ACS 140/400 **Modbus-Protokoll**

Modbus-Protokoll

3AFY 64200534 R0103
DE
Gültig ab: 1.5.1998

Inhaltsverzeichnis

Modbus-Protokoll 1

Vorstellung des Modbus-Protokolls. 1
Datenübertragung über Modbus-Netzwerke. 1

Zwei serielle Übertragungsarten 3
RTU-Modus 3

Modbus-Telegrammblöcke 5
RTU-Telegrammblöcke. 5
Verwendung des Adressfelds 5
Verwendung des Funktionsfelds. 6
Inhalt des Datenfelds 6
Inhalt des Fehlerprüffelds 7
Serielle Übertragung von Zeichen 8

Fehlerprüfverfahren. 9
Paritätsprüfung 9
Zyklische Fehlerprüfung (CRC) 10

Modbus-Funktionsformate 11
Wiedergabe numerischer Werte 11
Datenadressen in Modbus-Telegrammen 11
Feldinhalt von Modbus-Telegrammen 11

Functionscodes 13
03 Halteregister lesen. 13
06 Einzelregister voreinstellen 15
16 (10 Hexadezimal) Mehrere Register voreinstellen. 16

Ausnahmeantworten 17

Zyklische Fehlerprüfung (CRC). 21
CRC in das Telegramm einbetten 22
Beispiel 22

Modbus-Protokoll

In diesem Handbuch wird das serielle Modbus-Protokoll erläutert. Das Handbuch ist für Anwender vorgesehen, die einen Modbus-Master für ihr eigenes Steuersystem programmieren müssen.

Modicon besitzt des Urheberrecht auf den Inhalt dieses Kapitels, der mit Genehmigung von Schneider Automation (Modicon) wiedergegeben wird. Das Kapitel ist Teil des Handbuchs *Modicon Modbus Protocol Reference Guide* (PI-MBUS-300 Rev. E).

Vorstellung des Modbus-Protokolls

Das Modbus-Protokoll ist ein serielles Master-Slave-Protokoll. In diesem Anhang wird das Modbus-Protokoll nur soweit erläutert, wie dies für den vollständigen Zugriff auf Antriebe des Typs ACS 140 und ACS 400 erforderlich ist. Das Modbus-Protokoll definiert, welche Daten über den seriellen Kommunikationsanschluß übertragen werden. Als physische Schnittstelle für Antriebe des Typs ACS 140/400 ist ein Halbduplex vorgesehen. Beim ACS 140 ist ein RS485/232-Adapter erforderlich; je nach Adaptereinstellung entspricht der Spannungspegel entweder RS485 oder RS232. Der ACS 400 besitzt standardmäßig einen seriellen RS485-Anschluß; der RS 485/232-Adapter ist nur dann erforderlich, wenn ein R232-Bus verwendet wird.

Datenübertragung über Modbus-Netzwerke

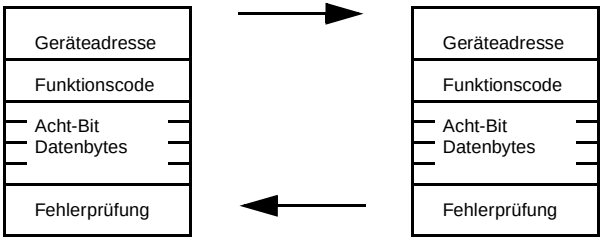
Die Standard-Modbus-Anschlüsse von Modicon-Controllern sind mit einer RS232C-kompatiblen serielle Schnittstelle verbunden, die die Belegung der Steckverbinderpins, die Kabelanschlüsse, die Signalpegel, die Baudrate und die Paritätsprüfung definiert. Die Controller können direkt oder via Modem in ein Netzwerk eingebunden werden.

Die Controller verwenden bei der Datenübertragung das Master-Slave-Prinzip, bei dem nur jeweils ein einzelnes Gerät (Master) eine Datenübertragung (Abfragen) einleitet. Die anderen Geräte (Slaves) reagieren und liefern die abgefragten Daten zum Master oder führen die vom Master angeforderte Aktion aus. Zu den typischen Master-Geräten zählen Hostrechner und Programmiergeräte. Zu den typischen Slaves zählen programmierbare Controller.

Der Master kann einzelne Slaves adressieren oder ein allgemeines Telegramm an alle Slaves senden. Slaves senden ein Telegramm (Antwort) auf Abfragen zurück, die einzeln an sie adressiert wurden. Auf allgemeine Abfragen, die vom Master übertragen wurden, werden keine Antworten zurückgesandt.

Das Modbus-Protokoll definiert das Format für die Abfragen vom Master, indem die Geräteadresse (oder Sendeadresse), ein Funktionscode zur Bestimmung der verlangten Aktion, alle zu übertragenden Daten und ein Fehlerprüffeld in das Protokoll eingetragen werden. Das Antworttelegramm der Slaves wird auch mit Hilfe des Modbus-Protokolls festgelegt. Es enthält Felder für die Bestätigung der ausgeführten Aktion, alle zurückzusendenden Daten und ein Fehlerprüffeld. Falls beim Empfang des Telegramms ein Fehler auftritt oder falls der Slave die angeforderte Aktion nicht ausführen kann, wird vom Slave ein Fehlertelegramm zurückgeschickt.

Abfragetelegramm vom Master



Antworttelegramm vom Slave

Abbildung 1 Abfrage-Antwort-Zyklus von Master und Slave

Die Abfrage: Der Funktionscode in der Abfrage teilt dem adressierten Slave-Gerät mit, welche Aktion durchzuführen ist. Die Datenbytes enthalten alle Zusatzinformationen, die der Slave zur Ausführung der Aktion benötigt. Funktionscode 03 fordert den Slave zum Beispiel auf, Halteregister auszulesen und deren Inhalt zurückzusenden. Das Datenfeld muß die Information enthalten, mit welchem Register begonnen werden soll und wieviele Register auszulesen sind. Mit Hilfe des Fehlerprüffelds kann der Slave die Vollständigkeit des Telegramminhalts feststellen.

Die Antwort: Wenn der Slave eine normale Antwort zurücksendet, entspricht der in der Antwort enthaltene Funktionscode dem Funktionscode in der Abfrage. Die Datenbytes enthalten die vom Slave gesammelten Daten, wie zum Beispiel Registerwerte oder den Status. Wenn ein Fehler auftritt, wird der Funktionscode geändert um anzuzeigen, daß die Antwort eine Fehlerantwort ist; die Datenbytes enthalten in diesem Fall einen Code, der den Fehler beschreibt. Mit Hilfe des Fehlerprüffelds kann der Master feststellen, ob der Telegramminhalt gültig ist.

Zwei serielle Übertragungsarten

Controller können zur Datenübertragung in Standard-Modbus-Netzwerken auf eine der folgenden Übertragungsarten eingestellt werden: ASCII oder RTU. Der Benutzer wählt neben der gewünschten Übertragungsart die Datenübertragungsparameter (Baudrate, Parität usw.) des seriellen Anschlusses. Die Übertragungsart und die Parameter *müssen für alle Geräte in einem Modbus-Netzwerk gleich sein*. Die Wahl von ASCII oder RTU ist nur bei Standard-Modbus-Netzwerken möglich. Dadurch wird der Bit-Inhalt von Telegrammfeldern definiert, die seriell in diesen Netzwerken übertragen werden. Durch die Wahl wird festgelegt, auf welche Weise Daten in die Telegrammfelder gepackt und entschlüsselt werden.

Hinweis! Antriebe des Typs ACS 140/ACS 400 unterstützen nur den RTU-Modus. In diesem Handbuch wird nur der RTU-Modus erläutert.

RTU-Modus

Wenn Controller zur Datenübertragung in einem Modbus-Netzwerk auf den RTU-Modus (Remote Terminal Unit) eingestellt sind, enthält jedes 8-Bit-Byte eines Telegramms zwei hexadezimale 4-Bit-Zeichen. Der Hauptvorteil dieses Modus liegt in der höheren Zeichendichte, die im Vergleich zum ASCII-Modus einen höheren Datendurchsatz bei gleicher Baudrate ermöglicht. Jedes Telegramm muß als kontinuierlicher Datenstrom übertragen werden.

Im RTU-Modus ist das Format für jedes Byte:

Codiersystem:	8-Bit binär, hexadezimal 0-9, A-F Zwei hexadezimale Zeichen in jedem 8-Bit-Feld des Telegramms
Bits pro Byte:	1 Start-Bit 8 Datenbits, letztes bedeutendes Bit wird zuerst übertragen 1 Bit für gerade/ungerade Parität; kein Bit ohne Parität 1 Stop-Bit bei Parität; 2 Bits ohne Parität
Fehlerprüffeld:	Zyklische Fehlerprüfung (CRC)

Modbus-Telegrammblöcke

Bei jeder der beiden Übertragungsarten (ASCII oder RTU) wird ein Modbus-Telegramm vom sendenden Gerät in einen Block gepackt, der einen bekannten Anfangs- und Endpunkt besitzt. Dadurch ist es dem empfangenden Gerät möglich, am Anfang des Telegramms zu beginnen, den Adressenabschnitt zu lesen, festzustellen, welches Gerät adressiert ist (oder alle Geräte, im Fall eines allgemeinen Telegramms) und festzustellen, wann das Telegramm beendet ist. Unvollständige Telegramme werden ermittelt und als Konsequenz Fehler gesetzt.

RTU-Telegrammblöcke

Im RTU-Modus beginnt ein Telegramm mit einem stillen Intervall von mindestens 3,5 Zeichen pro Zeiteinheit. Dies entspricht einem Vielfachen der Baudrate, mit der im Netzwerk die Datenübertragung stattfindet (in der untenstehenden Abbildung als T1-T2-T3-T4 angegeben). Das erste übertragene Feld ist die Geräteadresse.

Die für alle Felder zulässigen Zeichen sind Hexadezimale 0-9, A-F. Im Netzwerk eingebundenen Geräte überwachen kontinuierliche den Netzwerk-Bus, auch während der ‘stillen’ Intervalle. Wenn das erste Feld (das Adressfeld) empfangen wurde, wird es von jedem einzelnen Gerät decodiert um zu ermitteln, welches Gerät adressiert wurde.

Nach dem letzten übertragenen Intervall markiert ein identisches Intervall von mindestens 3,5 Zeichen pro Zeiteinheit das Ende des Telegramms. Nach diesem Intervall kann ein neues Telegramm beginnen.

Der gesamte Telegrammblock muß als kontinuierlicher Datenstrom übertragen werden. Falls ein stilles Intervall von mehr als 1,5 Zeichen pro Zeiteinheit vor dem Abschluß des Blocks auftritt, löscht das empfangende Gerät die Daten und nimmt an, daß es sich beim nächsten Byte um das Adressfeld eines neuen Telegramms handelt.

Beginnt ein neues Telegramm früher als 3,5 Zeichen pro Zeiteinheit nach einem vorangegangenen Telegramm, interpretiert es das empfangende Gerät als Fortsetzung des vorangegangenen Telegramms. Dadurch wird ein Fehler gesetzt, da der Wert im endgültigen Fehlerprüffeld (zyklische Fehlerprüfung) nicht für das zusammengesetzte Telegramm gültig ist. Ein typischer Telegrammblock ist unten abgebildet.

START	ADRESSE	FUNKTION	DATEN	ZYKL. PRÜFUNG	ENDE
T1–T2–T3–T4	8 BITS	8 BITS	n × 8 BITS	16 BITS	T1–T2–T3–T4

Abbildung 2 Telegrammblock

Verwendung des Adressfelds

Das Adressfeld eines Telegrammblocks enthält acht Bits (RTU). Gültige Adressen von Slave-Geräten liegen im Bereich von 0 bis 247 (Dezimal). Die einzelnen Slave-Geräte entsprechen zugewiesenen Adressen im Bereich von 1 bis 247. Ein Master adressiert ein Slave-Gerät, indem er die Slave-Adresse in das Adressfeld des Telegramms einträgt. Wenn das Slave-Gerät seine Anwort sendet, trägt es seine eigene Adresse in das Adressfeld der Antwort ein, um den Master zu informieren, welches der Slave-Geräte antwortet.

Adresse 0 wird für die Übertragung allgemeiner Telegramme verwendet, die von allen Slaves empfangen werden. Wenn das Modbus-Protokoll in Netzwerken auf höherer Ebene verwendet wird, sind allgemeine Telegramme unter Umständen nicht zulässig oder werden durch andere Verfahren ersetzt. Modbus-Plus nutzt zum Beispiel eine Mehrbenutzer-Datenbank, die bei jeder Tokenweitergabe aktualisiert werden kann.

Verwendung des Funktionsfelds

Das Feld für den Funktionscode eines Telegrammblocks enthält acht Bits (RTU). Gültige Codes liegen im Bereich 1 bis 255 (Dezimal). Einige dieser Codes gelten für alle Modicon-Controller, während andere Codes nur für bestimmte Modelle relevant sind und andere für zukünftige Anwendungen. Antriebe des Typs ACS140/ACS400 unterstützen die Codes 3, 6, und 16 (0x03, 0x06, und 0x10 Hex).

Wenn ein Telegramm vom Master zu einem Slave-Gerät übertragen wird, teilt das Funktionscodefeld dem Slave mit, welche Aktion durchzuführen ist. Ein Beispiel hierfür ist das Lesen einer Gruppe von Ausgängen.

Wenn der Slave dem Master antwortet, nutzt er das Funktionscodefeld, um entweder eine normale (fehlerfreie) Antwort anzuzeigen oder um anzuzeigen, daß ein Fehler aufgetreten ist (Ausnahmeantwort). Im Fall einer normalen Antwort wiederholt der Slave den ursprünglichen Funktionscode. Im Fall einer Ausnahmeantwort sendet der Slave einen Code, der dem ursprünglichen Funktionscode entspricht, dessen wichtigstes Bit allerdings auf eine logische 1 gesetzt wurde.

Im folgenden Beispiel hat die vom Master zum Slave gesendete Abfrage zum Lesen einer Gruppe von Halteregistern den folgenden Funktionscode:

0000 0011 (Hexadezimal 03)

Wenn das Slave-Gerät die angeforderte Aktion fehlerfrei durchführt, sendet es den gleichen Code in seiner Antwort zurück. Ereignet sich ein Fehler, lautet die Antwort:

1000 0011 (Hexadezimal 83)

Neben der Modifizierung des Funktionscodes zur Erzeugung einer Ausnahmeantwort stellt der Slave einen individuellen Code in das Datenfeld des Antworttelegramms. Dadurch wird der Master über die Art des Fehlers oder den Grund der Ausnahme informiert.

Das Anwendungsprogramm des Master-Geräts ist für die Bearbeitung der Ausnahmeantworten zuständig. Typische Prozesse sind zum Beispiel erneute Versuche der Telegrammübertragung, Diagnose des Slave-Geräts und die Unterrichtung des Bedieners.

Inhalt des Datenfelds

Das Datenfeld setzt sich aus Sätzen von je zwei hexadezimalen Zeichen im Bereich von 00 bis FF (hexadezimal) zusammen. Diese bestehen, je nach seriellem Übertragungsmodus des Netzwerks, aus einem RTU-Zeichen.

Das Datenfeld des von einem Master zu Slave-Geräten gesendeten Telegramms enthält zusätzliche Informationen, die der Slave verwenden muß, um die vom Funktionscode festgelegte Aktion durchführen zu können. Dazu gehören z.B. Einzel- und Registeradressen, die Anzahl der zu bearbeitenden Punkte oder die Zählung der Istwert-Datenbytes im Feld.

Wenn der Master einen Slave beispielsweise auffordert, eine Gruppe von Halteregistern auszulesen (Funktionscode 03), spezifiziert das Datenfeld das Anfangsregister und wieviel Register auszulesen sind. Wenn der Master in eine Gruppe von Registern im Slave schreibt (Funktionscode 10, hexadezimal), spezifiziert das Datenfeld das Anfangsregister, in wieviele Register geschrieben wird, die Zählung der dem Datenfeld folgenden Datenbytes und die Daten, die in die Register geschrieben werden.

Sofern kein Fehler auftritt, enthält das Datenfeld einer vom Slave zum Master gesendeten Antwort die abgefragten Daten. Tritt ein Fehler auf, so enthält das Feld einen Ausnahmecode, mit dessen Hilfe die Master-Anwendung den nächsten durchzuführenden Schritt festlegen kann.

Bei bestimmten Telegrammen kann kein Datenfeld (Länge Null) vorhanden sein. Wenn zum Beispiel ein Master-Gerät ein Slave-Gerät auffordert, Daten des Kommunikations-Fehlerspeichers zu übertragen (Funktionscode 0B, hexadezimal), dann braucht das Slave-Gerät keine zusätzlichen Informationen. Geräte des Typs ACS140/400 unterstützen den Funktionscode 0B (hexadezimal) nicht. Nur der Funktionscode definiert die Aktion.

Inhalt des Fehlerprüffelds

In Standard-Modbus-Netzwerken werden zwei Fehlerprüfverfahren verwendet. Der Inhalt des Fehlerprüffelds hängt vom verwendeten Verfahren ab.

ASCII

Wenn für die Erstellung von Zeichenblöcken das ASCII-Verfahren verwendet wird, enthält das Fehlerprüffeld zwei ASCII-Zeichen. Die Fehlerprüfzeichen sind das Ergebnis einer Längs-Redundandzprüfung (Longitudinal Redundancy Check = LRC), der der Telegramminhalt, mit Ausnahme des ersten “Doppelpunkts“ und der letzten CRLF-Zeichen, unterzogen wird.

Die LRC-Zeichen sind dem Telegramm im letzten Feld vor den CRLF-Zeichen angehängt.

RTU

Wenn der RTU-Modus zur Erstellung von Zeichenblöcken verwendet wird, enthält das Fehlerprüffeld einen 16-Bit-Wert, der in Form von zwei 8-Bit-Bytes implementiert wird. Die Fehlerprüfwert ist das Ergebnis einer zyklischen Fehlerprüfung, der der Telegramminhalt unterzogen wird.

Die LRC-Zeichen sind dem Telegramm im letzten Feld angehängt. Wenn dieser Schritt abgeschlossen ist, wird das niederwertige Byte im Feld zuerst angehängt und anschließend das höherwertige Byte. Das höherwertige CRC-Byte ist das letzte im Rahmen des Telegramms übertragene Byte.

Weitere Informationen über die Fehlerprüfung können weiter hinten in diesem Anhang nachgelesen werden.

Serielle Übertragung von Zeichen

Wenn in seriellen Standard-Modbus-Netzwerken Telegramme übertragen werden, wird jedes Zeichen bzw. Byte in der folgenden Reihenfolge übertragen (von links nach rechts):

Am wenigsten wichtiges Bit … Wichtigstes Bit

Bei der RTU-Zeichenblockbildung lautet die Bitfolge:

Mit Paritätsprüfung

Start	1	2	3	4	5	6	7	8	Par	Stop

Ohne Paritätsprüfung

Start	1	2	3	4	5	6	7	8	Stop	Stop

Fehlerprüfverfahren

In seriellen Standard-Modbus-Netzwerken kommen zwei Fehlerprüfverfahren zur Anwendung. Die Paritätsprüfung (gerade oder ungerade) kann optional bei jedem Zeichen durchgeführt werden. Die Blockprüfung (CRC) wird auf das gesamte Telegramm angewandt. Sowohl die Zeichenprüfung als auch die Telegrammblockprüfung werden im Master-Gerät auf den Telegramminhalt angewandt, bevor dieser gesendet wird. Das Slave-Gerät prüft jedes Zeichen und den gesamten Telegrammblock während des Empfangs.

Der Master wird vom Anwender so konfiguriert, daß zunächst ein voreingestelltes Zeitintervall ablaufen muß, bevor die Übertragung unterbrochen wird. Dieses Intervall ist so zu wählen, daß jeder Slave ausreichend Zeit hat, um normal zu antworten. Wenn der Slave einen Übertragungsfehler feststellt, wird das Telegramm nicht verarbeitet. Der Slave sendet keine Antwort zum Master. Daher wird das Zeitintervall überschritten und das Programm des Master-Geräts übernimmt die Fehlerbearbeitung. Beachten Sie, daß ein an einen nicht vorhandenen Slave adressiertes Telegramm ebenfalls zu einer Zeitüberschreitung führt. Für Antriebe des Typs ACS140/ACS400 sollte das Zeitintervall auf 100ms oder mehr eingestellt werden.

Paritätsprüfung

Anwender können die Controller auf die Prüfung hinsichtlich gerader bzw. ungerader Parität oder auf keine Paritätsprüfung einstellen. Dadurch wird festgelegt, wie das Paritätsbit in jedem Zeichen gesetzt wird.

Wenn eine Prüfung auf gerade oder ungerade Parität spezifiziert wurde, wird die Anzahl von 1-Bits im Datenfeld jedes Zeichens (acht bei RTU) gezählt. Das Paritätsbit wird dann auf 0 oder 1 gesetzt, so daß eine gerade oder ungerade Gesamtsumme von 1-Bits ermittelt wird.

Hier sind beispielsweise diese acht Datenbits in einem RTU-Zeichenblock enthalten:

1100 0101

Die Gesamtsumme von 1-Bits im Block ist vier. Wird auf gleiche Parität geprüft, lautet die Parität des Blocks 0, so daß die Gesamtzahl der 1-Bits noch immer eine gerade Zahl ist (vier). Wird ungerade Parität genutzt, wird das Paritätsbit 1und damit eine ungerade Anzahl (5) erzeugt.

Wenn das Telegramm übertragen wurde, wird das Paritätsbit errechnet und auf jeden Zeichenblock angewandt. Das empfangende Gerät zählt die Anzahl der 1-Bits und setzt einen Fehler, falls die Anzahl nicht dem für dieses Gerät konfigurierten Wert entspricht. Alle Geräte in einem Modbus-Netzwerk müssen auf die Verwendung des selben Paritätsprüfverfahrens eingestellt sein.

Beachten Sie, daß durch die Paritätsprüfung nur dann ein Fehler ermittelt werden kann, wenn während der Übertragung eine ungerade Anzahl von Bits aus einem Zeichenblock entfernt oder hinzugefügt werden. Wenn zum Beispiel auf ungerade Parität geprüft wird und aus einem Zeichen mit drei 1-Bits zwei 1-Bits entfernt werden, ist das Ergebnis noch immer eine ungerade Zahl von 1-Bits.

Wenn keine Paritätsprüfung spezifiziert wurde, wird kein Paritätsbit übertragen und es erfolgt keine Paritätsprüfung. Um den Zeichenblock zu vervollständigen, wird ein zusätzliches Stop-Bit übertragen.

Zyklische Fehlerprüfung (CRC)

Im RTU-Modus enthalten Telegramme ein Fehlerprüffeld, das auf der zyklischen Fehlerprüfung (CRC) basiert. Das CRC-Feld prüft den Inhalt des gesamten Telegramms. Die Prüfung wird in jedem Fall durchgeführt, unabhängig vom Paritätsprüfverfahren für die einzelnen Zeichen des Telegramms.

Das CRC-Feld besteht aus zwei Bytes, die einen binären 16-Bit-Wert enthalten. Der CRC-Ergebnis wird vom sendenden Gerät errechnet, das den CRC-Wert an das Telegramm anhängt. Das empfangende Gerät führt während des Erhalts des Telegramms eine Neuberechnung der CRC durch und vergleicht den errechneten Wert mit dem tatsächlichen Wert im CRC-Feld. Sind die beiden Werte nicht identisch, wird ein Fehler gesetzt.

Die zyklische Fehlerprüfung wird eingeleitet, indem zunächst ein 16-Bit-Register mit allen 1ern geladen wird. Dann werden aufeinanderfolgende 8-Bit-Bytes des Telegramms auf den aktuellen Inhalt des Registers angewandt. Nur die acht Datenbits in jedem Zeichen werden zur Durchführung der zyklischen Fehlerprüfung herangezogen. Start- und Stop-Bits, und - falls verwendet - das Paritätsbit werden bei der zyklischen Fehlerprüfung nicht berücksichtigt.

Während der Durchführung der zyklischen Fehlerprüfung wird jedes 8-Bit-Zeichen mit dem Registerinhalt verglichen und eine exklusive ODER-Verknüpfung hergestellt. Das Ergebnis wird in Richtung des am wenigsten wichtigen Bit verschoben, wobei an die Stelle des wichtigsten Bit eine Null gesetzt wird. Das am wenigsten wichtige Bit wird entfernt und untersucht. War des am wenigsten wichtige Bit eine 1, dann wird mit dem Inhalt des Registers eine exkusive ODER-Verknüpfung mit einem voreingestellten, feststehenden Wert durchgeführt. War das am wenigsten wichtige Bit eine 0, so findet keine exklusive ODER-Verknüpfung statt.

Dieser Vorgang wiederholt sich solange, bis acht Stellenverschiebungen durchgeführt wurden. Nach der letzten Verschiebung (acht), wird das nächste 8-Bit-Byte mit dem Registerinhalt verglichen und eine exklusive ODER-Verknüpfung hergestellt. Dieser Vorgang wiederholt sich wie oben beschrieben bei acht weiteren Verschiebungen. Der endgültige Registerinhalt ist nach Berechnung aller Telegrammbytes das Ergebnis der zyklischen Fehlerprüfung.

In einem Leiternetzwerk führt die Kontrollsummenfunktion (CKSM) die zyklische Fehlerprüfung des Telegramminhalts durch. Weiter hinten in diesem Anhang findet sich ein detailliertes Beispiel für die zyklische Fehlerprüfung bei Anwendungen mit Host-Controller.

Modbus-Funktionsformate

In diesem Kapitel wird der Dateninhalt von Modbus-Telegrammen, die von Antrieben des Typs ACS 140/400 unterstützt werden, detailliert erläutert.

Wiedergabe numerischer Werte

Sofern nicht anders angegeben, werden numerische Werte (wie zum Beispiel Adressen, Codes oder Daten) im *Text dieses Abschnitts als Dezimalwerte* wiedergegeben. In den *Telegrammfeldern der Abbildungen werden sie als hexadezimale Werte* wiedergegeben.

Datenadressen in Modbus-Telegrammen

Alle Datenadressen in Modbus-Telegrammen werden als Null referenziert. Das erste Auftreten eines Datenelements wird als Element Nr. 0 adressiert. Beispiel:

- Die als ‘Spule 1’ in einem programmierbaren Controller eingetragene Spule wird im Datenadressfeld eines Modbus-Telegramms als 0000 adressiert.
- Spule 127 (dezimal) wird als Spule 007E hexadezimal (126 dezimal) adressiert.
- Halteregister 40001 wird im Datenadressfeld des Telegramms als 0000 adressiert. Im Funktionscodefeld ist bereits eine ‘Halteregister’ -Operation spezifiziert. Daher ist die Referenz ‘4XXXX’ implizit.
- Halteregister 40108 wird als Register 006B hexadezimal (107 dezimal) adressiert.

Feldinhalt von Modbus-Telegrammen

Abbildung 3 Master-Abfrage mit RTU-Datenblock zeigt ein Beispiel für eine Modbus-Abfrage. *Abbildung 4 Slave-Antwort mit RTU-Datenblock* ist ein Beispiel für eine normale Antwort. Beide Beispiele zeigen den Inhalt in hexadezimaler Form und erläutern, wie ein Telegramm im RTU-Modus zu einem Datenblock zusammengefaßt wird.

Der Master schickt die Abfrage zum Lesen von Halteregistern an die Slave-Geräteadresse 40110. Inhalt des Telegramms ist Abfrage von drei Halteregistern 40108 bis 40110. Beachten Sie, daß das Telegramm die Adresse des Anfangsregisters als 0107 (006B hexadezimal) spezifiziert.

In der Antwort vom Slave wird der Funktionscode wiederholt; dadurch wird angezeigt, daß es sich um eine normale Antwort handelt. Im Feld ‘Bytezählung’ wird spezifiziert, wieviele *8-Bit-Datenelemente* zurückgesandt werden. Es zeigt die gezählten 8-Bit -Bytes, die den Daten für die RTU folgen.

Es wird zum Beispiel der Wert 63 (hexadezimal) als einzelnes 8-Bit-Byte im RTU-Modus (01100011) übertragen. Im Feld 'Bytezählung' werden diese Daten als einzelnes 8-Bit-Element gezählt, unabhängig vom Verfahren zur Zeichenblockerstellung (ASCII oder RTU).

Verwendung des Zeichenzählungsfelds: Wenn Sie Antworten in Zwischenspeichern ablegen, müssen Sie einen Byte-Zählungswert verwenden, der der Zählung der 8-Bit-Bytes in Ihren Telegrammdaten entspricht. Der Wert ist unabhängig vom übrigen Inhalt des Felds, einschließlich des Byte-Zählungsfelds. *Abbildung 4 Slave-Antwort mit RTU-Datenblock* zeigt, wie das Zeichenzählungsfeld bei einer typischen Antwort eingesetzt wird.

ABFRAGE		
Feldname	Beispiel (Hexadezimal)	RTU 8-Bit-Feld
Kopfzeile		Nicht vorhanden
Slave-Adresse	06	0000 0110
Funktion	03	0000 0011
Anfangsadreses Hoch	00	0000 0000
Anfangsadreses Nierig	6B	0110 1011
Anzahl der Register Hoch	00	0000 0000
Anzahl der Register Niedrig	03	0000 0011
Fehlerprüfung		CRC (16-Bits)
Nachsatz		Nicht vorhanden
	Gesamtzahl der Bytes:	8

Abbildung 3 Master-Abfrage mit RTU-Datenblock

ANTWORT		
Feldname	Beispiel (Hexadezimal)	RTU 8-Bit-Feld
Kopfzeile		Nicht vorhanden
Slave-Adresse	06	0000 0110
Funktion	03	0000 0011
Byte-Zählung	06	0000 0110
Daten Hoch	02	0000 0010
Daten Niedrig	2B	0010 1011
Daten Hoch	00	0000 0000
Daten Niedrig	00	0000 0000
Daten Hoch	00	0000 0000
Daten Niedrig	00	0000 0000
Fehlerprüfung		CRC (16 bits)
Nachsatz		Nicht vorhanden
	Gesamtzahl der Bytes:	11

Abbildung 4 Slave-Antwort mit RTU-Datenblock

Functionscodes

Antriebe des Typs ACS 140/ACS 400 unterstützen drei Modbus-Funktionscodes. Dadurch ist es dem Master möglich, ganzzahlige 16-Bit-Werte für den Antrieb ein- und auszulesen.

03 Halteregister lesen

Liest den binären Inhalt der Halteregister (4X Referenzen) im Slave. Allgemeine Telegramme werden nicht unterstützt.

Abfrage

Die Abfrage spezifiziert das Anfangsregister und die Anzahl der zu lesenden Register. Register werden ab Null adressiert: Register 1–16 werden als 0–15 adressiert.

Hier ein Beispiel die Aufforderung zum Lesen der Register 40108–40110 durch Slave-Gerät 17:

Feldname	
Kopfzeile	Beispiel (Hexadezimal)
Slave-Adresse	11
Funktion	03
Anfangsadreses Hoch	00
Anfangsadreses Niedrig	6B
Anzahl der Register Hoch	00
Anzahl der Register Niedrig	03
Fehlerprüfung	CRC (16-Bit)

Abbildung 5 Halteregister lesen- Abfrage

Antwort

Die Registerdaten in der Antwort werden jeweils als zwei Byte pro Register gepackt und der binäre Inhalt in jedem Byte genau ausgeglichen.

Bei Controllern des Typs 984-X8X (984-685) werden im Slave Daten in Einheiten von je 125 Registern abgefragt (984–685, etc.), bei allen anderen Controllern erfolgt die Abfrage in Einheiten von je 32 Registern. Die Antwort wird übertragen, sobald die Daten vollständig assembliert sind.

Hier ein Beispiel für die Antwort auf die vorangegangene Abfrage:

Feldname	
Kopfzeile	Beispiel (Hexadezimal)
Slave-Adresse	11
Funktion	03
Byte-Zählung	06
Daten Hoch (Register 40108)	02
Daten Niedrig (Register 40108)	2B
Daten Hoch (Register 40109)	00
Daten Niedrig (Register 40109)	00
Daten Hoch (Register 40110)	00
Daten Niedrig (Register 40110)	64
Fehlerprüfung CRC	CRC (16-Bit)

Abbildung 6 Halteregister lesen - Antwort

Der Inhalt von Register 40108 wird in Form von zwei Byte-Werten, d.h. 02 2B hexadezimal bzw. 555 dezimal wiedergegeben. Der Inhalt der Register 40109–40110 ist 00 00 und 00 64 hexadezimal bzw. 0 und 100 dezimal.

06 Einzelregister voreinstellen

Führt die Voreinstellung eines Wertes in einem einzelnen Halteregister durch (4X Referenz). Bei einem allgemeinen Telegramm führt die Funktion die Voreinstellung der gleichen Registerreferenz in allen angeschlossenen Slaves durch.

Abfrage

Die Abfrage spezifiziert die Registerreferenz, die voreingestellt werden soll. Register werden ab Null adressiert: Register 1 wird als 0 adressiert.

Der angeforderte Wert für die Voreinstellung wird im Datenfeld der Abfrage spezifiziert. Antriebe des Typs ACS 140/ACS 400 verwenden 16-Bit-Werte.

Hier ein Beispiel für die Aufforderung, die Voreinstellung der Register 40002 bis 00 03 hexadezimal in Slave-Gerät 17 durchzuführen:

ABFRAGE	
Feldname	Beispiel (Hexadeziaml)
Slave-Adresse	11
Funktion	06
Registeradresse Hoch	00
Registeradresse Niedrig	01
Voreingestellte Daten Hoch	00
Voreingestellte Niedrig	03
Fehlerprüfung CRC	CRC (16-Bis)

Abbildung 7 Einzelregister voreinstellen - Abfrage

Antwort

Nachdem der Inhalt der Register voreingestellt wurde, wird als normale Antwort ein Echo der Abfrage übertragen.

Hier ein Beispiel für eine Antwort auf die Abfrage:

ANTWORT	
Feldname	Beispiel (Hexadezimal)
Slave-Adresse	11
Funktion	06
Registeradresse Hoch	00
Registeradresse Niedrig	01
Voreingestellte Daten Hoch	00
Voreingestellte Niedrig	03
Fehlerprüfung CRC	CRC (16-Bits)

Abbildung 8 Einzelregister voreinstellen - Antwort

16 (10 Hexadezimal) Mehrere Register voreinstellen

Führt die Voreinstellung eines Wertes in mehreren, aufeinanderfolgenden Halteregistern durch (4X Referenz). Bei einem allgemeinen Telegramm führt die Funktion die Voreinstellung der gleichen Registerreferenz in allen angeschlossenen Slaves durch.

Bei Antrieben des Typs ACS 140/ACS 400 ist es möglich, mit Hilfe der Funktion zur Voreinstellung mehrerer Register in ein oder mehrere Register gleichzeitig zu schreiben. Es kann nur in Register innerhalb einer Gruppe geschrieben werden. Falls das Schreiben in eines der Register fehlschlägt, versucht der ACS 140/ACS 400 in andere Register zu schreiben; in diesem Fall enthält die Antwort ein entsprechendes Ausnahmetelegramm.

Abfrage

Die Abfrage spezifiziert die Registerreferenz, die voreingestellt werden soll. Register werden ab Null adressiert: Register 1 wird als 0 adressiert.

Die angeforderten Werte für die Voreinstellung werden im Datenfeld der Abfrage spezifiziert. Antriebe des Typs ACS 140/ACS 400 verwenden 16-Bit-Werte. Daten werden in Form von zwei Byte pro Register gepackt.

Hier ein Beispiel für die Aufforderung, die Voreinstellung der Register von 40002 bis 00 0A in Slave-Gerät 17 durchzuführen:

ABFRAGE	
Feldname	Beispiel (Hexadezimal)
Slave-Adresse	11
Funktion	10
Anfangsadresse Hoch	00
Anfangsadresse Niedrig	01
Anzahl der Register Hoch	00
Anzahl der Register Niedrig	01
Byte Count	02
Data Hi	00
Data Lo	0A
Error Check CRC	CRC (16-Bits)

Abbildung 9 Mehrere Register voreinstellen - Abfrage

Antwort

Die normale Antwort enthält die Slave-Adresse, den Funktionscode, die Anfangsadresse und die Anzahl der Register, die voreingestellt werden sollen.

Hier ein Beispiel für die Antwort auf die Abfrage.

ANTWORT	
Feldname	Beispiel (Hexadezimal)
Slave-Adresse	11
Funktion	10
Anfangsadresse Hoch	00
Anfangsadresse Niedrig	01
Anzahl der Register Hoch	00
Anzahl der Register Niedrig	01
Fehlerprüfung CRC	CRC (16-Bit)

Abbildung 10 Mehrere Register voreinstellen - Antwort

Ausnahmeantworten

Wenn ein Master-Gerät eine Abfrage an ein Slave-Gerät sendet, erwartet das Master-Gerät eine normale Antwort; eine Ausnahme bilden allgemeine Telegramme. Nach der Übertragung einer Abfrage durch den Master kann eines der vier Ereignisse eintreten:

1 Wenn das Slave-Gerät die Abfrage ohne Datenübertragungsfehler erhält und die Abfrage normal bearbeiten kann, wird eine normale Antwort zurückgesandt.

2 Wenn das Slave-Gerät aufgrund eines Datenübertragungsfehlers die Abfrage nicht erhält, wird keine Antwort zurückgesandt. Das Programm des Master-Geräts stellt für die Abfrage eine Zeitüberschreitung fest.

3 Wenn der Slave eine Abfrage empfängt, jedoch einen Datenübertragungsfehler ermittelt (Parität, LRC oder CRC), wird keine Antwort zurückgesandt. Das Programm des Master-Geräts stellt für die Abfrage eine Zeitüberschreitung fest.

4 Wenn der Slave die Abfrage ohne Datenübertragungsfehler erhält, die Abfrage jedoch nicht bearbeiten kann (zum Beispiel wenn die Aufforderung ergeht, den Betriebszustand einer nicht vorhandenen Spule bzw. ein nicht vorhandenes Register auszulesen), wird eine Ausnahmeantwort zurückgesandt, mit der das Master-Gerät über die Art des Fehlers informiert wird. Die Ausnahmeantwort besitzt zwei Felder, die sie von einer normalen Antwort unterscheiden:

Funktionscodefeld Bei einer normalen Antwort sendet der Slave eine Kopie des in der ursprünglichen Abfrage enthaltenen Funktionscodes im entsprechenden Feld der Antwort zurück. Alle Funktionscodes besitzen ein wichtigstes Bit 0 (alle Werte unter 80 hexadezimal). Bei einer Ausnahmeantwort setzt der Slave das wichtigste Bit des Funktionscodes auf 1. Dadurch wird der Wert des Funktionscode bei einer Ausnahmeantwort um genau 80 Hexadezimale höher, als er in einer normalen Antwort sein würde. Ist das wichtigste Bit des Funktionscode gesetzt, kann das Anwendungsprogramm des Master-Geräts die Ausnahmeantwort identifizieren und das Datenfeld auf den Ausnahmecode untersuchen,

Datenfeld: Bei einer normalen Antwort sendet der Slave Daten oder Statistiken im Datenfeld (alle Informationen, die abgefragt wurden). Bei einer Ausnahmeantwort sendet der Slave einen Ausnahmecode im Datenfeld. Dadurch wird der Betriebszustand des Slave definiert, der die Ausnahme verursacht hat. *Abbildung C-11 'Abfrage durch Master und Ausnahmeantwort von Slave'* zeigt ein Beispiel für eine Abfrage durch den Master und die Ausnahmeantwort vom Slave.

Beispiele für die Felder in Hexadezimalen.

ABFRAGE		
Byte	Inhalt	Beispiel
1	Slave-Adresse	0A
2	Funktion	01
3	Anfangsadresse Hoch	04
4	Anfangsadresse Niedrig	A1
5	Anzahl der Spulen Hoch	00
6	Anzahl der Spulen Niedrig	01
7	LRC	4F
AUSNAHMEANTWORTEN		
1	Slave-Adresse	0A
2	Funktion	81
3	Ausnahmecode	02
4	LRC	73

Abbildung 11 Abfrage durch Master und Ausnahmeantwort von Slave

In diesem Beispiel adressiert der Master eine Abfrage an Slave-Gerät 10 (0A hexadezimal). Der Funktionscode (01) steht für das Auslesen des Spulen-Betriebszustands. Es wird der Betriebszustand der Spule mit der Adresse 1245 (04A1 hexadezimal) angefordert. Beachten Sie, daß entsprechend dem Eintrag im Feld für die Spulenanzahl (0001) nur der Zustand einer einzelnen Spule auszulesen ist.

Falls die Spulenadresse im Slave-Gerät nicht vorliegt, sendet der Slave eine Ausnahmeantwort mit dem gezeigten Ausnahmecode (02). Dadurch wird eine unzulässige Datenadresse für den Slave spezifiziert. Handelt es sich beim Slave zum Beispiel um einen Typ 984-385 mit 512 Spulen, würde dieser Code zurückgesandt.

Die Modicon-Ausnahmecodes enthält *Tabelle 1 Standard-Ausnahmecodes.*

Tabelle 1 Standard-Ausnahmecodes.

Code	Name	Bedeutung
01	UNZULÄSSIGE FUNKTION	Der zusammen mit der Abfrage empfangene Funktionscode bezieht sich auf eine für den Slave unzulässige Aktion. Wenn eine Programmabfrage durchgeführt wird, zeigt dieser Code an, daß keine Programmfunktion vorausgegangen ist.
02	UNZULÄSSIGE DATENADRESSE	Die zusammen mit der Abfrage empfangene Datenadresse ist für den Slave nicht zulässig.
03	UNZULÄSSIGER DATENWERT	Ein im Abfragedatenfeld enthaltener Wert ist für den Slave nicht zulässig.
04	SLAVE-GERÄTEFEHLER	Während der Slave versucht hat, die angeforderte Aktion durchzuführen, ist ein nicht behebbarer Fehler aufgetreten.
05	RÜCKMELDUNG	Der Slave hat die Abfrage akzeptiert und bearbeitet sie, allerdings ist hierzu viel Zeit erforderlich. Diese Antwort wird gesandt, um einen Zeitfehler im Master zu verhindern. Der Master kann anschließend eine Programmabfrage veranlassen, um festzustellen, ob die Bearbeitung abgeschlossen ist.
06	SLAVE-GERÄT BELEGT	Der Slave ist mit der Bearbeitung eines zeitaufwendigen Programmbefehls beschäftigt. Der Master versucht, die Abfrage zu einem späteren Zeitpunkt erneut zu übertragen.
07	NEGATIVE RÜCKMELDUNG	Der Slave kann die in der Abfrage enthaltene Programmfunktion nicht ausführen. Dieser Code steht für eine fehlgeschlagene Programmierabfrage unter Verwendung von Funktionscode 13 oder 14 dezimal. Der Master fragt Diagnose- oder Fehlerdaten vom Slave ab.
08	PARITÄTSFEHLER IM SPEICHER	Der Slave hat versucht, den erweiterten Speicher auszulesen und dabei einen Paritätsfehler im Speicher festgestellt. Der Master kann die Abfrage erneut versuchen, allerdings muß das Slave-Gerät möglicherweise instandgesetzt werden.

Zyklische Fehlerprüfung (CRC)

Das CRC-Feld besteht aus zwei Bytes, die einen binären 16-Bit-Wert enthalten. Der CRC-Ergebnis wird vom sendenden Gerät errechnet, das den CRC-Wert an das Telegramm anhängt. Das empfangende Gerät führt während des Erhalts des Telegramms eine Neuberechnung der CRC durch und vergleicht den errechneten Wert mit dem tatsächlichen Wert im CRC-Feld. Sind die beiden Werte nicht identisch, wird ein Fehler gesetzt.

Die zyklische Fehlerprüfung wird eingeleitet, indem zunächst ein 16-Bit-Register in alle 1er geladen wird. Dann werden aufeinanderfolgende 8-Bit-Bytes des Telegramms auf den aktuellen Inhalt des Registers angewandt. Nur die acht Datenbits in jedem Zeichen werden zur Durchführung der zyklischen Fehlerprüfung herangezogen. Start- und Stop-Bits, und - falls verwendet - das Paritätsbit werden bei der zyklischen Fehlerprüfung nicht berücksichtigt.

Während der Durchführung der zyklischen Fehlerprüfung wird jedes 8-Bit-Zeichen mit dem Registerinhalt verglichen und eine exklusive ODER-Verknüpfung hergestellt. Das Ergebnis wird in Richtung des am wenigsten wichtigen Bit verschoben, wobei an die Stelle des wichtigsten Bit eine Null gesetzt wird. Das am wenigsten wichtige Bit wird entfernt und untersucht. War des am wenigsten wichtige Bit eine 1, dann wird mit dem Inhalt des Registers eine exklusive ODER-Verknüpfung mit einem voreingestellten, feststehenden Wert durchgeführt. War das am wenigsten wichtige Bit eine 0, so findet keine exklusive ODER-Verknüpfung statt.

Dieser Vorgang wiederholt sich solange, bis acht Stellenverschiebungen durchgeführt wurden. Nach der letzten Verschiebung (acht), wird das nächste 8-Bit-Byte mit dem Registerinhalt verglichen und eine exklusive ODER-Verknüpfung hergestellt. Dieser Vorgang wiederholt sich wie oben beschrieben bei acht weiteren Verschiebungen. Der endgültige Registerinhalt ist nach Berechnung aller Telegrammbytes das Ergebnis der zyklischen Fehlerprüfung.

Durchführung der zyklischen Fehlerprüfung CRC:

1 Ein 16-Bit-Register mit FFFF hexadezimal (alle 1er) laden. Dies ist das CRC-Register.

2 Das erste 8-Bit-Byte des Telegramms über eine exklusive ODER-Funktion mit dem niederwertigen Byte des 16-Bit-CRC-Registers verknüpfen.

3 Das CRC-Register um ein Bit nach rechts (zum am wenigsten wichtigen Bit) verschieben und das wichtigste Bit durch 0 ersetzen. Das am wenigsten wichtige Bit herausnehmen und untersuchen.

4 (Falls das am wenigsten wichtige Bit 0 war): Schritt 3 wiederholen (weitere Verschiebung).
(Falls das am wenigsten wichtige Bit 1 war): Das CRC-Register über die exklusive ODER-Funktion mit dem Polynomwert A001 hexadezimal (101 0 0000 0000 0001) verknüpfen.

5 Schritte 3 und 4 wiederholen, bis 8 Verschiebungen durchgeführt worden sind. Das Ergebnis ist ein vollständig bearbeitetes 8-Bit-Byte.

6 Schritte 2 bis 5 beim nächsten 8-Byte-Bit des Telegramms wiederholen. Fortfahren, bis alle Bytes bearbeitet worden sind.

7 Der endgültige Inhalt des CRC-Registers ist der CRC-Wert.

CRC in das Telegramm einbetten

Wenn die 16-Bit-CRC (2 8-Bit-Bytes) zusammen mit dem Telegramm übertragen wird, erfolgt zunächst die Übertragung des niederwertigen Byte und anschließend die des höherwertigen Byte. In diesem Beispiel lautet der CRC-Wert 1241 hexadezimal (0001 0010 0100 0001):

Adr.	Funkt.	Daten-zählung	Daten	Daten	Daten	Daten	CRC Niedrig 41	CRC Hoch 12

Abbildung 12 CRC Byte-Folge

Beispiel

Ein Beispiel für die zyklische Fehlerprüfung mit Programmiersprache C ist auf den folgenden Seiten aufgeführt. Alle der möglichen CRC-Werte werden in zwei Felder geladen, die über den Telegrammpuffer einfach als Funktionsinkremente indiziert werden. Ein Feld enthält alle der 256 möglichen CRC-Werte für das höherwertige Byte im 16-Bit-CRC-Feld; das andere enthält alle Werte für das niederwertige Byte. Durch die Indizierung der CRC mit Hilfe dieser Vorgehensweise wird die Fehlerprüfung schneller durchgeführt als dies durch die Berechnung eines neuen CRC-Wertes für jedes neue Zeichen im Puffer möglich ist.

Die Funktion verwendet zwei Parameter:

`unsigned char *puchMsg`
Eine Hinweisadresse des Telegrammpuffers, die binäre Daten für die Durchführung der zyklischen Fehlerprüfung (CRC) enthält

`unsigned short usDataLen`
Die Anzahl der Bytes im Telegrammpuffer.

Die Funktion gibt das CRC-Ergebnis in Kurzform ohne Vorzeichen wieder .

```
/* Table of CRC values for high-order byte */

static unsigned char auchCRCHi [ ] = {

0x00,0xC1,0x81,0x40,0x0l,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xCl,
0x81,

0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x0l,
0xC0,

0x80,0x41,0x0l,0xC0,0x80,0x41,0x00,0xCl,0x81,0x40,0x00,0xC1,0x81,0x40,
0x01,

0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0
,0x41,

0x00,0xCl,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,
0x81,

0x40,0x0l,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,
0xC0,

0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80
,0x01,

0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,
0x40,

0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,
0x81,

0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,
0xC0,

0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,
0x01,

0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,
0x41,

0x00,0xC1,0x81,0x40,0x00,0xCl,0x81,0x40,0x0l,0xC0,0x80,0x41,0x00,0xC1,
0x81,

0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,
0xC0,

0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,
0x01,

0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,
0x41,

0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,
0x81,

0x40

};
```

```
/* Table of CRC values for low-order byte*/

static char auchCRCLo [ ] = {

0x00,0xC0,0xC1,0x01,0xC3,0x03,0x02,0xC2,0xC6,0x06,0x07,0xC7,0x05,0xC5,
0xC4,

0x04,0xCC,0x0C,0x0D,0xCD,0x0F,0xCF,0xCE,0x0E,0x0A,0xCA,0xCB,0x0B,0xC9,
0x09,

0x08,0xC8,0xD8,0x18,0xl9,0xD9,0xlB,0xDB,0xDA,0xlA,0xlE,0xDE,0xDF,0xlF,
0xDD,

0xlD,0xlC,0xDC,0x14,0xD4,0xD5,0x15,0xD7,0x17,0x16,0xD6,0xD2,0x12,0x13,
0xD3,

0xll,0xDl,0xD0,0xl0,0xF0,0x30,0x31,0xF1,0x33,0xF3,0xF2,0x32,0x36,0xF6,
0xF7,

0x37,0xF5,0x35,0x34,0xF4,0x3C,0xFC,0xFD,0x3D,0xFF,0x3F,0x3E,0xFE,0xFA,
0x3A,

0x3B,0xFB,0x39,0xF9,0xF8,0x38,0x28,0xE8,0xE9,0x29,0xEB,0x2B,0x2A,0xEA,
0xEE,

0x2E,0x2F,0xEF,0x2D,0xED,0xEC,0x2C,0xE4,0x24,0x25,0xE5,0x27,0xE7,0xE6,
0x26,

0x22,0xE2,0xE3,0x23,0xE1,0x21,0x20,0xE0,0xA0,0x60,0x61,0xA1,0x63,0xA3,
0xA2,

0x62,0x66,0xA6,0xA7,0x67,0xA5,0x65,0x64,0xA4,0x6C,0xAC,0xAD,0x6D,0xAF,
0x6F,

0x6E,0xAE,0xAA,0x6A,0x6B,0xAB,0x69,0xA9,0xA8,0x68,0x78,0xB8,0xB9,0x79,
0xBB,

0x7B,0x7A,0xBA,0xBE,0x7E,0x7F,0xBF,0x7D,0xBD,0xBC,0x7C,0xB4,0x74,0x75,
0xB5,

0x77,0xB7,0xB6,0x76,0x72,0xB2,0xB3,0x73,0xB1,0x71,0x70,0xB0,0x50,0x90,
0x91,

0x51,0x93,0x53,0x52,0x92,0x96,0x56,0x57,0x97,0x55,0x95,0x94,0x54,0x9C,
0x5C,

0x5D,0x9D,0x5F,0x9F,0x9E,0x5E,0x5A,0x9A,0x9B,0x5B,0x99,0x59,0x58,0x98,
0x88,

0x48,0x49,0x89,0x4B,0x8B,0x8A,0x4A,0x4E,0x8E,0x8F,0x4F,0x8D,0x4D,0x4C,
0x8C,

0x44,0x84,0x85,0x45,0x87,0x47,0x46,0x86,0x82,0x42,0x43,0x83,0x41,0x81,
0x80,

0x40

};
```

```
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg ;        / * message to calculate CRC upon
* /
unsigned short usDataLen;       /* quantity of bytes in message
*/
{
unsigned char uchCRCHi = 0xFF;  /* high byte of CRC initialized
*/
unsigned char uchCRCLo = 0xFF;  /* low byte of CRC initialized*/
unsigned uIndex;                /* will index into CRC lookup table
*/

while (usDataLen--)             /* pass through message buffer*/
{
   uIndex = uchCRCHi ^ *puchMsg++;/* calculate the CRC*/
   uchCRCHi = uchCRCLo ^ auchCRCHi [uIndex] ;
   uchCRCLo = auchCRCLo [uIndex] ;
}
return (uchCRCHi << 8 | uchCRCLo);
```

3AFY 64200534 R0103
DE
Gültig ab: 1.5.1998

ABB Automation Products GmbH
GG Standard Antriebe
Postfach 10 02 61
68002 Mannheim
Telefon: 0800-2667220
Telefax: 0621/381-1622