ADwin-CAN

Hardware - Beschreibung

Registerbelegung

Programmierbeispiele

Version 1.2

August 2000

Inhalt

Installation

Die ***ADwin-CAN***-Karte ist über die Pfostenstecker an der Rückseite mit dem Prozessormodul verbunden. Die Karte kann mit dem Prozessormodul alleine betrieben, oder mit weiteren Karten der ***ADwin***-Serie kombiniert werden. Sollen weiter ***ADwin***-Komponenten verwendet werden, müssen die einzelnen Komponenten mit Hilfe einer Backplane verbunden werden. Die nachfolgende Grafik zeigt eine Kartenkombination in der Draufsicht (***ADwin-CAN*** und ein weiteres ***ADwin-***Modul).

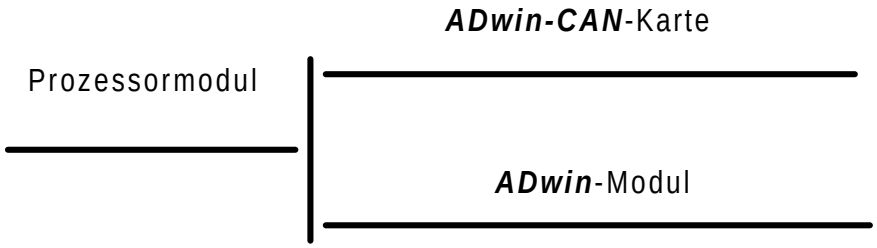

Abbildung 1: Kombination einer *ADwin-CAN*-Karte und einem anderen *ADwin*-Modul

Bei dem hier dargestellten ***ADwin***-Modul kann es sich um ein beliebiges Modul der ***ADwin***-Serie handeln. Wenn es sich um ein Modul mit großer Baulänge handelt, muß die ***ADwin-CAN***-Karte mit einer Expanderkarte verlängert werden. Das komplette System kann um bis zu drei ***ADwin***-Standard-Karten oder um eine ***ADwin-light***-Karte erweitert werden. Die ***ADwin-CAN***-Karte und jede Erweiterungskarte belegen je einen ISA-Slot.

Das komplette System wird in den PC eingesteckt. Die Karte befindet sich nun im I/O-Bereich des PCs. Die I/O-Adresse kann mit Hilfe der weißen DIP-Schalter eingestellt werden. Dabei ist zu beachten, daß die Einstellung der Adresse auf der linken Karte erfolgt (von der PC-Rückseite gesehen). Die Schalter auf den anderen Karten müssen alle auf „off“ stehen. Die Nachfolgende Tabelle zeigt die Zuordnung zwischen den Adressen und den Schalterstellungen.

Schalter Nr.

	2	3	4	5	6	7	8	9
Basisadresse:								
150h	off	off	on	off	on	off	on	off
190h	off	off	on	off	off	on	on	off
200h	off	off	off	off	off	off	off	on
300h	off	off	off	off	off	off	on	on

Tabelle 1: Einstellung der Basisadressen

Die Karte kann nun gebootet und programmiert werden.
Der Anschluß des CAN-Bus erfolgt über den 9-pol. Sub-D-Stecker (unten). Die 9-pol. Sub-D-Buchse (oben) ist nicht belegt. Die nachfolgende Grafik zeigt die Pinbelegung des 9-pol-Sub-D-Steckers.

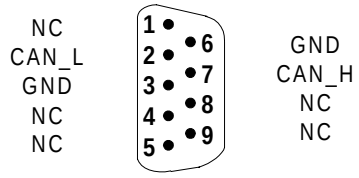

Abbildung 2: Pinbelegung des 9-pol. Sub-D-Steckers

Funktionsbeschreibung

Die ***ADwin-CAN***-Karte stellt die Verbindung zwischen dem Prozessormodul der ***ADwin***-Karte und dem CAN-Bus her. Die Karte ist mit dem CAN-Controller AN82527 bestückt. Die Funktionsweise der ***ADwin-CAN***-Karte basiert daher auf der Funktionsweise des CAN-Controllers. Auf die Register des Controllers kann aus ***ADbasic*** direkt zugegriffen werden.

Funktionsweise des CAN-Controllers AN82527

Der Controller enthält 255 Register. Diese Register dienen zum einen zur Konfiguration und Statusanzeige des CAN-Controllers. Hier werden Busgeschwindigkeit, Interrupthandling, usw. eingestellt. Des weiteren stellt der Controller 15 Message Objekte zur Verfügung. Über diese Message Objekte erfolgt die eigentliche Kommunikation. Ein Message Objekt kann zum Senden oder Empfangen von Nachrichten konfiguriert werden. Eine Ausnahme stellt hier das Message Objekt Nummer 15 dar. Dieses kann nur zum Empfangen von Daten genutzt werden. Das Message Objekt 15 hat im Gegensatz zu den anderen Objekten einen Puffer für eine weitere Nachricht. Die gepufferte Nachricht wird nachgeschoben, wenn die erste ausgelesen wurde.

Soll ein Message Objekt zum Empfangen von Nachrichten genutzt werden, so müssen die entsprechenden Bits zum Empfangen von Nachrichten gesetzt und ein Identifier vorgegeben werden. Anschließend werden alle ankommenden Nachrichten mit diesem Identifier in diesem Message Objekt gespeichert.

Ein Message Objekt kann auch für den Empfang von Nachrichten mit verschiedenen Identifiern genutzt werden. Dies kann mit Hilfe der globalen Maske realisiert werden (CAN-Register 6 und 7). Der Inhalt dieser Maske wird bitweise mit dem Identifier der ankommenden Nachricht verknüpft. Ist das Bit in der Maske „1“, muß das Identifierbit der ankommenden Nachricht mit dem Identifierbit im Message Objekt identisch sein, damit die Daten der ankommenden Nachricht im Message Objekt gespeichert werden. Ist ein Bit in der Maske „0“, ist es unerheblich für die Übernahme der Nachricht, ob die Identifierbits übereinstimmen oder nicht. Die nachfolgende Tabelle soll das Übernahmeverfahren für Nachrichten beispielhaft verdeutlichen. (x= Nachricht wird übernommen ; 0 = Nachricht wird nicht übernommen)
Die beiden niederwertigsten Bits der Maske sollen 0 sein, alle anderen Bits 1.

ID Message Objekt/ ID Nachricht	1	2	3	4
1	**x**	**x**	**x**	**0**
2	**x**	**x**	**x**	**0**
3	**x**	**x**	**x**	**0**
4	**0**	**0**	**0**	**x**

Tabelle 2 : Nachrichtenübernahme

Für das Message Objekt 15 existiert eine eigene Maske, die unabhängig von der anderen Maske ist (CAN-Register 12 bis 15).

Soll eine Nachricht gesendet werden, muß ein Message Objekt zum Senden konfiguriert werden, d.h. Bits die das Senden ermöglichen, werden gesetzt. Des weiteren ist der Identifier der Nachricht, die Anzahl der Datenbytes und die Daten selbst zu übergeben. Die Nachricht steht nun im Message Objekt zum Senden bereit und wird gesendet, sobald ein Zugriff auf den Bus möglich ist.

Ansteuerung des CAN-Controllers

Das Beschreiben und Auslesen der Register des CAN-Controllers wird über ein ASIC vorgenommen. Beide Vorgänge erfolgen in zwei Schritten. Zunächst wird die Adresse übergeben. Im zweiten Schritt werden die Daten übergeben, bzw. übernommen. Beim Auslesen der Daten muß zwischen Anlegen der Adresse und Übernehmen der Daten eine Zeit von mindestens 500ns vergehen. Diese Zeit benötigt der CAN-Controller um die Daten sicher auf die Datenleitungen zu legen.
Die Kommunikation kann wesentlich komfortabler mit den Funktionen erfolgen, die in der mitgelieferten INCLUDE-Datei enthalten sind. Die nachfolgende Grafik verdeutlicht den Datenfluß.

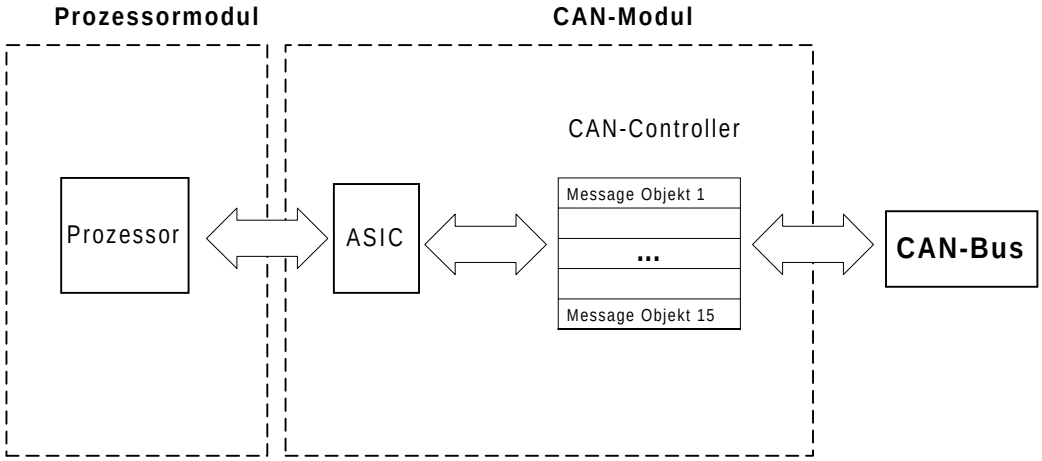

Abbildung 3 : Datenfluß über die *ADwin-CAN*-Karte

Hardwarespezifikationen der ADwin-CAN-Karte

Die Can-Bus-Frequenz hängt vom angelegten Takt und der Konfiguration des Controllers ab. Die externe Frequenz, die an den CAN-Controller angelegt ist beträgt 16 MHz.

Der Interruptausgang des CAN-Controllers ist mit dem Eventeingang des Prozessors verbunden. Dies eröffnet die Möglichkeit auf eintreffende Nachrichten sofort zu reagieren, ohne durch Polling den Prozessor zu belasten. Zur Nutzung dieser Funktion muß der Interrupt des CAN-Controllers freigegeben werden (Control-Register).

Der CAN-Controller arbeitet im Mode 1.
Der CAN-Bus ist vom PC mittels Optokopplern galvanisch getrennt.

Programmierung der *ADwin-CAN*-Karte

Die ***ADwin-CAN***-Karte kann mit dem Echtzeitentwicklungstool ***ADbasic*** komfortabel programmiert werden. Zur Programmierung der ***ADwin-CAN***-Karte steht eine INCLUDE-Datei zur Verfügung, die Funktionen zum Setzen und Lesen von Registern, zum Initialisieren von Message Objekten, zum Senden und Empfangen von Datensätzen und zur Initialisierung des CAN-Controllers enthält. Die Funktionen sind in der mitgelieferten INCLUDE-Datei „`can.inc`“ enthalten. Diese INCLUDE-Datei muß zunächst in das ***ADbasic***-Programm eingebunden werden. Dies geschieht mit der Befehlszeile

```
#include can.inc
```

Dieser Befehl muß am Programmanfang stehen. Zusätzlich muß in der Entwicklungsumgebung das Verzeichnis eingetragen werden, in dem die Include-Dateien stehen. Dies wird im Menüpunkt „Options/Directory“ eingetragen.
Nachfolgend werden die CAN spezifischen Funktionen erläutert.

SET_REG

Syntax: *SET_REG(REGISTERNUMMER,WERT)*

REGISTERNUMMER: Nummer des Registers im CAN-Controller
WERT: Datenbyte, das in das Register geschrieben werden soll

Beschreibt ein Register des CAN-Controllers mit einem Wert.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()
  set_reg(0,1)            'Setzt das Control Register auf den Wert 1
EVENT:
  ...
```

Hinweis:

Die Registernummer, die in diesem Befehl angegeben werden muß, entspricht der Registernummer des CAN-Controllers. D.h., daß das Controlregister mit der Adresse 0 angesprochen wird, das Statusregister mit der Adresse 1 usw., bis zum letzten Register des CAN-Controllers, das dann mit 255 angesprochen wird.

GET_REG

Syntax: *GET_REG(REGISTERNUMMER)*

REGISTERNUMMER: Nummer des Registers im CAN-Controller

Liest den Inhalt eines Registers des CAN-Controllers aus und übergibt diesen Wert einer Variablen.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()
  par_1 = get_reg(1)      'Liest das Statusregister aus und übergibt den ‘Wert an
                          den Parameter 1
EVENT:
  ...
```

Hinweis:

Die Registernummer, die in diesem Befehl angegeben werden muß, entspricht der Registernummer des CAN-Controllers. D.h., daß das Controlregister mit der Adresse 0 angesprochen wird, das Statusregister mit der Adresse 1 usw., bis zum letzten Register des CAN-Controllers, das dann mit 255 angesprochen wird.

INIT_CAN

Syntax: *INIT_CAN()*

Initialisiert die CAN-Controllerkarte und den CAN-Controller.
Der Befehl hat keinen Übergabe- und keinen Rückgabeparameter.
Dieser Befehl ist vor dem Arbeiten mit dem CAN-Controller notwendig, da mit ihm nicht nur der CAN-Controller der Karte initialisiert wird, sondern auch Adressen in ***ADbasic*** festgelegt werden, die für die Kommunikation mit dem CAN-Controller notwendig sind.
Der Befehl führt folgende Aktionen auf dem CAN-Controller aus:

- Reset
- Alle Filter werden auf Don't care gesetzt
- Clockoutregister wird auf 0 gesetzt (externe Frequenz wird nicht geteilt)
- Bus Configuration Register wird auf 0 gesetzt
- Die Übertragungsrate für den CAN-Bus wird auf 125 kBit/s gesetzt
- Alle Message Objekte werden gesperrt

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()              'Initialisierung des CAN-Controllers
EVENT:
  ...
```

Hinweis:

Der Befehl muß zu Beginn des Programms ausgeführt werden.

EN_RECEIVE

Syntax: *EN_RECEIVE(Objektnummer, Identifier)*

Objektnummer:	Nummer des Message Objekts im CAN-Controller
Identifier:	Identifier der Nachrichten, die in diesem Message Objekt empfangen werden sollen

Gibt ein Message Objekt zum Empfangen von Nachrichten frei und trägt den Identifier (11 Bit) ein, der dem Message Objekt zugeordnet werden soll.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()
  en_receive(1,200)       'Initialisiert das Message Objekt 1 zum Empfangen
                          'von CAN-Nachrichten mit dem Identifier 200
EVENT:
  ...
```

Hinweis:

Ein Message Objekt kann nur Nachrichten vom CAN-Bus empfangen, wenn es zuvor mit „EN_RECEIVE" freigegeben wurde. Wird ein Message Objekt mit der Funktion „EN_TRANSMIT" und „ EN_RECEIVE" angesprochen, so ist der Aufruf ausschlaggebend, der zuletzt ausgeführt wurde.

EN_RECEIVE_29

Syntax: *EN_RECEIVE(Objektnummer, Identifier)*

Objektnummer: Nummer des Message Objekts im CAN-Controller
Identifier: Identifier der Nachrichten, die in diesem Message Objekt empfangen werden sollen

Gibt ein Message Objekt zum Empfangen von Nachrichten frei und trägt den Identifier (29 Bit) ein, der dem Message Objekt zugeordnet werden soll.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()
  en_receive_29(1,200)   'Initialisiert das Message Objekt 1 zum Empfangen
                         'von CAN-Nachrichten mit dem Identifier 200
EVENT:
  ...
```

Hinweis:

Ein Message Objekt kann nur Nachrichten vom CAN-Bus empfangen, wenn es zuvor mit „EN_RECEIVE_29“ freigegeben wurde. Wird ein Message Objekt mit der Funktion „EN_TRANSMIT“ und „EN_RECEIVE“ angesprochen, so ist der Aufruf ausschlaggebend, der zuletzt ausgeführt wurde.

EN_TRANSMIT

Syntax: *EN_TRANSMIT(Objektnummer, Identifier)*

Objektnummer: Nummer des Message Objekts im CAN-Controller
Identifier: Identifier, den die Nachrichten haben sollen, die von diesem Message Objekt abgesendet werden sollen

Gibt ein Message Objekt zum Senden frei und trägt den Identifier (11 Bit) ein, der dem Message Objekt zugeordnet werden soll.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()
  en_transmit(6,40)      'Initialisiert das Message Objekt 6 zum Senden
                         'von CAN-Nachrichten mit dem Identifier 40
EVENT:
  ...
```

Hinweis:

Ein Message Objekt kann nur Nachrichten auf den CAN-Bus senden, wenn es zuvor mit „EN_TRANSMIT“ freigegeben wurde. Wird ein Message Objekt mit der Funktion „EN_TRANSMIT“ und „EN_RECEIVE“ angesprochen, so ist der Aufruf ausschlaggebend, der zuletzt ausgeführt wurde.

EN_TRANSMIT_29

Syntax: *EN_TRANSMIT(Objektnummer, Identifier)*

Objektnummer: Nummer des Message Objekts im CAN-Controller
Identifier: Identifier, den die Nachrichten haben sollen, die von diesem Message Objekt abgesendet werden sollen

Gibt ein Message Objekt zum Senden frei und trägt den Identifier (29 Bit) ein, der dem Message Objekt zugeordnet werden soll.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()
  en_transmit_29(6,40)    'Initialisiert das Message Objekt 6 zum Senden
                          'von CAN-Nachrichten mit dem Identifier 40
EVENT:
  ...
```

Hinweis:

Ein Message Objekt kann nur Nachrichten auf den CAN-Bus senden, wenn es zuvor mit „EN_TRANSMIT_29“ freigegeben wurde. Wird ein Message Objekt mit der Funktion „EN_TRANSMIT“ und „EN_RECEIVE“ angesprochen, so ist der Aufruf ausschlaggebend, der zuletzt ausgeführt wurde.

TRANSMIT

Syntax: *TRANSMIT(MSG_num)*

MSG_num: Nummer des Message Objekts im CAN-Controller

Sendet ein Datentelegramm über den CAN-Bus, sobald das Message Objekt Zugriffsrecht hat. Die Anzahl der Datenbytes und die Datenbytes selbst für diese Nachricht werden aus dem Feld „can_msg“ entnommen. Dieses Feld hat neun Elemente. Die ersten acht Elemente enthalten die Datenbytes eins bis acht. Das neunte Element steht für die Anzahl der Datenbytes. Hier muß ein Wert zwischen eins und acht eingetragen werden. Die Werte im Feld „can_msg“ müssen vor der Ausführung des „transmit“ Befehls eingetragen werden.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  Init_can()
  en_transmit(6,40)       'Initialisiert das Message Objekt 6 zum senden
                          'von CAN-Nachrichten mit dem Identifier 40
  can_msg[1] = 100        'Dem Feld werden die Werte übergeben, die
  can_msg[2] = 200        'später gesendet werden sollen. Die Nachricht
  can_msg[9] = 2          'enthält 2 Datenbytes mit den Werten 100 und 200
EVENT:
  transmit(6)             'sendet eine Nachricht, über Message
                          'Objekt 6 mit dem Identifier 40
```

Hinweis:

Dieser Befehl erfüllt seine Funktion nur, wenn das entsprechende Message Objekt zuvor mit „EN_TRANSMIT“ zum Senden konfiguriert wurde.

READ_MSG

Syntax: *READ_MSG(MSG_nummer)*

MSG_num: Nummer des Message Objekts im CAN-Controller

Liest eine empfangene Nachricht aus dem Message Objekt aus. Dabei wird zunächst überprüft, ob seit der letzten Abfrage eine neue Nachricht für dieses Message Objekt angekommen ist. Ist dies nicht der Fall werden keine weiteren Register ausgelesen und der Rückgabewert ist 0. Wenn eine neue Nachricht in das Message Objekt geschrieben wurde, werden die Daten ausgelesen und in das vordefinierte Feld „can_msg“ übertragen. Der Rückgabewert entspricht in diesem Fall dem Identifier der empfangenen Nachricht. Die Bits, die den Empfang einer neuen Nachricht anzeigen, werden zurückgesetzt, so daß eine weitere Nachricht empfangen werden kann.

Anwendungsbeispiel:

```
#INCLUDE CAN.INC
INIT:
  init_can()
  en_transmit(1,200)      'Initialisiert das Message Objekt 1 zum Einlesen
                          'von CAN-Nachrichten mit dem Identifier 200
EVENT:
  par_9 = read_msg(1)     'Liest die empfangenen Daten aus dem Message Objekt 1
                          'aus und übergibt den Identifier an den Parameter 9.
                          'Die Datenbytes stehen im Feld can_msg
```

Hinweis:

Das entsprechende Message Objekt muß zuvor zum Empfangen freigegeben werden, mit dem Befehl „en_receive“.

Beispielprogramme

Das nachfolgende Programm zeigt die Initialisierung des CAN-Controllers im Init-Teil und das zyklische Auslesen und Senden im Event-Teil:

```
REM Programm initialisiert den CAN-Controller,
REM konfiguriert ein Message Objekte als Sender
REM und eins als Empfänger. Das Programm tauscht
REM alle 10 ms Daten zwischen CAN-Controller und
REM Transputer aus.

#include can.inc

dim ergebnis as integer

init:
  init_can                    'Initialisierung des CAN-Controllers

 REM Filter einschalten
  set_reg(6,255)              'Global Mask Register Standard
  set_reg(7,255)

  set_reg(12,255)             'Message 15 Mask Register
  set_reg(13,255)
  set_reg(14,255)
  set_reg(15,255)

  set_reg(0,65)               ‘Schreibrecht für CPU auf
                              ‘Konfigurationsregister
  set_reg(63,0)               'Bus Übertragungsrate
  set_reg(79,20)              'aus 1 MBit/s
  set_reg(0,0)                'Schreibrecht wird zurückgegeben

  en_receive(2,385)           'Message Objekt 2 wird zum Lesen
                              'konfiguriert Identifier 385

  en_transmit(3,1)            'Message Objekt 3 wird zum Schreiben
                              'konfiguriert Identifier 1


event:

  REM 1 Datensatz lesen und 1 Datensatz schreiben

  ergebnis = read_msg(2)      'Daten einlesen
                              'Wenn es neue Daten gibt werden sie in
                              'das Feld can_msg geschrieben


  can_msg[1]=1                'Daten, die gesendet werden sollen
  can_msg[2]=2                'werden in das Feld can_msg
  can_msg[3]=3                'geschrieben. Aus diesem werden
  can_msg[4]=4                'beim späterer Senden die Daten
  can_msg[5]=5                'entnommen.
  can_msg[6]=6
  can_msg[7]=7
  can_msg[8]=8
  can_msg[9]=8                '8 Datenbytes

  transmit(3)                 'Nachricht senden
```

Das nachfolgende Beispielprogramm zeigt die Initialisierung des CAN-Controllers und das nachfolgende interruptgesteuerte Auslesen von neuen Nachrichten:

```
REM Programm initialisiert den CAN-Controller
REM und konfiguriert ein Message Objekte als
REM Empfänger. Das Programm liest
REM interruptgesteuert Nachrichten ein,
REM sobald eine Neue eintrifft.

#include can.inc

dim ergebnis,status,objekt as integer

init:
  init_can                    'Initialisierung des CAN-Controllers

  REM Filter setzen
  set_reg(6,255)              'Global Mask Register Standard
  set_reg(7,255)

  set_reg(12,255)             'Message 15 Mask Register
  set_reg(13,255)
  set_reg(14,255)
  set_reg(15,255)

  en_receive(1,385)           'Message Objekt 1 wird zum Lesen konfiguriert
                              'Nur Nachrichten mit dem Identifier 385
                              'werden gespeichert

  set_reg(0,65)               'Schreibrecht für CPU auf
                              'Konfigurationsregister
  set_reg(63,0)               'Bus Übertragungsrate
  set_reg(79,20)              'auf 1 MBit/s
  set_reg(0,0)                'Schreibrecht wird zurückgegeben

  status = get_reg(1)         'Status einlesen
  set_reg(0,2)                'Interrupt enable; auf CAN-Controller


event:

  objekt = get_reg(5fh)       'Interruptregister lesen

  if (objekt = 2) then        'Daraus die Nummer des Message
    objekt = 15               'Objekts ermitteln
  else                        `in dem die neue Nachricht steht
    objekt = objekt - 2
  endif

  ergebnis = read_msg(objekt) 'Neue Daten auslesen
                              'Die Daten stehen im Feld can_msg
```