Ingenieurbüro für Echtzeitprogrammierung

CAN-BusTreiber

Dok-Rev. 2.6 vom 13.05.2019

Software-Rev. 7.6 vom 27.03.2019

Inhaltsverzeichnis

1 Urheberrecht und Haftung 4
1.1 Handhabung 4
1.2 Erklärung 4

2 Allgemeine Beschreibung 5
2.1 Behandlung von RTR-Frames 6

3 Initialisierung 7

4 Empfangsfenster setzen 10

5 Lesen von CAN-Messages 11
5.1 Lesen einer CAN-Message 11
5.2 Lesen mehrerer CAN-Messages 12

6 Setzen eines Timeout beim Senden 13

7 Schreiben von CAN-Messages 14
7.1 Schreiben einer CAN-Message 14
7.2 Schreiben mehrerer CAN-Messages 14

8 Abfrage der Anzahl Empfangsmessages 16

9 Identifier setzen (Softwarefilterung) 17

10 Identifier löschen (Softwarefilterung) 18

11 RTR-Answer Buffer einrichten 19

12 Feature auslesen 20

13 Alle Fehlermeldungen im Überblick 21

14 Ethernet über CAN 22
14.1 Initialisierung 22

Revisionsliste:

Rev.	Datum	Na.	Änderung
1.0	11.07.1996	Ko	Erstellung
1.1	06.07.2000	Ko	Umstellung auf software.dot
1.2	19.03.2002	Ko	PEARL-Prozeduren in Großbuchstaben
1.3	11.06.2002	Ko	gültig ab Software Rev. 3.0; Umstellung auf CAN2.0B aktiv, Ethernet über den CAN-Bus
1.4	26.03.2002	Ko	Sync-Seg ergänzt
1.5	11.03.2003	Ko	Softwarefilterung auf IR-Ebene ergänzt
1.6	09.07.2003	Ko	Senden/Empfangen von *n* CAN-Nachrichten eingefügt
1.7	15.07.2003	Ko	Parameter bei `can_n_write` geändert
1.8	21.07.2003	Ko	CIA-Empfehlung beim MPC555 ergänzt
1.9	18.03.2004	Ko	`"ENTRY"` fehlte bei den PEARL-SPC's
2.0	16.08.2006	Ko	Rechtschreibung und `can_eth_init` ergänzt
2.1	28.02.2007	Ko	Neue Funktionen `RTR_Answer`, `get_feature`
2.2	06.02.2008	Ko	Beschreibung CAN-Init überarbeitet
2.3	14.05.2009	Kr	Bemerkung zum Timeout
2.4	19.08.2014	Kr	Warnung beim Aufbau des Datentyps und Compiler Parameter ergänzt
2.5	21.01.2019	Ko	Korrektur Timeout-Parameter
2.6	13.05.2019	Ko	can_get_feature Beschreibung angepasst

1 Urheberrecht und Haftung

Alle Rechte an diesen Unterlagen liegen bei der IEP GmbH, Langenhagen.

Die Vervielfältigung, auch auszugsweise, ist nur mit unserer ausdrücklichen schriftlichen Genehmigung zulässig.

In Verbindung mit dem Kauf von Software erwirbt der Käufer einfaches, nicht übertragbares Nutzungsrecht. Dieses Recht zur Nutzung bezieht sich ausschließlich darauf, daß dieses Produkt auf oder in Zusammenhang mit jeweils **einem** Computer zu benutzen ist. Das Erstellen einer Kopie ist ausschließlich zu Archivierungszwecken unter Aufsicht des Käufers oder seines Beauftragten zulässig. Der Käufer haftet für Schäden, die sich aus der Verletzung seiner Sorgfaltspflicht ergeben, z.B. bei unautorisiertem Kopieren, unberechtigter Weitergabe der Software usw.. Der Käufer gibt mit dem Erwerb der Software seine Zustimmung zu den genannten Bedingungen. Bei unlizensiertem Kopieren muß vorbehaltlich einer endgültigen juristischen Klärung von Diebstahl ausgegangen werden. Dies gilt ebenso für Dokumentation und Software, die durch Modifikation aus Unterlagen und Programmen von IEP hervorgegangen ist, gleichgültig, ob die Änderungen als geringfügig oder erheblich anzusehen sind.

Eine Haftung seitens IEP für Schäden, die auf den Gebrauch von Software, Hardware oder Benutzung dieses Manuskriptes zurückzuführen sind, wird ausdrücklich ausgeschlossen, auch für den Fall fehlerhafter Software oder irrtümlicher Angaben.

Das Einverständnis des Käufers oder Nutzers für den Haftungsausschluß gilt mit dem Kauf und der Nutzung der Software und dieser Unterlagen als erteilt.

1.1 Handhabung

Lesen Sie bitte zuerst sorgfältig diese Dokumentation bevor Sie anfangen zu programmieren. Sie sparen Zeit und vermeiden Probleme.

1.2 Erklärung

Wir behalten uns das Recht vor, Änderungen, die einer Verbesserung der Schaltung oder des Produktes dienen, ohne besondere Hinweise vorzunehmen. Trotz sorgfältiger Kontrolle kann für die Richtigkeit der hier gegebenen Daten, Schaltpläne, Programme und Beschreibungen keine Haftung übernommen werden. Die Eignung des Produktes für einen bestimmten Einsatzzweck wird nicht zugesichert.

2 Allgemeine Beschreibung

Die Ansteuerung der CAN-Bausteine erfolgt über eine Prozedurschnittstelle. Es stehen die im folgenden beschriebenen Funktionen zur Verfügung. Zuerst wird die C, dann die PEARL-Syntax dargestellt. Zum Senden und Empfangen wird die Struktur `CAN_MESSAGE` mit folgendem Aufbau genutzt:

```
typedef struct CAN_MESSAGE
{
   ULONG  identifier  ;
   WORD   frame_format;
   WORD   rtr         ;
   WORD   data_length ;
   UBYTE  data[8]     ;
}
   CAN_MESSAGE ;
```

```
TYPE CAN_MESSAGE STRUCT
[
   identifier   FIXED(31) ,
   frame_format FIXED(15) ,
   rtr          FIXED(15) ,
   data_length  FIXED(15) ,
   data         CHAR(8)
] ;
```

Wird für `frame_format` der kurze Identifier (11 Bit, CAN2.0A) gewählt, darf `identifier` im Bereich 0....2032 liegen. Bei der Wahl des langen Identifier (29 Bit, CAN2.0B) ist der Bereich von 0....536870911 (0x1FFFFFFF) zulässig. Für `frame_format` sind die Werte `FRAME_SHORT=0` oder `FRAME_LONG=1` zulässig. Mit `rtr=1` wird eine Remote Datenübertragung angefordert. Sonst ist `rtr` auf `0` zu setzen. In der `data_length` ist die Anzahl der Datenbytes angegeben, zulässig sind Werte zwischen 0 und 8. In `data` sind die Nutzdaten abgelegt. `data` muß **nicht** als `CHAR(8)` abgelegt werden, nur die Länge **muß** 8 Byte sein! So könnten z.B. für `data` auch 2 `FIXED(31)` Variablen angelegt werden.

Bitte beachten Sie:

Eine CHAR(1)/data[1] wird grundsätzlich auf ein Wort aufgerundet! Es ist also nicht zulässig, einen 8 Bit langen Datentyp einzeln bzw. mit einer ungeraden Länge zu verwenden, wenn danach noch weitere Daten kommen!
Ein PEARL-Programm darf nicht mit `MODE=PAD` übersetzt werden bzw. ein C-Programm muß mit `#pragma MEMBER_PADDING_68K` und `#pragma STRUCT_SIZE_WORD` übersetzt sein!
Werden die obigen Regeln nicht beachtet, so kann fremder Speicher überschrieben werden und es können undefinierte Abstürze des gesamten Systems auftreten!

Diese Beschreibung ersetzt nicht das Studium des Datenblattes des CAN-Controllers. Genauso wenig werden Kenntnisse über die Funktionsweise des CAN-Busses vermittelt.

Bitte beachten Sie, dass diese Beschreibung erst für CAN-Treiber ab Release 3.0 gültig ist.

2.1 Behandlung von RTR-Frames

RTR-Frames werden je nach CAN-Controller unterschiedlich gehandhabt. Das Verschicken einer CAN-Message mit gesetztem RTR-Flag ist mit allen Controllern gleich: es wird ein `CAN_WRITE` mit gesetztem RTR-Flag aufgerufen. Daten können in einem RTR-Telegramm nicht verschickt werden, die Datenlänge muß auf `0` gesetzt werden.

Beim Empfang von CAN-Messages mit gesetztem RTR-Flag verhalten sich die Controller unterschiedlich. Der SJA1000 sowie der MCP2515 empfangen diese Message normal, so dass sie mit einem `CAN_READ` gelesen werden kann.

Die Familie der PowerPC-CAN-Interfaces (Ausnahme: MPC5200 wie SJA1000) verhält sich anders: Um auf eine RTR-Message zu antworten, muß vorher schon eine passende Antwort bereit gestellt werden! Steht keine Antwort bereit, so wird die RTR-Message verworfen, sie kann nicht mit einem `CAN_READ` gelesen werden.

3 Initialisierung

Vor dem Aufruf anderer Funktionen muß die CAN-Schnittstelle initialisiert werden. Das geschieht mit der Funktion `can_init`. Dabei wird ein Reset auf dem CAN-Baustein ausgelöst, es wird festgelegt, welches Interface genutzt werden soll, wie groß die Baudrate ist und wie viele Plätze der Empfangspuffer hat. Der Empfangspuffer wird gelöscht.

```
WORD can_init( WORD can_nr, CAN_INIT *can_init_ptr ) ;

SPC CAN_INIT ENTRY ( FIXED(15), CAN_INIT IDENT )
                    RETURNS( FIXED(15) ) GLOBAL;
```

Die Struktur `CAN_INIT` hat folgenden Aufbau:

```
typedef struct CAN_INIT
{
   ULONG  can_clock  ;
   UWORD  phase_seg1 ;
   UWORD  phase_seg2 ;
   UWORD  samp       ;
   UWORD  sjw        ;
   UWORD  baudrate   ;
   WORD   anz        ;
}
   CAN_INIT ;
```

```
TYPE CAN_INIT STRUCT
[
   can_clock    FIXED(31) ,
   phase_seg1   FIXED(15) ,
   phase_seg2   FIXED(15) ,
   samp         FIXED(15) ,
   sjw          FIXED(15) ,
   baudrate     FIXED(15) ,
   anz          FIXED(15)
] ;
```

Die `can_nr` gibt die Nummer der CAN-Schnittstelle an, zulässig sind z.Z. die Werte 1 bis Anzahl der CAN-Schnittstellen. Um die Baudrate einzustellen, stehen 2 Möglichkeiten zur Verfügung:

1) Sie wollen eine Baudrate mit den Einstellungen, die von der CIA empfohlen sind, einsetzen. Dann müssen Sie nur in `baudrate` einen der folgenden Werte eintragen. Alle anderen Angaben werden dann ignoriert bzw. automatisch gesetzt:

Wert	Baudrate	Nominal bit time	Number of time quanta per bit	Length of time quantum	Location of sample point
8	1 MBit	1 µs	8	125 ns	6 (750 ns)
7	800 kBit	1,25 µs	10	125 ns	8 (1 µs)
6	500 kBit	2 µs	16	125 ns	14 (1.75 µs)
5	250 kBit	4 µs	16	250 ns	14 (3.5 µs)
4	125 kBit	8 µs	16	500 ns	14 (7 µs)
3	50 kBit	20 µs	16	1,25 µs	14 (17.5 µs)
2	20 kBit	50 µs	16	3,125 µs	14 (43.75 µs)
1	10 kBit	100 µs	16	6,25 µs	14 (87.5 µs)
0	keine Baudrate				

Die obigen Werte können mit einem MPC555 aufgrund der Taktfrequenz von 40 MHz nicht eingestellt werden. Für den MPC555 gilt die folgende Tabelle:

Wert	Baudrate	Nominal bit time	Number of time quanta per bit	Length of time quantum	Location of sample point
8	1 MBit	1 µs	20	50 ns	15 (750 ns)
7	-	-	-	-	-
6	500 kBit	2 µs	20	100 ns	17 (1.7 µs)
5	250 kBit	4 µs	16	250 ns	14 (3.5 µs)
4	125 kBit	8 µs	16	500 ns	14 (7 µs)
3	50 kBit	20 µs	20	1 µs	17 (17 µs)
2	20 kBit	50 µs	20	2,5 µs	17 (42.5 µs)
1	10 kBit	100 µs	20	5 µs	17 (85 µs)
0	keine Baudrate				

2) Haben Sie bei `baudrate` eine `0` eingetragen, so müssen Sie alle anderen Parameter mit sinnvollen Werten füllen:

   `can_clock` gibt die Länge eines Timequantums in ns an.

   `phase_seg1` sind die Anzahl der Timequanten bis zum Abtastzeitpunkt.

   `phase_seg2` ist die Anzahl der Timequanten nach dem Abtastzeitpunkt. Die beiden phase_seg-Werte zusammen plus 1 für das Sync-Seg ergeben die number of time quanta per bit.

   `samp` legt die Anzahl der Samples beim Abtasten fest, zulässig sind 1 oder 3.

   `sjw` bestimmt die synchronisation jump width, zulässig sind Werte von 1 - 4.

`anz` gibt die Anzahl Plätze des Empfangspuffers an, max. werden 64 Plätze unterstützt. Die Akzeptanzmaske wird so gesetzt, daß alle Messages empfangen werden können.

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
E_OK	0	kein Fehler
E_NO_CAN	-1	CAN-Baustein nicht verfügbar
E_WRONG_PHYS	-3	Interface unzulässig
E_WRONG_READ_BUFFER	-4	Empfangspuffergröße unzulässig
E_NO_MEMORY	-14	kein Speicher für Empfangspuffer
E_SAMP	-17	Der Wert ist nicht 1 oder 3
E_BAUDRATE	-18	Es wurde eine nicht unterstützte Baudrate eingegeben
E_PHASE_SEG1	-19	Wert außerhalb des Bereiches 1-16
E_PHASE_SEG2	-20	Wert außerhalb des Bereiches 1-8
E_CAN_CLOCK	-21	Wert außerhalb des Bereiches (bausteinabhängig)
E_JUMP_WIDTH	-22	Wert außerhalb des Bereiches 1-4
E_RESET_IMPOSS	-23	der CAN-Baustein macht keinen Reset

Wird ein Fehler zurückgegeben, so ist der CAN-Baustein nicht sende-/empfangsbereit, d.h. die Initialisierung muß wiederholt werdend.

Beispiel: Einstellung der 500 kBit Baudrate nach CIA Empfehlung:

```
can_clock  =  125   ; // Time quantum = 125 ns
phase_seg1 =   13   ; // Location of Sample Point
phase_seg2 =    2   ; // Number of Time Quanta per Bit: 1+13+2=16
samp       = SAMP_1 ; // 1 Sample nehmen
sjw        =    1   ; // Synchronisation Jump Width
```

4 Empfangsfenster setzen

Nur Messages, deren Identifier akzeptiert werden, werden in den Empfangspuffer eingetragen. Beim Aufruf dieser Funktion wird der Empfangspuffer gelöscht. Da die Möglichkeiten von der Hardware abhängig sind, wurde für C-Programmierer eine UNION angelegt, die bei neuen CAN-Bausteinen entsprechend erweitert wird. Pearl-Programmierer müssen sich eine Struktur anlegen, die der von Ihnen gewünschten Betriebsart entspricht.

```
WORD can_set_accept( WORD can_nr, CAN_SET_ACCEPT *paccept ) ;

SPC CAN_SET_ACCEPT ENTRY ( FIXED(15), CAN_SET_ACCEPT IDENT )
                           RETURNS( FIXED(15) ) GLOBAL ;
```

Der Aufbau der Struktur `CAN_SET_ACCEPT` ist der Datei `canproto.h` zu entnehmen. Die Bedeutung der Parameter ist den jeweiligen Datenblättern zu entnehmen.

Als Rückgabewerte können auftreten:

Name	Wert	Bedeutung
`E_OK`	0	kein Fehler
`E_NO_CAN`	-1	CAN-Baustein nicht verfügbar
`E_NO_INIT`	-5	keine Initialisierung durchgeführt

Wenn Sie Ethernet über CAN fahren, müssen die für die Ethernet-Pakete gewählten Identifier von der Acceptanz-Maske mit abgedeckt werden.

5 Lesen von CAN-Messages

5.1 Lesen einer CAN-Message

Die Verwaltung des Empfangspuffers wird von der CAN-Betreuungstask durchgeführt. Beim Lesen mit `can_read` wird die übergebene Struktur gefüllt. Der Aufrufer wird bis zum Eintreffen einer Message blockiert. Beim Aufruf der `can_init`-Funktion werden evt. noch wartende Leser freigegeben.

```
WORD can_read( WORD can_nr, CAN_MESSAGE *read_ptr ) ;

SPC  CAN_READ ENTRY ( FIXED(15), CAN_MESSAGE IDENT )
                      RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
	>0	Anzahl noch wartender Messages
E_OK	0	kein Fehler
E_NO_CAN	-1	CAN-Baustein nicht verfügbar
E_ERROR_LEVEL	-2	Warning Level des 82C200 erreicht
E_NO_INIT	-5	keine Initialisierung durchgeführt
E_OFF_BUS	-6	Off-Bus
E_OVERRUN	-7	Overrun aufgetreten
E_READ_BUF_OVERRUN	-8	Empfangspuffer übergelaufen
E_RESET	-9	Reset ausgelöst (mit `can_init`)

Wird ein Wert größer Null zurückgegeben, so liegen noch weitere Nachrichten im Empfangspuffer vor, die sofort abgeholt werden können.

Bei den Fehlern `E_OVERRUN` und `E_READ_BUF_OVERFLOW` sind Empfangsdaten verlorengegangen! Es sind aber gültige Daten von der Schnittstelle gelesen und in der übergebenen Struktur eingetragen worden. Diese beiden Fehlermeldungen werden nur einmal übergeben, außer sie treten erneut auf. `E_OVERRUN` heißt, daß die Daten nicht schnell genug vom CAN-Baustein abgeholt worden sind. Das kann nur passieren, wenn das System zulange im "OFF" ist oder eine Interruptroutine auf höherem Level zu lange läuft. `E_READ_BUF_OVERRUN` zeigt an, daß der Empfangspuffer zu klein ist bzw. zu selten Daten von der CAN-Schnittstelle abgeholt wurden. Bei dem Fehler `E_ERROR_LEVEL` ist der Warning-Level des CAN-Bausteins erreicht, die Daten wurden aber trotzdem korrekt übertragen.

5.2 Lesen mehrerer CAN-Messages

Die Verwaltung des Empfangspuffers wird von der CAN-Betreuungstask durchgeführt. Beim Lesen mit `can_n_read` wird die übergebene Struktur gefüllt. Der Aufrufer wird bis zum Eintreffen einer Message blockiert. Beim Aufruf der `can_init`-Funktion werden evt. noch wartende Leser freigegeben. Es können mehrere Messages auf einmal abgeholt werden. Die maximale Anzahl wird als 2.ter Parameter (`max`) übergeben. Der Aufrufer muß den benötigten Platz für die Messages zur Verfügung stellen, steht weniger Platz als mit `max` festgelegt zur Verfügung, werden andere Daten überschrieben! Über den Pointer `p_anz` wird die Anzahl der tatsächlich gelesenen Messages zurückgegeben.

```
WORD can_n_read( WORD can_nr, WORD max,
                 WORD *p_anz, CAN_MESSAGE *read_ptr ) ;

SPC  CAN_N_READ ENTRY ( FIXED(15), FIXED(15),
                        FIXED(15) IDENT, CAN_MESSAGE() IDENT )
                        RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt die Anzahl noch vorhandener Messages bzw. eine Fehlernummer zurück:

Name	Wert	Bedeutung
	>0	Anzahl noch wartender Messages
`E_OK`	0	kein Fehler
`E_NO_CAN`	-1	CAN-Baustein nicht verfügbar
`E_ERROR_LEVEL`	-2	Warning Level erreicht
`E_NO_INIT`	-5	keine Initialisierung durchgeführt
`E_OFF_BUS`	-6	Off-Bus
`E_OVERRUN`	-7	Overrun aufgetreten
`E_READ_BUF_OVERRUN`	-8	Empfangspuffer übergelaufen
`E_RESET`	-9	Reset ausgelöst (mit `can_init`)

Wird ein Wert größer Null zurückgegeben, so liegen noch weitere Nachrichten im Empfangspuffer vor, die sofort abgeholt werden können. Es werden nicht mehr als `max` Messages zurückgegeben.

Bei den Fehlern `E_OVERRUN` und `E_READ_BUF_OVERFLOW` sind Empfangsdaten verlorengegangen! Es sind aber gültige Daten von der Schnittstelle gelesen und in der übergebenen Struktur eingetragen worden. Diese beiden Fehlermeldungen werden nur einmal übergeben, außer sie treten erneut auf. `E_OVERRUN` heißt, daß die Daten nicht schnell genug vom CAN-Baustein abgeholt worden sind. Das kann nur passieren, wenn das System zulange im "OFF" ist oder eine Interruptroutine auf höherem Level zu lange läuft. `E_READ_BUF_OVERRUN` zeigt an, daß der Empfangspuffer zu klein ist bzw. zu selten Daten von der CAN-Schnittstelle abgeholt wurden. Bei dem Fehler `E_ERROR_LEVEL` ist der Warning-Level des CAN-Bausteins erreicht, die Daten wurden aber trotzdem korrekt übertragen.

6 Setzen eines Timeout beim Senden

Mit der `can_set_timeout`-Funktion kann ein Timeout beim Senden angegeben werden.

```
WORD can_set_timeout( WORD can_nr, LONG timeout ) ;

SPC  CAN_SET_TIMEOUT ENTRY ( FIXED(15), FIXED(31) )
                             RETURNS( FIXED(15) ) GLOBAL ;
```

`timeout` ist in Millisekunden anzugeben.

Bei PEARL-Programmen sollte immer eine FIXED(31) Variable mit dem Wert der gewünschten msec genutzt werden. Damit Unabhängigkeit von der Einstellung MODE=CLOCK50 erreicht wird.

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
`E_OK`	0	kein Fehler
`E_NO_CAN`	-1	CAN-Baustein nicht verfügbar
`E_NO_INIT`	-5	keine Initialisierung durchgeführt

Ein Timeout kann z.B. auftreten, wenn kein weiterer Teilnehmer am Bus ist, da dann keine Bestätigung für die erfolgreiche Versendung erfolgt und der CAN-Baustein ununterbrochen sendet. Im Fehlerfall wird die Sendung abgebrochen und der Write-Auftrag mit einer entsprechenden Fehlermeldung zurückgegeben.

> !!! Achtung das eingestellte Timeout zählt für den gesamten `CAN_WRITE` bzw. `CAN_N_WRITE` Aufruf.

7 Schreiben von CAN-Messages

7.1 Schreiben einer CAN-Message

Mit der `can_write`-Funktion wird eine Message verschickt.

```
WORD can_write( WORD can_nr, CAN_MESSAGE *write_ptr ) ;

SPC  CAN_WRITE ENTRY ( FIXED(15), CAN_MESSAGE IDENT )
                       RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
`E_OK`	0	kein Fehler
`E_NO_CAN`	-1	CAN-Baustein nicht verfügbar
`E_ERROR_LEVEL`	-2	Warning Level des 82C200 erreicht
`E_NO_INIT`	-5	keine Initialisierung durchgeführt
`E_OFF_BUS`	-6	Off-Bus
`E_RESET`	-9	Reset ausgelöst (mit `can_init`)
`E_IDENTIFIER`	-10	Identifier unzulässig
`E_LENGTH`	-11	Länge unzulässig
`E_RTR`	-13	`rtr`-Wert unzulässig
`E_TIMEOUT`	-15	Timeout beim Senden

Bei Rückgabe von `E_ERROR_LEVEL` wurde die Nachricht erfolgreich verschickt. Bei allen anderen Fehlern wurde die Nachricht nicht verschickt.

7.2 Schreiben mehrerer CAN-Messages

Mit der `can_n_write` -Funktion können mehrere Messages verschickt werden. In `anz` wird übergeben, wie viele Nachrichten verschickt werden sollen. Im Fehlerfall kann in `p_anz` nachgesehen werden, wie viele Nachrichten bis zum Auftreten des Fehlers verschickt worden sind.

```
WORD can_n_write( WORD can_nr, WORD anz,
                  WORD *p_anz, CAN_MESSAGE *write_ptr );

SPC  CAN_N_WRITE ENTRY ( FIXED(15), FIXED(15),
                         FIXED(15) IDENT, CAN_MESSAGE IDENT )
                         RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
`E_OK`	0	kein Fehler
`E_NO_CAN`	-1	CAN-Baustein nicht verfügbar
`E_ERROR_LEVEL`	-2	Warning Level erreicht
`E_NO_INIT`	-5	keine Initialisierung durchgeführt
`E_OFF_BUS`	-6	Off-Bus
`E_RESET`	-9	Reset ausgelöst (mit `can_init`)
`E_IDENTIFIER`	-10	Identifier unzulässig
`E_LENGTH`	-11	Länge unzulässig
`E_RTR`	-13	`rtr`-Wert unzulässig
`E_TIMEOUT`	-15	Timeout beim Senden

Bei Rückgabe von `E_ERROR_LEVEL` wurden die Nachrichten erfolgreich verschickt. Bei allen anderen Fehlern wurde ggf. ein Teil der Nachrichten nicht verschickt. Wie viele Nachrichten verschickt wurden, kann über `p_anz` ermittelt werden.

8 Abfrage der Anzahl Empfangsmessages

Es wird die Anzahl noch im Empfangspuffer stehender Messages zurückgegeben.

```
WORD can_nr_of_messages( WORD can_nr, WORD *nr_of_message_ptr ) ;

SPC  CAN_NR_OF_MESSAGES ENTRY ( FIXED(15), FIXED(15) IDENT )
                                RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
E_OK	0	kein Fehler
E_NO_CAN	-1	CAN-Baustein nicht verfügbar
E_NO_INIT	-5	keine Initialisierung durchgeführt
E_OFF_BUS	-6	Off-Bus
E_OVERRUN	-7	Overrun aufgetreten
E_READ_BUF_OVERRUN	-8	Empfangspuffer übergelaufen

Bei den . . .OVERRUN-Fehlern wird die richtige Anzahl der im Empfangspuffer stehenden Messages zurückgegeben. Die Fehlermeldungen werden nicht gelöscht!

9 Identifier setzen (Softwarefilterung)

Kann über die Hardware Acceptance-Filter keine Filterung der Identifier durchgeführt werden, so können empfangene CAN-Pakete auf IR-Ebene mit bis zu 16 Identifiern verglichen werden. Nur Pakete, die mit einem Identifier übereinstimmen, werden weitergereicht, alle anderen Pakete werden sofort verworfen. Pakete mit dem Identifier `0x00000000` werden nicht gefiltert, sondern immer weitergereicht. Wird keine Softwarefilterung gesetzt, so werden alle Pakete empfangen.

Die Softwarefilterung greift erst nach einer evt. Filterung über die Acceptance-Maske.

```
WORD can_add_accept_id( WORD can_nr, ULONG identifier ) ;

SPC  CAN_ADD_ACCEPT_ID ENTRY ( FIXED(15), FIXED(31) )
                               RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
`E_OK`	0	kein Fehler
`E_NO_CAN`	-1	CAN-Baustein nicht verfügbar
`E_NO_INIT`	-5	keine Initialisierung durchgeführt
`E_OFF_BUS`	-6	Off-Bus
`E_NO_FREE_ACCEPT`	-29	Kein Platz für Identifier

Wird `E_NO_FREE_ACCEPT` als Rückgabewert geliefert, so konnte der entsprechende Identifier nicht mehr in die Liste eingetragen werden und findet auch keine Berücksichtigung. Bei einem `CAN_INIT` werden alle Einträge gelöscht.

10 Identifier löschen (Softwarefilterung)

Mit dieser Funktion können Identifier wieder gelöscht werden. Wird für `identifier` der Wert `0xFFFFFFFF` übergeben, werden alle Identifier gelöscht, d.h. es werden alle Pakete empfangen.

```
WORD can_del_accept_id( WORD can_nr, ULONG identifier ) ;

SPC  CAN_DEL_ACCEPT_ID ENTRY ( FIXED(15), FIXED(31) )
                               RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
E_OK	0	kein Fehler
E_NO_CAN	-1	CAN-Baustein nicht verfügbar
E_NO_INIT	-5	keine Initialisierung durchgeführt
E_OFF_BUS	-6	Off-Bus
E_ACCEPT_NOT_FOUND	-30	Der Identifier stand nicht in der Liste

Wird `E_ACCEPT_NOT_FOUND` als Rückgabewert geliefert, so konnte der entsprechende Identifier nicht in der Liste gefunden werden. Diese Fehlermeldung hat nur informativen Charakter. Bei einem `CAN_INIT` werden alle Einträge gelöscht.

11 RTR-Answer Buffer einrichten

Bei den CAN-Controllern der PowerPC Familie besteht die Notwendigkeit RTR-Answer Buffer einzurichten, sonst wird auf RTR-Telegramme nicht geantwortet.

Die Funktion `can_set_rtr_answer` hat die gleichen Aufrufparamter wie ein `can_write`, der Unterschied besteht darin, dass das Telegramm nicht sofort verschickt wird, sondern erst als Reaktion auf ein empfangenes Telegramm mit gesetztem RTR-Bit, das den gleichen Identifier enthält.

Die Anzahl der möglichen RTR-Buffer ist durch die Hardware begrenzt und kann mit der Funktion `can_get_feature` ermittelt werden.

Mit `DATA_LENGTH` von Null wird ein bestehender Eintrag eines RTR-Buffers für die angegebene ID gelöscht. Die Funktion `can_set_rtr_answer` kann für eine ID mehrfach aufgerufen werden, der Inhalt des Puffers wird jeweils überschrieben, so dass immer der letzte Eintrag verschickt wird.

Wurde ein RTR-Buffer verschickt, so erhält man mit einem `can_read` das verschickte Telegramm mit gesetztem RTR-Bit. Dies kann genutzt werden, um z.B. neue Antwortdaten für ein RTR-Telegramm zu hinterlegen.

```
WORD can_set_rtr_answer( WORD can_nr, CAN_MESSAGE *m ) ;

SPC  CAN_SET_RTR_ANSWER ENTRY ( FIXED(15), CAN_MESSAGE IDENT )
                                RETURNS( FIXED(15) ) GLOBAL ;
```

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
`E_OK`	0	kein Fehler
`E_NO_CAN`	-1	CAN-Baustein nicht verfügbar
`E_NO_INIT`	-5	keine Initialisierung durchgeführt
`E_OFF_BUS`	-6	Off-Bus
`E_RTR_OVERFLOW`	-32	Liste voll (kein Eintrag erfolgt)
`E_RTR_NFOUND`	-33	Nicht gefunden (beim Löschen)

Wird `E_RTR_NFOUND` beim Löschen als Rückgabewert geliefert, so konnte der entsprechende Identifier nicht in der Liste gefunden werden. Bei einem `CAN_INIT` werden alle Einträge gelöscht.

12 Feature auslesen

Mit dieser Funktion kann der Bausteintyp und, falls vorhanden, die Anzahl der RTR-Answer Buffer ausgelesen werden.

```
WORD can_get_feature( WORD can_nr, WORD* typ, WORD* anz_rtr ) ;

SPC  CAN_GET_FEATURE( FIXED(15), FIXED(15) IDENT, FIXED(15) IDENT)
                                RETURNS( FIXED(15) ) GLOBAL ;
```

Die Variable `typ` gibt den Bausteintyp wieder:

Typ	Baustein
1	SAJ1000
2	MCP2510
3	MPC5200 / MPC5125
4	MPC555
5	MPC5554
6	MPC5674 / MPC5777

Der Variablen `anz_rtr` können Sie die in der Hardware verfügbare Anzahl der RTR-Answer Buffer entnehmen. Ein Rückgabewert von `0` zeigt an, dass diese Hardware RTR-Answer Buffer nicht unterstützt.

Der Rückgabewert gibt eine evt. Fehlernummer zurück:

Name	Wert	Bedeutung
E_OK	0	kein Fehler
E_NO_CAN	-1	CAN-Baustein nicht verfügbar
E_NO_INIT	-5	keine Initialisierung durchgeführt
E_NIMPL	-31	Funktion nicht implementier

Bei einer Fehlermeldung haben die Variablen `typ` und `anz_rtr` keine Bedeutung.

13 Alle Fehlermeldungen im Überblick

In der folgenden Tabelle sind alle Fehlermeldungen des CAN-Bustreibers beschrieben:

Name	Wert	Bedeutung
E_OK	0	Kein Fehler
E_NO_CAN	-1	CAN-Baustein nicht verfügbar
E_ERROR_LEVEL	-2	Interner Warning Level erreicht
E_WRONG_PHYS	-3	ungültiges Interface gewählt
E_WRONG_READ_BUFFER	-4	Empfangspuffergrösse ungültig
E_NO_INIT	-5	keine Initialisierung
E_OFF_BUS	-6	CAN nicht am Bus
E_OVERRUN	-7	Overrun aufgetreten
E_READ_BUF_OVERRUN	-8	Empfangspuffer übergelaufen
E_RESET	-9	Reset, Read-Puffer gelöscht
E_IDENTIFIER	-10	Identifier unzulässig
E_LENGTH	-11	Länge unzulässig
E_WRONG_COMMAND	-12	Unbekannter Befehl
E_RTR	-13	RTR unzulässig
E_NO_MEMORY	-14	kein Speicher für Empfangspuffer
E_TIMEOUT	-15	Timeout aufgetreten
E_FRAME	-16	Frame Format unzulässig
E_SAMP	-17	SAMP unzulässig
E_BAUDRATE	-18	Baudrate unzulässig
E_PHASE_SEG1	-19	Phase_Seg1 unzulässig
E_PHASE_SEG2	-20	Phase_Seg2 unzulässig
E_CAN_CLOCK	-21	Clock-Wert unzulässig
E_JUMP_WIDTH	-22	SJW unzulässig
E_RESET_IMPOSS	-23	Reset nicht möglich
E_SET_ACCEPT	-24	Werte für Controller unzulässig
E_ETH_MASK	-25	ID-Maske > xxxxxx00
E_DUP_INIT	-26	mehrfacher Init nicht zulässig
E_WRONG_ETH_BUFFER	-27	Ethernetpuffergrösse unzulässig
E_NO_SEMA	-28	Semaphore nicht erhalten
E_NO_FREE_ACCEPT	-29	Kein Platz für Identifier
E_ACCEPT_NOT_FOUND	-30	Identifier zum Löschen nicht gefunden
E_NIMPL	-31	Funktion nicht implementiert
E_RTR_OVERFLOW	-32	RTR-Answer Buffer voll
E_RTR_NFOUND	-33	Indentifier nicht gefunden RTR-Answ

14 Ethernet über CAN

Es besteht die Möglichkeit, über den CAN-Bus Ethernet-Pakete zu verschicken. Der CAN-Bus wird dabei als Hardwarelayer genutzt, ebenso wie z.B. beim SLIP das serielle Kabel. Ob der CAN-Treiber Unterstützung für CIP (CAN-IP) bietet, sehen Sie beim Aufruf von `CAN_Version`: Wird `Eth-Treiber` *x.x* ausgegeben, so ist die Unterstützung integriert.

Maximal können Sie 256 CAN-Teilnehmer in ein Ethernet-Segment einbinden. Dazu muß jeder Teilnehmer einen eindeutigen Identifier haben, mit dem die Pakete verschickt werden. Die Empfänger setzen die Ethernet-Pakete wieder zusammen und reichen sie an den TCP/IP-Stack weiter, der anhand der IP-Adresse das Paket bearbeitet oder verwirft. Der maximale Datendurchsatz liegt auf einem MC68332 (16MHz) und 1 MBit Baudrate über den CAN-Bus bei ca. 15 KByte. Wenn der CAN-Treiber auf einer höheren Priorität als der TCP/IP-Stack läuft, bleibt die Echtzeitfähigkeit des CAN-Bus erhalten, d.h. normale Pakete werden in den Strom der Ethernet-Pakete eingefügt. Über die Wahl der Identifier für die Ethernet-Paket kann die Wichtigkeit auf dem CAN-Bus festgelegt werden.

14.1 Initialisierung

Um Ethernet über den CAN-Bus fahren zu können, muß zuerst der Netzwerkstack (`NETIO`, siehe RT-LAN Handbuch) gestartet werden. Nun muß der CAN-Treiber mit dem Aufruf von `can_init` gestartet werden. Dann darf die Prozedur `can_init_eth` aufgerufen werden. Sie darf nur einmal je Kanal aufgerufen werden, d.h. eine Umparametrierung im laufenden Betrieb ist nicht möglich.

```
WORD can_init_eth( WORD can_nr, CAN_ETHERNET *peth )

SPC  CAN_INIT_ETH ENTRY ( FIXED(15), CAN_ETHERNET IDENT )
                         RETURNS( FIXED(15) ) GLOBAL ;
```

Die Struktur `CAN_ETHERNET` hat folgenden Aufbau:

```
typedef struct CAN_ETHERNET
{
   ULONG   eth_address    ;
   ULONG   id_own_address ;
   ULONG   id_mask        ;
   ULONG   id_address     ;
   IOqueue ldn            ;
   WORD    frame_format   ;
   WORD    anz            ;
}
     CAN_ETHERNET ;
```

```
TYPE CAN_ETHERNET STRUCT
[
   eth_address    FIXED(31) ,
   id_own_address FIXED(31) ,
   id_mask        FIXED(31) ,
   id_address     FIXED(31) ,
   ldn            FIXED(15) ,
   frame_format   FIXED(15) ,
   anz            FIXED(15)
]
```

Die Parameter haben folgende Bedeutung:

`eth_address:` Es wird die Basis Ethernet-Adresse für die Kommunikation über den CAN-Bus festgelegt. Die tatsächliche Adresse wird durch das „odern“ mit der `id_own_address` ermittelt. Die `eth_address` muß also bei allen Teilnehmern eines Segmentes gleich sein. Eine mögliche Adresse wäre z.B. 192.168.20.0. Diese Adresse ergibt umgerechnet `0xC0A81400` oder dezimal 3232240640. Das untere Byte der Adresse muß 0 sein, damit sich eindeutige Adressen ergeben.

`id_own_address:` `id_own_address` muß für jeden Teilnehmer eindeutig sein. Es dürfen keine doppelten Adressen vergeben werden. Je nach Größe von `id_mask` sind Werte von 0 bis max. 255 zulässig.

`id_mask:` Es wird die Anzahl der für die Ethernet-Kommunikation benötigten Identifier festgelegt. Gleichzeitig wird der interne Speicherbedarf des CAN-Treibers beeinflußt. Je Teilnehmer werden 10 Byte Speicher benötigt. In der Maske dürfen max. die unteren 8 Bits auf 0 stehen, dann würden 256 Identifier belegt und intern 2,5 KByte benötigt. Wird z.B. in `id_mask` der Wert von `0xFFFFFFF0` eingetragen, so können nur 16 Teilnehmer am Ethernet über CAN-Bus teilnehmen, intern würden aber auch nur 160 Byte benötigt. Bei der obigen Maske darf `id_own_address` bis max. 15 gehen. Für jeden Teilnehmer, der ein Ethernet-Telegramm überträgt, wird auf der Empfängerseite noch ein CE mit 1600 Byte angefordert.

`id_address:` `id_address` legt den Basis-Identifier für die Ethernet-Kommunikation über den CAN-Bus fest. Der reale Identifier ergibt sich wie bei der `eth_address` durch „aufodern“ mit `id_own_address`. Daher müssen auch hier die unteren Bits zu `0` gesetzt werden.

`ldn:` Wird `ldn` auf 0 gesetzt, kommen die Default-Werte (Kanal 1 = 107, Kanal 2 =106 usw.) zum Einsatz. Von diesen Werten sollte nicht ohne Not abgewichen werden.

`frame_format:` Wie bei den normalen CAN-Telegrammen wird festgelegt, ob lange oder kurze Identifier Verwendung finden sollen. Das Format muß natürlich entsprechend der `id_address` festgelegt werden.

`anz:` Hier kann die Anzahl der Empfangspuffer für die Interruptebene festgelegt werden. Dort werden die CAN-Telegramme zwischengepuffert, wenn die Grundebene nicht ans laufen kommt. Maximal dürfen 256 Empfangspuffer angelegt werden. Je Puffer werden min. 18 Byte belegt.

Beispiel: Es sollen max. 32 Teilnehmer am Ethernet über CAN teilnehmen. Folgende Parameter können gewählt werden:

```
eth_address    = 0xC0A81400 ;// Adresse = 192.168.20.0
id_own_address = 0x00000001 ;// eigene Adresse; muß eindeutig sein
id_mask        = 0xFFFFFFE0 ;// für 32 Teilnehmer
id_address     = 0x01230000 ;// CAN-Identifier
ldn            = 0          ;// default Wert nehmen
frame_format   = FRAME_LONG ;// lange Identifier
anz            = 240        ;// da paßt ein Eth-Telegramm rein
```

Name	Wert	Bedeutung
E_OK	0	Kein Fehler
E_NO_CAN	-1	CAN-Baustein nicht verfügbar
E_ETH_MASK	-25	ID-Maske > xxxxxx00
E_DUP_INIT	-26	mehrfacher Init nicht zulässig
E_WRONG_ETH_BUFFER	-27	Ethernetpuffergrösse unzulässig