im komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* abgelegt.
- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*
  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*
  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*
  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*
  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*
  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*
  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*
  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort *MBAddr* übergeben. Eventuelle für die Funktion notwendige Daten werden über *pMemoryAddr* mitgegeben.

VAR_INPUT

```
VAR_INPUT
UnitID : BYTE;
Quantity : WORD;
MBAddr : WORD;
cbLength : UINT;
pMemoryAddr : DWORD;
Execute : BOOL;
Timeout : TIME;
END_VAR
```

UnitID [▶ 18]

UnitID: Modbus Stationsadresse [▶ 18] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.

Quantity: Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.

MBAddr: Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert.
Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.

cbLength : Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]*. *cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.

pMemoryAddr: Speicheradresse in der SPS, die mit ADR(ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.

Execute : Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.

Timeout : Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

VAR_OUTPUT

```
VAR_OUTPUT
BUSY : BOOL;
Error : BOOL;
ErrorId : MODBUS_ERRORS;
cbRead : UINT;
ND_VAR
```

Busy: Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.

Error: Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.

ErrorId: Zeigt eine Fehlernummer [▶ 21] im Falle einer gestörten oder fehlerhaften Kommunikation an.

cbRead: Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion

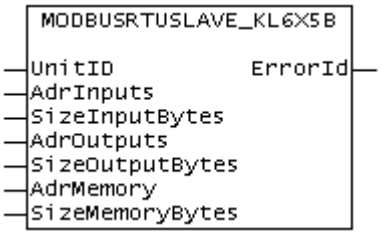
Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikationsport notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme. Auf einem Buscontroller BC müssen die I/O-Adressen manuell zugewiesen werden. Siehe Hardwarezuordnung am Buscontroller BC [▶ 22].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT ab V2.8	PC (i386), CX1000	ModbusRTU.lib (ab Version 2.2)
TwinCAT ab V2.8	Buscontroller BC	ModbusRTU.lb6 (ab Version 2.2)

4 ModbusRtuSlave_KL6x5B



Der Funktionsbaustein *ModbusRTUslave_KL6x5B* realisiert einen Modbus-Slave, der über eine serielle Busklemme KL6001, KL6011 oder KL6021 kommuniziert. Der Baustein verhält sich passiv, bis er von einem angeschlossenen Modbus-Master Telegramme empfängt.

Ein Beispielprogramm für einen https://infosys.beckhoff.com/content/1031/tcplclibmodbusrtubc/Resources/zip/455300235.zip verdeutlicht die Funktionsweise.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
AdrInputs : POINTER TO BYTE; (* Pointer to the Modbus input area *)
SizeInputBytes : UINT;
AdrOutputs : POINTER TO BYTE; (* Pointer to the Modbus output area *)
SizeOutputBytes : UINT;
AdrMemory : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
SizeMemoryBytes : UINT;
END_VAR
```

UnitID [▶ 18]

UnitID: Modbus Stationsadresse [▶ 18] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.

AdrInputs: Startadresse des Modbus-Input-Bereiches [▶ 19]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Input-Variable) berechnet werden.

SizeInputBytes: Größe des Modbus-Input-Bereiches in Bytes. Die Größe kann mit SIZEOF(Input-Variable) berechnet werden.

AdrOutputs : Startadresse des Modbus-Output-Bereiches [▶ 19]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Output-Variable) berechnet werden.

SizeOutputBytes: Größe des Modbus-Output-Bereiches in Bytes. Die Größe kann mit SIZEOF(Output-Variable) berechnet werden.

AdrMemory : Startadresse des Modbus-Memory-Bereiches [▶ 19]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Memory-Variable) berechnet werden.

SizeMemoryBytes : Größe des Modbus-Memory-Bereiches in Bytes. Die Größe kann mit SIZEOF(Memory-Variable) berechnet werden.

VAR_OUTPUT

```
VAR_OUTPUT
ErrorId : Modbus_ERRORS;
ND_VAR
```

ErrorId: Zeigt eine Fehlernummer [▶ 21] im Falle einer gestörten oder fehlerhaften Kommunikation an.

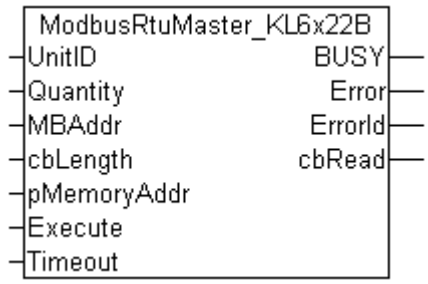
Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikationsport notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme. Auf einem Buscontroller BC müssen die I/O-Adressen manuell zugewiesen werden. Siehe Hardwarezuordnung am Buscontroller BC [▶ 22].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT ab V2.8	PC (i386), CX1000	ModbusRTU.lib
TwinCAT ab V2.8	Buscontroller BC	ModbusRTU.lb6

5 ModbusRtuMaster_KL6x22B



Der Funktionsbaustein *ModbusRtuMaster_KL6x22B* realisiert einen Modbus-Master, der über eine serielle Busklemme KL6031 oder KL6041 kommuniziert. Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Ein Beispielprogramm für einen Buscontroller BC (https://infosys.beckhoff.com/content/1031/tcplclibmodbusrtubc/Resources/zip/455297291.zip) verdeutlicht die Funktionsweise.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*
  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*
  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* abgelegt.
- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*
  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*
  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*
  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*
  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*
  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse *pMemoryAddr* zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*
  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*
  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort *MBAddr* übergeben. Eventuelle für die Funktion notwendige Daten werden über *pMemoryAddr* mitgegeben.

VAR_INPUT

```
VAR_INPUT
    UnitID          : UINT;
    Quantity        : WORD;
    MBAddr          : WORD;
    cbLength        : UINT;
    pMemoryAddr     : DWORD;
    Execute         : BOOL;
    Timeout         : TIME;
END_VAR
```

UnitID: Modbus Stationsadresse (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.

Quantity: Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.

MBAddr: Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert.
Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.

cbLength : Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]*. *cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.

pMemoryAddr: Speicheradresse in der SPS, die mit ADR(ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.

Execute : Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.

Timeout : Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Busy: Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.

Error: Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.

ErrorId: Zeigt eine Fehlernummer im Falle einer gestörten oder fehlerhaften Kommunikation an.

cbRead: Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion

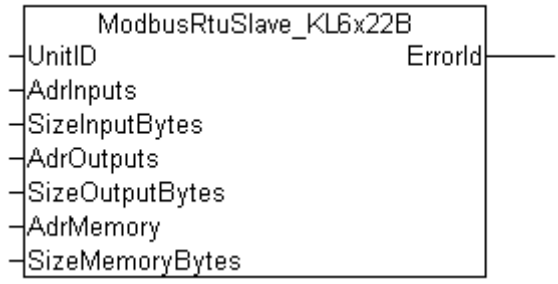
Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikationsport notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem Buscontroller BC müssen die I/O-Adressen manuell zugewiesen werden. Siehe Hardwarezuordnung am Buscontroller BC.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT ab V2.8	PC (i386), CX1000	ModbusRTU.lib (ab Version 2.2)
TwinCAT ab V2.8	Buscontroller BC	ModbusRTU.lb6 (ab Version 2.2)

6 ModbusRtuSlave_KL6x22B



Der Funktionsbaustein *ModbusRTUslave_KL6x22B* realisiert ein Modbus-Slave, der über eine serielle Busklemme KL6031 oder KL6041 kommuniziert. Der Baustein verhält sich passiv, bis er von einem angeschlossenen Modbus-Master Telegramme empfängt.

Ein Beispielprogramm für einen Buscontroller BC (https://infosys.beckhoff.com/content/1031/tcplclibmodbusrtubc/Resources/zip/455300235.zip) verdeutlicht die Funktionsweise.

VAR_INPUT

```
VAR_INPUT
    UnitID          : UINT;
    AdrInputs       : POINTER TO BYTE; (* Pointer to the Modbus input area  *)
    SizeInputBytes  : UINT;
    AdrOutputs      : POINTER TO BYTE; (* Pointer to the Modbus output area *)
    SizeOutputBytes : UINT;
    AdrMemory       : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
    SizeMemoryBytes : UINT;
END_VAR
```

UnitID: Modbus Stationsadresse (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.

AdrInputs: Startadresse des Modbus-Input-Bereiches. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Input-Variable) berechnet werden.

SizeInputBytes: Größe des Modbus-Input-Bereiches in Bytes. Die Größe kann mit SIZEOF(Input-Variable) berechnet werden.

AdrOutputs : Startadresse des Modbus-Output-Bereiches [▶ 19]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Output-Variable) berechnet werden.

SizeOutputBytes: Größe des Modbus-Output-Bereiches in Bytes. Die Größe kann mit SIZEOF(Output-Variable) berechnet werden.

AdrMemory : Startadresse des Modbus-Memory-Bereiches [▶ 20]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Memory-Variable) berechnet werden.

SizeMemoryBytes : Größe des Modbus-Memory-Bereiches in Bytes. Die Größe kann mit SIZEOF(Memory-Variable) berechnet werden.

VAR_OUTPUT

```
VAR_OUTPUT
    ErrorId : Modbus_ERRORS;
END_VAR
```

ErrorId: Zeigt eine Fehlernummer im Falle einer gestörten oder fehlerhaften Kommunikation an.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikationsport notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem Buscontroller BC müssen die I/O-Adressen manuell zugewiesen werden. Siehe Hardwarezuordnung am Buscontroller BC.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT ab V2.8	PC (i386), CX1000	ModbusRTU.lib (ab Version 2.0)
TwinCAT ab V2.8	Buscontroller BC	ModbusRTU.lb6 (ab Version 2.0)

7 Modbus Stationsadresse

Modbus definiert gültige Stationsadressen im Bereich 1 bis 247. Ein Modbus-Slave antwortet nur auf Telegramme, die seine eigene Adresse enthalten. Die Adresse 0 ist keine gültige Stationsadresse, sondern wird für Broadcast-Telegramme an alle Stationen verwendet, die nicht beantwortet werden. Die Adressen 248 bis 255 sind reserviert.

Die Bibliothek ModbusRTU definiert weitere Sammeladressen. Dadurch wird es möglich, eine Station auf mehrere Adressen antworten zu lassen.

```
TYPE MODBUS_UNITID :
(
MODBUS_UNITID_BROADCAST := 0,
MODBUS_UNITID_ALLVALID := 256, (* response on address 1..247 *)
MODBUS_UNITID_ALLBUTBROADCAST := 257, (* response on address 1..255 *)
MODBUS_UNITID_ALL := 258 (* response on address 0..255 *)
);
END_TYPE
```

8 Modbus Adressbereiche

Modbus definiert Zugriffsfunktionen für verschiedene Datenbereiche. Diese Datenbereiche werden in einem TwinCAT SPS-Programm als Variablen, beispielsweise als Word-Arrays, deklariert und dem Modbus-Slave-Funktionsbaustein als Eingangsparameter übergeben. Um die Bereiche eindeutig zu unterscheiden, hat jeder Bereich eine andere Modbus-Startadresse. Dieser Offset muss bei der Adressierung berücksichtigt werden.

Inputs

Der Datenbereich *Inputs* beschreibt üblicherweise die physikalischen Eingangsdaten, auf die nur lesend zugegriffen werden kann. Das können digitale Eingänge (Bit) oder Analogeingänge (Wort) sein. Es liegt in der Hand des SPS-Programmierers, ob er dem Kommunikationspartner den direkten Zugriff auf die physikalischen Eingänge erlauben möchte. Es ist ebenfalls möglich, für die Modbus-Kommunikation einen Input-Bereich zu definieren, der nicht mit den physikalischen Eingängen identisch ist:

- Definition der Modbus-Input-Daten als direktes Abbild der physikalischen Eingänge. Anfang und Größe des Datenbereichs können frei festgelegt werden und sind durch die reale Größe des Eingangsprozessabbildes der verwendeten Steuerung begrenzt.

  VAR Inputs AT%IW0 : ARRAY[0..255] OF WORD; END_VAR

- Definition der Modbus-Input-Daten als separater Modbus-Datenbereich unabhängig von den physikalischen Eingängen

  VAR Inputs : ARRAY[0..255] OF WORD; END_VAR

Der Zugriff auf den *Input*-Bereich durch einen Modbus-Master ist mit folgenden Modbus-Funktionen möglich:

```
2 : Read Input Status
4 : Read Input Registers
```

Adressierung

Der *Input*-Bereich wird mit einem Offset 0 adressiert, das heißt, dass die im Telegramm übertragene Adresse 0 das erste Element im Input-Datenbereich anspricht.

Beispiele:

SPS-Variable	Zugriffsart	Adresse im Modbus-Telegramm	Adresse im Endgerät (Geräte-abhängig)
Inputs[0]	Wort	16#0	30001
Inputs[1]	Wort	16#1	30002
Inputs[0], Bit 0	Bit	16#0	10001
Inputs[1], Bit 14	Bit	16#1E	1001F

Outputs

Der Datenbereich *Outputs* beschreibt üblicherweise die physikalischen Ausgangsdaten, auf die lesend und schreibend zugegriffen werden kann. *Outputs* können digitale Ausgänge (Coils) oder Analogausgänge (Output Register) sein. Wie bei den *Inputs* ist kann der Bereich als physikalische Ausgangsvariable oder als einfache Variable deklariert werden.

- Definition der Modbus-Output-Daten als direktes Abbild der physikalischen Ausgänge. Anfang und Größe des Datenbereichs können frei festgelegt werden und sind durch die reale Größe des Ausgangsprozessabbildes der verwendeten Steuerung begrenzt.

  VAR Outputs AT%QW0 : ARRAY[0..255] OF WORD; END_VAR

- Definition der Modbus-Output-Daten als separater Modbus-Datenbereich unabhängig von den physikalischen Ausgängen

  VAR Outputs : ARRAY[0..255] OF WORD; END_VAR

Der Zugriff auf den *Output*-Bereich durch einen Modbus-Master ist mit folgenden Modbus-Funktionen möglich:

```
1 : Read Coil Status
3 : Read Holding Registers
5 : Force Single Coil
6 : Preset Single Register
15 : Force Multiple Coils
16 : Preset Multiple Registers
```

Adressierung

Der *Output*-Bereich wird mit einem Offset 16#800 adressiert, das heißt, dass die im Telegramm übertragene Adresse 16#800 das erste Element im Output-Datenbereich anspricht.

Beispiele:

SPS-Variable	Zugriffsart	Adresse im Modbus-Telegramm	Adresse im Endgerät (Geräte-abhängig)
Outputs[0]	Wort	16#800	40801
Outputs[1]	Wort	16#801	40802
Outputs[0], Bit 0	Bit	16#800	00801
Outputs[1], Bit 14	Bit	16#81E	0081F

Memory

Der Datenbereich *Memory* beschreibt einen SPS Variablenbereich ohne physikalische I/O-Zuordnung.

- Definition der Modbus-Memory-Daten als SPS-Merker. Anfang und Größe des Datenbereichs können frei festgelegt werden.

  VAR Memory AT%MW0 : ARRAY[0..255] OF WORD; END_VAR

- Definition der Modbus-Memory-Daten als Variable ohne Merkeradresse

  VAR Memory : ARRAY[0..255] OF WORD; END_VAR

Der Zugriff auf den *Memory*-Bereich durch einen Modbus-Master ist mit folgenden Modbus-Funktionen möglich:

```
3 : Read Holding Registers
6 : Preset Single Register
16 : Preset Multiple Registers
```

Adressierung

Der *Memory*-Bereich wird mit einem Offset 16#4000 adressiert, das heißt, dass die im Telegramm übertragene Adresse 16#4000 das erste Wort im Output-Datenbereich anspricht.

Beispiele:

SPS-Variable	Zugriffsart	Adresse im Modbus-Telegramm	Adresse im Endgerät (Geräte-abhängig)
Memory[0]	Wort	16#4000	44001
Memory[1]	Wort	16#4001	44002

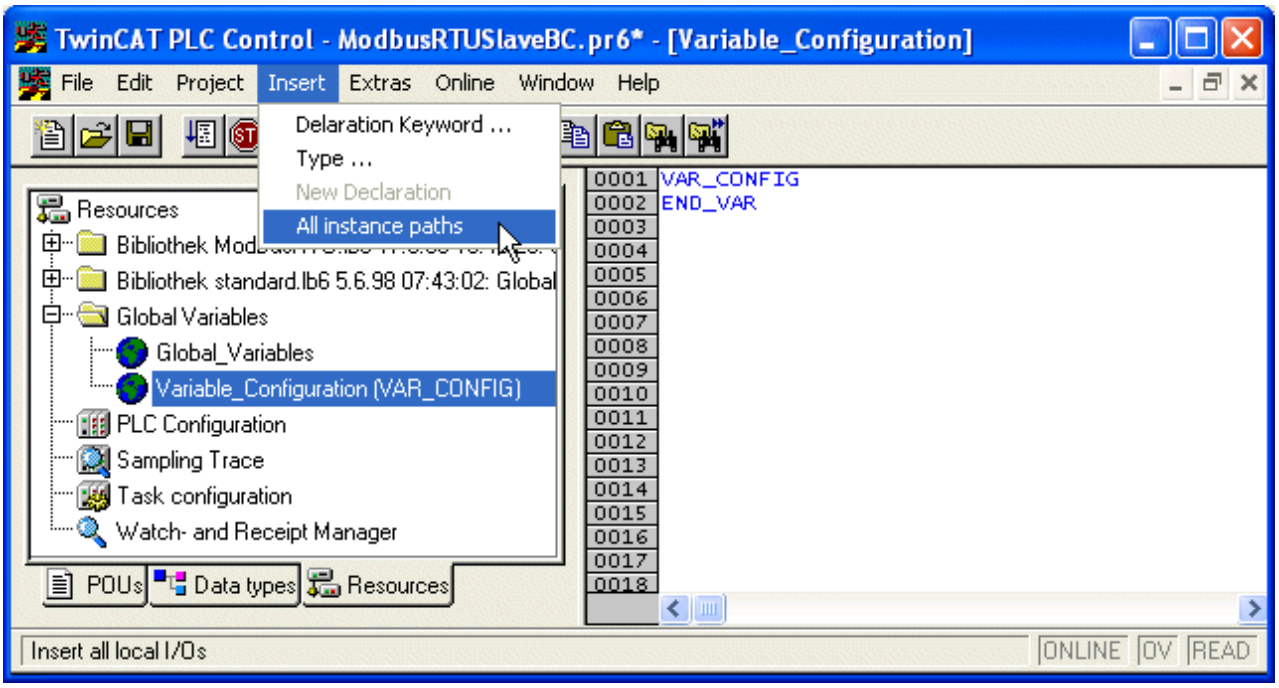
9 Modbus RTU Fehlernummern

```
TYPE MODBUS_ERRORS :
(
(* Modbus communication errors *)
MODBUSERROR_NO_ERROR, (* 0 *)
MODBUSERROR_ILLEGAL_FUNCTION, (* 1 *)
MODBUSERROR_ILLEGAL_DATA_ADDRESS, (* 2 *)
MODBUSERROR_ILLEGAL_DATA_VALUE, (* 3 *)
MODBUSERROR_SLAVE_DEVICE_FAILURE, (* 4 *)
MODBUSERROR_ACKNOWLEDGE, (* 5 *)
MODBUSERROR_SLAVE_DEVICE_BUSY, (* 6 *)
MODBUSERROR_NEGATIVE_ACKNOWLEDGE, (* 7 *)
MODBUSERROR_MEMORY_PARITY, (* 8 *)
MODBUSERROR_GATEWAY_PATH_UNAVAILABLE, (* A *)
MODBUSERROR_GATEWAY_TARGET_DEVICE_FAILED_TO_RESPOND, (* B *)

(* additional Modbus error definitions *)
MODBUSERROR_CHARREC_TIMEOUT := 16#20, (* 20 hex *)
MODBUSERROR_ILLEGAL_DATA_SIZE, (* 21 hex *)
MODBUSERROR_ILLEGAL_DEVICE_ADDRESS, (* 22 hex *)
MODBUSERROR_ILLEGAL_DESTINATION_ADDRESS, (* 23 hex *)
MODBUSERROR_ILLEGAL_DESTINATION_SIZE, (* 24 hex *)
MODBUSERROR_NO_RESPONSE, (* 25 hex *)

(* Low level communication errors *)
MODBUSERROR_TXBUFFOVERRUN := 102, (* 102 *)
MODBUSERROR_SENDTIMEOUT := 103, (* 103 *)
MODBUSERROR_DATASIZEOVERRUN := 107, (* 107 *)
MODBUSERROR_STRINGOVERRUN := 110, (* 110 *)
MODBUSERROR_INVALIDPOINTER := 120, (* 120 *)
MODBUSERROR_CRC := 150, (* 150 *)

(* High level PLC errors *)
MODBUSERROR_INVALIDMEMORYADDRESS := 232, (* 232 *)
MODBUSERROR_TRANSMITBUFFERTOOSMALL (* 233 *)
);
END_TYPE
```

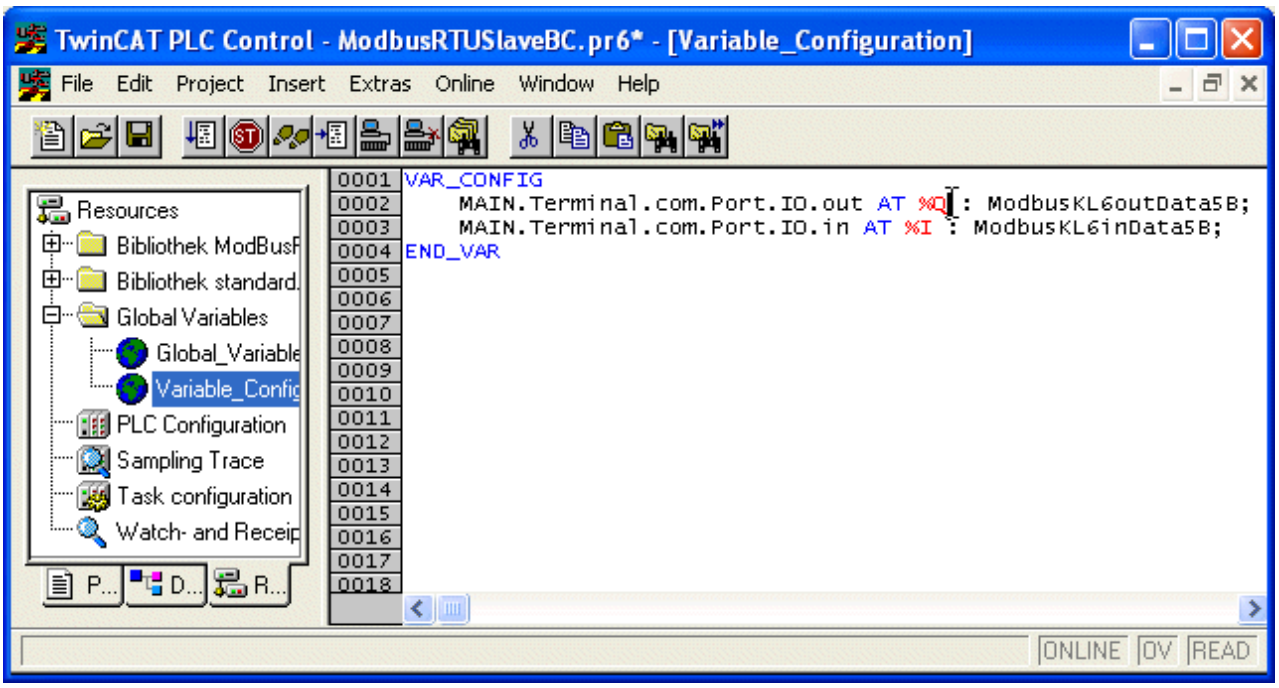
10 Hardwarezuordnung am Buscontroller BC

Bei einer PC-Steuerung oder einer CX1000-Steuerung wird die verwendetet serielle Schnittstelle im TwinCAT System Manager mit dem Modbus-RTU Funktionsbaustein verknüpft. Im Gegensatz dazu wird in einem Programm für einen Buscontroller BC diese Zuordnung direkt in der SPS-Programmierumgebung manuell durchgeführt.

Nachdem im SPS-Programm eine Instanz eines Modbus-RTU Funktionsbausteins angelegt wurde, wird die Hardwareadresse der anzusprechenden seriellen Busklemme in der *Variablenkonfiguration* vergeben. Dazu wird im Bereich *Resources* im Baustein *Variable_Configuration (VAR_CONFIG)* der Menüpunkt *Insert - All Instance Paths* gewählt.



Durch Wahl dieses Menüpunktes werden die in allen Funktionsbausteinen verwendeten lokalen I/O-Adressen aufgelistet.



Die Adresse der Variablen fehlt zunächst und muss nun entsprechend Busklemmenbestückung des Buscontrollers manuell zugewiesen werden. Falls die Busklemme als erste nicht-digitale Klemme am Buscontroller gesteckt ist, lautet die Adresse beispielsweise Null.

```
VAR_CONFIG
MAIN.Terminal.com.Port.IO.out AT %QB0 : ModbusKL6outData5B;
MAIN.Terminal.com.Port.IO.in AT %IB0 : ModbusKL6inData5B;
END_VAR
```

11 Klemmenkonfiguration

Die Busklemmen KL6001, KL6011, KL6021, KL6031 und KL6041 können mit der Konfigurationssoftware KS2000 parametriert werden. Alternativ ist auch eine Konfiguration über SPS-Bausteine möglich, die in der seriellen Kommunikationsbibliothek ComLib.lib enthalten sind. Für den Fall, dass die serielle Kommunikationsbibliothek nicht zusammen mit der Modbus-RTU-Bibliothek verwendet wird, kann die Basis-Bibliothek *KL6config.lib* eingebunden werden, die mit der Modbus-RTU-Bibliothek mitgeliefert wird. Diese Bibliothek enthält folgende Bausteine aus der seriellen Kommunikationsbibliothek.

- KL6configuration
- KL6ReadRegisters
- KL6WriteRegisters
- ComReset

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT ab V2.8	PC (i386), CX1000	ComLibV2.lib oder alternativ KL6config.lib
TwinCAT ab V2.8	Buscontroller BC	ComLibV2.lb6 oder alternativ KL6config.lb6

Mehr Informationen:
www.beckhoff.de/ts6255

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de