CAN-Board

Um die ersten Schritte in der CAN-Welt machen zu können, sind mindestens zwei, besser drei Busteilnehmer nötig. Sie haben also entweder

- sowieso schon Erfahrung mit CAN und andere Busteilnehmer greifbar. Sie kennen diese Geräte einigermaßen gut.
- Mindestens zwei TCAN-Module gekauft, einen Adapter für das Plug & Play Lab und eine andere Hardware-Plattform für das zweite TCAN-Modul aufbereitet.
- das TCAN-Entwicklungspaket gekauft, worin als weitere Busteilnehmer zwei Boards mit dem CAN-SLIO-Baustein von Philips enthalten sind.

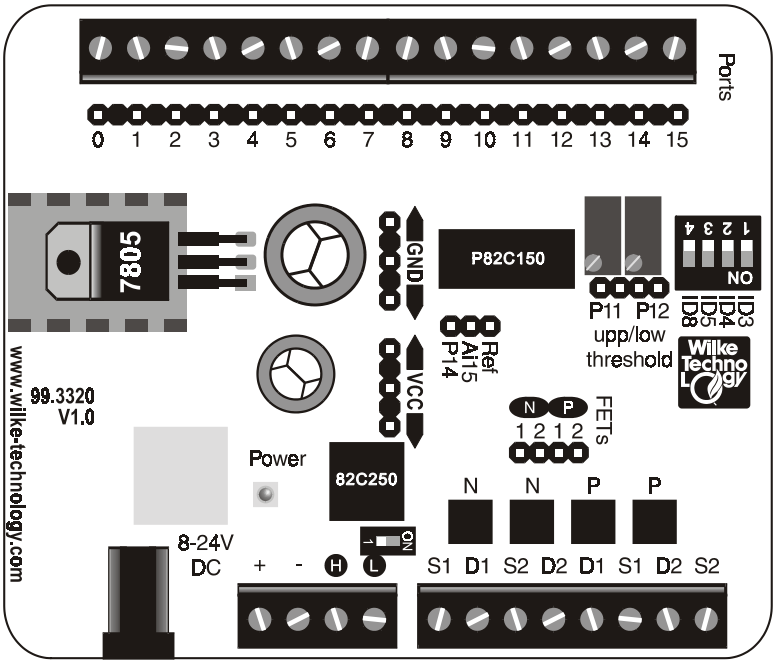
In diesem Abschnitt finden Sie:

- CAN-SLIO-Board
- CAN-SLIO-Chip
- Identifier des SLIO
- Bitrate automatisch erkennen
- SLIO-Nachrichtenformat
- Statusbyte - Data-Byte 1
- SLIOs auf dem Bus finden
- Einige Besonderheiten für Interressierte
- Remote Frames
- Bit-Timing
- Oszillator und Kalibrierung
- Initialisierung
- Sign-On-Message
- Registerübersicht
- SLIO digitale Ein- und Ausgänge
- SLIO-Analog-Ausgänge
- Analog-Konfiguration
- Starten der A/D-Wandlung
- Zwei SLIOs an einem Bus

CAN-SLIO-Board

Das CAN-SLIO-Board ist eine I/O-Einheit, die über den CAN-Bus entfernt von der Steuereinheit seriell angebundenen ist. Das Kernstück ist der CAN-SLIO-Chip (SLIO=Serial-Linked-I/O), der in den folgenden Abschnitten ausführlich beschrieben ist. Das Board enthält:

- ein Netzteil mit Spannungsregler
- den CAN-Treiber-Chip
- einen zuschaltbaren Abschlußwiderstand für den CAN-Bus
- einen DIP-Schalter, um den Identifier einzustellen
- Zwei Potis, um 2 analoge Spannungen einzustellen
- eine einfache Filterschaltung für die quasi-analogen Ausgänge
- 2 N-FETs und 2 P-FETs zur beliebigen Verwendung
- Schraubklemmen für 16 digitale I/O
- Schraubklemmen für den CAN-Bus
- Schraubklemmen für die Stromversorgung, falls die Buchse nicht verwendet wird



Technische Daten des CAN-SLIO-Boards:

Außenmaße / Gewicht:	ca.100 x 85 x 27 mm / ca. 75g
Stromversorgung	8V...24V / ca. 22 mA (unbelastete I/O)
Belastbarkeit der I/O-Pins	±4mA, alle Pins zusammen < 200mW
Reset	Power-ON-Reset auf dem Board durch R-C-Glied
Temperaturbereich	-40...+85°C
A/D-Eingang	Auflösung 6...7-Bit, bis zu 6 Kanäle über Analogschalter
D/A-Ausgänge	2 DPM (distributed pulse modulated) Auflösung 10-Bit Wiederholfrequenz 1024 Bitzeiten

Um das Board in Betrieb zu nehmen, schließen Sie es an einen CAN-Bus mit einem quarzkontrollierten Teilnehmer an (Schraubanschlüsse ‚H' und ‚L'). Ein solcher Teilnehmer ist z.B. ein BASIC-Tiger®-CAN-Modul, welches auf dem CAN-Adapter des Plug & Play Labs Platz findet. Wenn sich das SLIO-Board am physikalischen Ende des Busses befindet, schlten Sie den DIP-Schalter neben der Schraubklemme auf ‚ON', um den Abschlußwiderstand zuzuschalten. Wenn sich das Board nicht am Bus-Ende befindet, schalten Sie den DIP-Schalter aus.

Verdrahten Sie weitere Hardware, die an I/O-Pins angeschlossen werden soll.

Stellen Sie an dem 4er-DIP-Schalter die gewünschten ID-Bits ein. Alle Busteilnehmer sollten einen eindeutigen Identifier haben, der nicht noch einmal im Bussystem vorkommt.

Laden Sie in das Modul TCAN-4/4 eines der Beispielprogramme. Das grundlegendste Beispielprogramm ist ‚SLIO_FIND1.TIG', welches das Finden des SLIO-Chips bewerkstelligt. Damit verbunden ist die automatische Bitratenerkennung und ständige Re-synchronisation des SLIO-Chips.

CAN-SLIO-Chip

Der CAN-SLIO-Chip P82C150 ist ein seriell angebundener I/O-Chip (SLIO=Serial-Linked-I/O), dessen Register über den CAN-Bus beschreibbar sind. Der SLIO-Chip unterstützt die Protokol-Spezifikationen 2.0A und 2.0B (passive). Abstriche bei dem Bit-Timing bestehen aufgrund der automatischen Bitraten-Erkennung, sofern diese genutzt wird. In einem System mit P82C150-Knoten mit automatischer Bitraten-Erkennung muß sich mindestens ein aktiver CAN-Knoten mit Quarz befinden.

Auf dem CAN-SLIO-Board befindet sich ein SLIO-Chip, der die Bitrate selbst erkennen muß. Dies ist möglich im Bereich von Bitzeiten von 8µsec bis ca. 50µsec. Die folgenden Beispiele sind auf eine Bitzeit von ca. 22µsec eingestellt.

Identifier des SLIO

Der CAN-SLIO verarbeitet 11-Bit-Identifier. 29-Bit-Identifier werden ignoriert. 4 Bits der Empfangsmaske werden nach dem Reset von den Portpins P0...P3 eingelesen. Diese Portpins sind danach als I/O-Pins verwendbar. Somit lassen sich bis zu 16 SLIO-Chips an einem Bus betreiben (bei Verwendung von externem Clock reduziert sich die Zahl auf 8). Jeder SLIO-Chip verwendet 2 Identifier, die sich im untersten Bit unterscheiden. Die höhere Priorität ist für Nachrichtenempfang reserviert (Bit ID.0=0).

ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	ID.2	ID.1	ID.0
0	1	P3	1	0	P2	P1	P0	1	0	DIR

DIR	0: Nachrichten, die vom Host an den SLIO-CHIP gesendet werden (Schreibregister, Konfiguration). 1: Nachrichten, die vom SLIO-CHIP an den Host gesendet werden (Leseregister, RTR).
P0, P1, P2, P3	Programmierbare ID-Bits, die nach dem Reset von den Pins P0...P3 gelesen werden.

Bitrate automatisch erkennen

Nach dem Einschalten oder einem Reset befindet der SLIO-Chip sich im Sleep-Modus. Der interne Oszillator ist gestoppt und alle Output-Treiber sind de-aktiviert. Beim ersten dominanten Bit auf dem Bus geht er in den ‚differential mode'. Um den Chip auf dem CAN-Bus verfügbar zu machen, muß zunächst die Bitrate erkannt werden. Dazu benötigt der SLIO-Chip ein bestimmtes Bitmuster auf dem Bus. Eine bestimmte Nachricht mit festgelegtem Identifier und festgelegten Datenbytes erzeugt dieses Bitmuster. Die gleiche Nachricht wird später verwendet, um den SLIO-Chip zu re-synchronisieren und damit aktiv zu halten. Spätestens nach 8000 Bitzeiten muß erneut eine re-synchronisation erfolgen, sonst geht der SLIO in wieder in einen

inaktiven Zusatand über. Diese CAN-Nachricht wird in Tiger-BASIC® so zusammengesetzt:

```
calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst
```


Der Identifier ist AAh und die Datenbytes sind AAh und 4. Da der Identifier linksbündig in einem WORD bzw. 2 Bytes stehen muß, werden die 11 Bits um 5 nach links geschoben. In Tiger-BASIC® steht in einem WORD jedoch das low-Byte zuerst, in der Message wird aber das high-Byte zuerst benötigt. Die Funktion NTOS$ dreht die Bytes beim Einbauen in den String durch die negativ angegebene Anzahl Bytes (-2). Diese Nachricht dient der Synchronisation, sie wird nicht von dem SLIO empfangen und sollte auch bei anderen Busteilnehmern nicht durch den Filter gelangen.

Die gewählte Bitrate sollte möglichst 10 Zeitsegmente enthalten oder dem nahe kommen, um die Bitratenerkennung zu erleichtern. Im Beispiel ist diese Bedingung erfüllt. Die Bitzeit sollte zwischen 8µsec und 50µsec liegen.

In den folgenden Beispielprogrammen ist der Filter der CAN-Hardware (Access-Code und Access-Mask) so eingestellt, daß möglicht nur die SLIO-Nachricht durchkommt. In einem System mit viel Datenverkehr würde dieses allererste einfache Beispielprogramm jede durchkommende Nachricht für eine SLIO-Nachricht halten.

Wenn die Bitrate erkannt ist, dann meldet sich der SLIO-Chip mit seiner ‚Sign-On'-Nachricht. Diese hat das gleiche Format wie alle SLIO-Nachrichten und meldet den Inhalt des Daten-Input-Registers.

SLIO-Nachrichtenformat

Im Identifier jeder SLIO-Nachricht ist das Richtungsbit ‚DIR' auf 1 gesetzt, die Variablen Bits sind von den Ports P0...P3 eingelesen worden. Sowohl gesendete als auch empfangene CAN-Nachrichten enthalten 3 Datenbytes. Das erste Byte enthält die Registeradresse sowie Statusinformationen, die beiden anderen Bytes den Inhalt des angegebenen Registers. Nach jeder erfolgreich versendeten Nachricht verzögert der SLIO-Chip seine nächste eventuell anstehende Nachricht um 3 Bitzeiten, um niedriger priorisierten Knoten eine Sendemöglichkeit einzuräumen. Dies ist wichtig im Falle von Wackelkontakten an einem Pin, der Flankengetriggert eine CAN-Nachricht auslöst.

Die Statusinformationen im ersten Byte werden bei empfangenen Nachrichten ignoriert. Jede empfangene Nachricht wird durch eine neue Nachricht mit dem Inhalt des adressierten Registers bestätigt. (Ausnahme: siehe Analog-Konfiguration). Wenn der SLIO-Chip eine CAN-Nachricht empfangen hat, dann sendet er als Antwort eine Nachricht, die 4 Statusbits, die zuvor empfangene Registeradresse sowie den Inhalt

des Registers enthält. (Ausnahmen siehe A/D-Konfiguration). So werden Leseregister gelesen, und bei Schreibregistern wird der Schreibvorgang bestätigt.

Statusbyte - Data-Byte 1

Status				Register-Adresse			
RSTD	EW	BM1	BM0	A3	A2	A1	A0

RSTD — 1: Sign-On-Nachricht
0: andere Nachrichten

EW — 1: ‚error warning limit' (32) ist erreicht. Bit wird gesetzt, wenn der Receive Error Counter oder der Transmit Error Counter den Wert 32 überschritten hat. Immer 1 in der ‚Sign-On'-Nachricht.

BM0, BM1 — Busmode Statusbits (siehe unten).

A0...A3 — Registeradresse. Bestimmt, welches interne SLIO-Register gelesen oder beschrieben wird.

Der Busmodus wird in den Statusbits BM0 und BM1 angezeigt:

Bus Mode	Bits		Reception Level		Transmission	
	BM1	BM0	recessive	dominant	Tx1	Tx0
0=differential	0	0	Rx0>Rx1	Rx0<Rx1	enabled	enabled
1=one wire Rx1	0	1	Rx1<REF	Rx1>REF	enabled	enabled
2=one wire Rx0	1	0	Rx0>REF	Rx0<REF	disabled	enabled
3=sleep	1	1	Rx0>REF and Rx1<REF	Rx0<REF and Rx1>REF	disabled	enabled

SLIOs auf dem Bus finden

Das Beispiel SLIO_FIND1.TIG geht davon aus, daß nur ein SLIO-Board angeschlossen ist. Es sollen erste Erkenntnisse über den SLIO gewonnen werden. Jede ankommende Nachricht wird für eine SLIO-Nachricht gehalten. Die Bytes werden in HEX so angezeigt, wie sie ankommen, so daß die oben erwähnten Bits leicht nachprüfbar sind. Die Adresse der SLIO ist an dem DIP-Schalter einstellbar. Nach dem Verstellen des DIP-Schalters muß das Board aus- und wieder eingeschaltet werden, denn der SLIO liest die ID-Bits nur beim Power-On bzw. nach einem Reset ein. Die Sign-On-Nachricht sieht z.B. so aus (alle DIP-Schalter aus):

03 50 A0 E0 80 00

03: Das Frame-Info-Byte zeigt an, daß es sich um eine Standard-Frame-Nachricht mit 3 datenbytes handelt.

50 A0: der Identifier enthält nur die festen SLIO-ID-Bits, die variablen Bits sind ‚0', da alle DIP-Schalter aus sind. Die 11 Bits sind rechtsbündig in den beiden Bytes enthalten: 0101 0000 101 – 0 0000 0000.

C0: RSTD=1 -> Sign-On-Nachricht, EW ist in Sign-On immer 1, BM=10 -> Busmodus ist ‚Sleep-Mode'. Die Registeradresse ist 0, es ist also das Data-Input-Register.

80 00: Inhalt des Data-Input-Registers. Bei offenen Pins ist der Inhalt eher zufällig.

Verstellen Sie den DIP-Schalter und Schalten Sie das SLIO-Board aus und wieder ein. Lassen Sie dann SLIO_FIND1 nochmal laufen.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_FIND1.TIG
' FIndet SLIO-Baustein(e) oder meldet Fehler.
' Verbinde CAN-SLIO-Board(s) mit unterschiedlichen Adressen
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210           ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung dieses SLIO

BYTE slio_stat
WORD slio1_id, calib_id
LONG ac_code, ac_mask
STRING id$(4), calib$(5)
STRING slio1_dout$(6), slio1_doe$(6)    ' data out, data out enable

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever
  WORD fi
  STRING id1$, c$

  install_device #LCD, "LCD1.TDD"

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
     27 1F FF FF &                ' access mask
     10 45 &                      ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                      ' single filter mode, outctrl

  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
'
  print #LCD, "trying to find SLIO";
  slio_stat = NOT_READY
  while slio_stat = NOT_READY     ' pruefe SLIO-Status
    put #CAN, calib$              ' Kalibrierbotschaft auf den Bus
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 2, fi
    if fi > 5 then                ' wenn min 1 Botschaft da ist
      print #LCD, "<1bh>A<0><1><0f0h>"; ' Cursor setzen
```

```
        get #CAN, 1, c$
        using "UH<2><2>   0 0 0 0 2" ' und HEX anzeigen
        print_using #LCD, asc(c$);" ";
        get #CAN, #0, #UFCI_IBU_FILL, 2, fi ' mehr?
      endwhile
      print #LCD, "<1bh>A<0><2><0f0h>SLIO found";
      slio_stat = READY
    endif
  endwhile

  for ever = 0 to 0 step 0        ' Endlosschleife
  next
END
```


Für die später folgenden Beispiele ist ein etwas komplizierteres Unterprogramm ‚find_slios' geschrieben worden, welches weitere Gegebenheiten berücksichtigt:

Es können mehrer SLIOs am Bus sein.

Es könnten noch Busteilnehmer vorhanden sein, die keine SLIO sind.

Es könnte gar keine SLIO da sein, aber das Programm soll weiterlaufen.

Die SLIOs brauchen regelmäßig Kalibrier-Bitmuster auf dem CAN-Bus, sonst gehen sie wieder in den Sleep-Modus und die Ausgänge werden inaktiv.

Das Unterprogramm wartet eine Sekunde, um allen eventuell vorhandenen SLIOs die Möglichkeit zu geben, Ihre Sign-On-Nachricht zu schicken, und von der Anwendung auch berücksichtigt zu werden. Andererseits wird nicht länger gewartet, denn es könnte sein, daß sich keine SLIO meldet. Dieses Beispiel meldet diese Tastsache und beendet das Main-Programm.

Die Empfangenen Nachrichten werden unersucht, ob sie auch von einer SLIO stammen. Kein anderer Busteilnehmer darf eine der 16 möglichen SLIO-IDs haben. Andere als SLIO-Nachrichten werden in der Erkennungsphase nur aus dem Puffer gelesen und verworfen.

Die Task ‚keep_alive' sendet die Kalibrier-Nachrichten, um die SLIOs aufzuwecken und sie anschließend aktiv zu halten. Spätestens nach 8000 Bitzeiten soll eine Kalibrier-Nachricht auf dem Bus erscheinen. Kein Teilnehmer empfängt sie, nur ein für die Re-synchronisation der SLIOs notwendiges Bitmuster wird dadurch erzeugt. Während der Erkennungsphase wird die Kalibrier-Nachricht etwas öfter gesendet.

Die Task ‚keep_alive' sendet auch Nachrichten aus sofern dies nötig ist. In diesem Beispiel wird es nicht nötig, da nur die Erkennung demonstriert werden soll. Die anderen Beispiele benutzen jedoch die Ausgabe der Task ‚keep_alive' über die Variable ‚can_out$'.

Wenn Sie dieses Beispiel jetzt einzeln verstehen, können Sie in den folgenden Beispielen dieses Unterprogramm als ‚Black-Box' betrachten.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_FIND2.TIG
' Findet SLIO-Baustein(e) oder meldet Fehler.
' Verbinde CAN-SLIO-Board(s) mit unterschiedlichen Adressen
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen
#INCLUDE CANSLIO.INC               ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50               ' warte ca. 2200 * 22usec        wait
#define SLIOSLOT 10000             ' msec Zeit, sich 'bereit' zu melden

'                  3  210           ID-Bitpositionen
#define ID_SLIO1 00000000000b      ' ID-Bit-Einstellung dieses SLIO

                                   ' globale Variable
BYTE no_of_slios                   ' Zaehler SLIOs im System
WORD slio1_id, calib_id            ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask              ' Access-Code und Access-Mask
LONG slio_equip                    ' ein Bit fuer jede SLIO
LONG alive_wait                    ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                   ' Kalibrierungsbotschaft
STRING can_out$(13)                ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' Endlosschleife
  WORD fi                          ' Pufferfuellung
  STRING c$

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                 ' access code
     27 1F FF FF &                 ' access mask
     10 45 &                       ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                       ' single filter mode, outctrl

  call find_slios                  ' stellt Liste aller im System
                                   ' befindlichen SLIOs zusammen
  if slio_equip = 0 then           ' wenn keine SLIO gefunden
    goto no_slio_found             ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif

  for ever = 0 to 0 step 0         ' Endlosschleife
    print #LCD, "<1bh>A<0><3><0f0h>task main running"
  next

no_slio_found:
```

```
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'--------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                    ' Schleifenvariable
  WORD ibu_fill                   ' Eingangspufferfuellung
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  LONG r_id                       ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  slio_equip = 0                  ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst
  can_out$ = ""                   ' muss initialisiert sein

  alive_wait = ALIVETIME/2        ' in der Initphase kuerzer
  run_task keep_alive             ' sendet Kalibrier-Botschaften
                                  ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                     ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
rx_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                  ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                    ' warte auf naechste Botschaft
      endif
'--------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
```

```
      r_id = r_id bitand SLIO_ID_MASK   ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then       ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h
      endif
      r_id = r_id shr 3                 ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id          ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1     ' zaehle die gefundene SLIO
                                        ' ignoriere hier Info in Daten
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
'       using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
'       for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'         print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'       next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
' Wenn 'can_out$' nicht leer ist, wird es gesendet und anschliessend
' wieder auf leer gesetzt.
'--------------------------------------------------------------------
TASK keep_alive
  BYTE ever

  for ever = 0 to 0 step 0             ' Endlosschleife
    put #CAN, calib$                   ' sende Kalibrier-Botschaft
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
    if can_out$ <> "" then
      put #CAN, can_out$               ' Sendebotschaft
      can_out$ = ""                    ' leer = gesendet
    endif
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
  next
END
```

Einige Besonderheiten für Interressierte

Um die nicht ganz unkomplizierte Thematik in der Einführung nicht unnötig zu erschweren, wurden einige Besonderheiten nicht an Ort und Stelle erwähnt.

Remote Frames

Remote-Frames können eingesetzt werden, um entfernt über den CAN-Bus Nachrichten hervorzurufen, sprich Werte abzufragen und Informationen zu erhalten. Obwohl die SLIO auch ohne gesetztes RTR-Bit antwortet, darf das Bit natürlich gesetzt sein. Empfangene Remote Frames müssen den DLC (data length code) jedoch auf 3 gesetzt haben, sonst werden sie ignoriert. Die Antwort enthält immer die Bytes des Daten-Input-Registers.

Bit-Timing

Die nominale Bitzeit des CAN-SLIO-Chips ist unterteilt in 10 Bitsegmente. Das Synchronisations-Segment und das Propagation-Time-Segement leiten das Bit ein. Es folgt mit 4 Segmenten Phase 1, an deren Ende das Bit abgetastet wird. Phase 2 ist ebenfalls 4 Segment lang. Diese Timingstruktur ist nicht veränderbar. Der Quarz-kontrollierte Host sollte auch so eingestellt werden, daß die Bitzeit in 10 Segmente zerlegt wird.

1 Bitzeit									
BT1	BT2	BT3	BT4	BT5	BT6	BT7	BT8	BT9	BT10
Sync	Prop	phase_1				phase_2			

Oszillator und Kalibrierung

Der SLIO-Chip hat einen internen R-C-Oszillator, der automatisch durch die CAN-Nachrichten auf dem Bus kalibriert wird. Während des Starts wird jede Nachricht verwendet, um die Bitzeit einzustellen. Zur genauen Kalibrierung ist ein bestimmtes Bitmuster auf dem Bus notwendig. Beispiel einer geeigneten Kalibrier-Nachricht (| = Stuffbit, die wichtigen Bits sind unterstrichen):

SOF	**arbitration**	**control field**	**data byte 1**	**data byte 2**	**CRC field**
0	000101010100	000|010	10101010	0000|0100	000|01011100000|0

Identifier ist AAh, zwei Datenbytes mit den Werten AAh, 4 sind in der Nachricht enthalten.

Initialisierung

In der Resetphase (RST=high) sind alle Ausgänge des CAN_SLIO hochohmig. An den Pins P0...P3 werden die 4 ID-Bits eingelesen. Die anderen ID-Bits sind unveränderbar festgelegt. Gemäß der CAN-Definition dürfen nicht zwei Knoten den gleichen Identifier verwenden. Ein CAN-SLIO hat eine der 16 möglichen Bitkombinationen.

Reset-Zustand:

Status-Bits	Identifier-Bits
RSTD = 1	ID.3 gleich P0
EW = 1	ID.4 gleich P1
BM1 = 0	ID.5 gleich P2
BM0 = 0	ID.8 gleich P3

Der SLIO-Chip muss mindestens 3 Nachrichten auf dem Bus empfangen haben, bevor der Bus-Mode gewechselt wird. Die erste Botschaft wird benutzt, um die Bitzeit zu messen, deswegen sollte die Nachricht eine Sequenz ‚010101' enthalten. Die 2. und die 3. Nachricht wird trotz korrektem Empfang nicht mit einem ACK bestätigt. Nach 3 korrekt empfangenen Nachrichten sendet der SLIO-Chip seine ‚sign-on'-Nachricht.

Der CAN-SLIO-Chip wertet auch Nachrichten als korrekt empfangen, wenn diesen ein Error-Passive-Frame auf Grund des fehlenden ACK folgt. Diese Situation ist gegeben, wenn ein Host mit einem oder mehreren CAN-SLIOs zusammenarbeitet und die SLIOs noch nicht calibriert sind.

Sign-On-Message

Diese spezielle Botschaft wird einmal von jedem CAN-SLIO versandt, nachdem der Chip kalibriert wurde. Damit wird die Betriebsbereitschaft des Knotens angezeigt.

Die Sign-On-Nachricht meldet den Inhalt des data-Input-Registers und kann von anderen Nachrichten durch das gesetzte Bit RSTD unterschieden werden:

RSTD = 1: Sign-On-Nachricht
RSDT = 0: andere Nachrichten

Anmerkung: In der Sign-On-Nachricht ist das EW-Bit gesetzt. Trotzdem wird der Status und die Error-Zähler zurückgesetzt auf 0.

Registerübersicht

Register Adresse	Registername	Funktion
0	Data Input	Enthält die Input-Pegel der Pins P0 bis P15
1	Event Positive Edge	Aktiviert bei gesetztem Bit das Absetzen einer Nachricht bei positiver Flanke an den Pins P0...P15
2	Event Negative Edge	Aktiviert bei gesetztem Bit das Absetzen einer Nachricht bei negativer Flanke an den Pins P0...P15
3	Data Output	Enthält die Output-Pegel der Pins P0 bis P15
4	Output Enable	Aktiviert bei gesetztem Bit die Ausgangstreiber an den Pins P0...P15
5	Analog Configuration	Bits 0...4: ohne Funktion Bits 5,6,7: Position des Analogschalters Bits 8,9,10: Position der Monitoring-Schalter Bit 11: ohne Funktion Bits 12,13,14: Ergebnis der Analog-Komparatoren Bits 15: startet A/D-Wandlung, wenn gesetzt
6	DPM1	Distributed Pulse Modulation. Enthält den 10-Bit-Analogwert in den Bits 6...15. Quasi-Analog-Ausgang 1 am Pin P10
7	DPM2	Distributed Pulse Modulation. Enthält den 10-Bit-Analogwert in den Bits 6...15. Quasi-Analog-Ausgang 2 am Pin P4
8	A/D-Conversion	Enthält das 10-Bit-Ergebnis der A/D-Wandlung in den Bits 6...15

SLIO digitale Ein- und Ausgänge

Der SLIO-Chip stellt 16 Pins zur Verfügung, die als digitale Eingänge oder digitale Ausgänge arbeiten können. Einige Pins haben jedoch die Möglichkeit, andere Funktionen zu übernehmen, z.B. analoge Eingänge, analoge Ausänge sowie das Einlesen der ID-Bits nach dem Reset. Einige Pins sind daher auf dem CAN-SLIO-Board besonders beschaltet und nicht einfach auf die Stiftleiste geführt:

P0...P3	besitzen Pull-up-Widerstände von 47k und Pull-down-Widerstände von 4k7, um mit dem DIP-Schalter die ID-Bits einzustellen.
P4, P10	sind mögliche quasi-analoge Ausgänge und über einfache R-C-Filter geführt.
P15, P16	sind über einen 100k-Widerstand verbunden und an P15 befindet sich ein 3,3nF-Kondensator gegen Masse. Diese Beschaltung ist notwendig, um P15 als Analog-In zu verwenden.

Digitale Eingänge: Nach dem Reset oder Power-On sind alle Pins Eingänge mit ca. 500k Eingangswiderstand. Die Pegelzustände spiegeln sich im ‚Data-In'-Register (Adresse 0) wieder. Das Register ‚Output-Enable' (Adresse 4) hat für jeden Portpin ein Bit, welches auf ‚0' steht und damit den Output-Treiber de-aktiviert.

Digitale Ausgänge: Das Register ‚Output-Enable' (Adresse 4) hat für jeden Portpin ein Bit, welches auf ‚1' den Output-Treiber aktiviert. Das Register muß explizit beschrieben werden, um Ausgangspins zu aktivieren. Das Bitmutser für alle aktivierten Ausgänge wird in das Register ‚Data-Output' (Adresse 3) geschrieben.

Events: CAN-Nachrichten werden automatisch versandt, wenn in den Registern ‚Event Positive Edge' und/oder ‚Event Negative Edge' das Bit für den gewünschten Pin gesetzt ist und die dazu passende Flanke auftritt.

Das folgende Beispielprogramm setzt alle Portpins zu Ausgängen und schaltet sie abwechselnd auf high und low. Um die SLIO(s) zu finden, wird das Unterprogramm ‚**find_slios**' zusammen mit der Task ‚**keep_alive**' verwendet, so wie unter ‚SLIOs auf dem Bus finden' beschrieben. Der Identifier der zuletzt gefundenen SLIO wird verwendet. Die Ports werden zu Ausgängen, indem in dem ‚Output-Enable'-Register alle Bits auf ‚1' gesetzt werden. Dann wird in das ‚Data-Output'-Register abwechselnd 0000 und FFFFh geschrieben. Da der SLIO mit einer Bestätigungs-Nachricht antwortet, wenn er eine Nachricht empfangen hat, gibt die Task ‚**show_slio**' diese SLIO-Nachrichten und den Identifier auf dem LCD in HEX aus. Fremde Nachrichten, dazu zählen automatisch extended Frames, werden von ‚show_slio' verworfen, es wird nur der Empfangspuffer bereinigt.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_HIGH_LOW.TIG
' Blinkt alle Outputs des SLIO 'high-low'
' Verbinde CAN-SLIO-Board, alle DIP-Schalter aus
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen
#INCLUDE CANSLIO.INC               ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50               ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000              ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0                ' SLIO-Status 'nicht bereit'
#define READY 1                    ' SLIO-Status 'bereit'

'                  3  210           ID-Bitpositionen
#define ID_SLIO1 00000000000b      ' ID-Bit-Einstellung dieses SLIO

                                   ' globale Variable
BYTE no_of_slios                   ' Zaehler SLIOs im System
BYTE slio_stat                     ' SLIO status
WORD slio1_id, calib_id            ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask              ' Access-Code und Access-Mask
LONG slio_equip                    ' ein Bit fuer jede SLIO
LONG alive_wait                    ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                   ' Kalibrierungsbotschaft
STRING slio1_dout$(6), slio1_doe$(6)' data out, data out enable
STRING can_out$(13)                ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' Endlosschleife
  WORD fi                          ' Pufferfuellung

  install_device #LCD, "LCD1.TDD"

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                 ' access code
     27 1F FF FF &                 ' access mask
     10 45 &                       ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                       ' single filter mode, outctrl

                                   ' ID dieses SLIO: 0=standard frame
```

```
'  slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO-Identifier
' slio1_id wird letzte gefundene SLIO
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  can_out$ = ""                   ' muss initialisiert sein

  call find_slios                 ' stellt Liste aller im System
                                  ' befindlichen SLIOs zusammen
  if slio_equip = 0 then          ' wenn keine SLIO gefunden
    goto no_slio_found            ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif
  wait_duration 1500
  print #LCD, "<1>";

  run_task show_slio              ' zeigt an, was die SLIO(s) senden

  slio1_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO_OUT_ENABLE
  slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
  can_out$ = slio1_doe$
  wait_duration 100
  for ever = 0 to 0 step 0        ' Endlosschleife ----------
    slio1_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO_DATA_OUT
    slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID high-Byte
    can_out$ = slio1_dout$
    wait_duration 100
    slio1_dout$ = "<0><0><0><3><0><0>" ' SLIO_DATA_OUT   SLIO_DATA_OUT
    slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID einbauen
    can_out$ = slio1_dout$
    wait_duration 100
  next                            ' Endlosschleife ----------

no_slio_found:                    ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                        ' Schleifenvariable
  WORD ibu_fill                       ' Eingangspufferfuellung
  BYTE frameformat, msg_len           ' Frame-Format, Botschaftslaenge
  LONG r_id                           ' Empfangs-ID
  STRING msg$(13), data$(8)           ' Botschaft und Daten

  for ever = 0 to 0 step 0            ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then               ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
```

```
                                    ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then       ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont              ' warte auf naechste Botschaft
        endif
      else                          ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then         ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                ' warte auf naechste Botschaft
      endif                         ' ---------------------------
                                    ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"      ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      print #LCD, "                     <13>DATA:";' loesche Datenanz.
      if msg_len > 0 then           ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie und zeige an
        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
        next
      endif
    endif
  next
END


'---------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'---------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                  ' Schleifenvariable
  WORD ibu_fill                 ' Eingangspufferfuellung
  BYTE frameformat, msg_len     ' Frame-Format, Botschaftslaenge
  LONG r_id                     ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)     ' Botschaft und Daten

  slio_equip = 0                ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5         ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2      ' in der Initphase kuerzer
  run_task keep_alive           ' sendet Kalibrier-Botschaften
                                ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                   ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
```

```
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont                ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont                  ' warte auf naechste Botschaft
      endif
'-------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
                                        ' maskiere DIR-Bit weg
      r_id = r_id bitand SLIO_ID_MASK   ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then       ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h
      endif
      r_id = r_id shr 3                 ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id          ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1     ' zaehle die gefundene SLIO
                                        ' ignoriere hier Info in Daten
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'-------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'-------------------------------------------------------------------
TASK keep_alive
  BYTE ever

  for ever = 0 to 0 step 0             ' Endlosschleife
    put #CAN, calib$                   ' sende Kalibrier-Botschaft
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
```

```
      put #CAN, can_out$                ' Sendebotschaft
      can_out$ = ""                     ' leer = gesendet
    endif
    wait_duration alive_wait            ' warte max 4000 Bitzeiten
  next
END
```

SLIO-Analog-Ausgänge

Der SLIO-Chip stellt zwei quasi-analoge Ausgänge zur Verfügung. Auf dem CAN-SLIO-Board sind diese Ausgänge über einfache R-C-Filter geführt.

Der analoge Wert wird erzeugt, indem über einen festen Zeitraum mehr oder weniger ‚1'-Bits generiert werden (DMP=Distributed Pulse Modulation). Integriert man die ‚1'-Bits erhält man für wenige ‚1'-Bits pro Zeiteinheit eine niedrige Spannung, für viele ‚1'-Bits pro Zeiteinheit eine hohe Spannung (0...5V). Bei der Integration muß ein Kompromiß geschlossen werden zwischen Reaktionsgeschwindigkeit der analogen Spannung und Welligkeit. Die Erzeugung eines Analogwertes ist nach 1024 Bitzeiten abgeschlossen und wird anschließend wiederholt. Grundlage für die Anzahl der ‚1'-Bits ist der Inhalt des Registers ‚DPM1' für Pin P10 oder ‚DMP2' für Pin P4. Im Register befindet sich in einem WORD linksbündig der 10-Bit-Wert für die DPM-Ausgabe.

Die Ausgangstreiber der Pins, die zur Analogausgabe verwendet werden, müssen im Register ‚Output-Enable' aktiviert werden.

Das Beispielprogramm erzeugt auf dem Pin P10 eine Sägezahnkurve, indem aufsteigende Analogwerte ausgegeben werden. Die SLIO(s) werden mit Hilfe des Unterprogrammes ‚**find_slios**' zusammen mit der Task ‚**keep_alive**' gefunden, so wie unter ‚SLIOs auf dem Bus finden' beschrieben. Danach übernimmt die Hauptschleife des Programms die Aufgabe, die SLIOs zu re-kalibrieren, da hier sowieso eine schnelle und konstant sich wiederholende Ausgabe stattfindet. Die Task ‚**show_slio**' zeigt die ankommenden SLIO-Nachrichten und den Identifier auf dem LCD in HEX zur Kontrolle an. Fremde Nachrichten, dazu zählen automatisch extended Frames, werden von ‚show_slio' verworfen, es wird nur der Empfangspuffer bereinigt.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_DPM1.TIG
' erzeugt DPM-Outputs des SLIO (digital->analog)
' Verbinde CAN-SLIO-Board, alle DIP-Schalter aus
' messe am Pin P10
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50              ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000             ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210          ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung dieses SLIO

BYTE no_of_slios                  ' Zaehler SLIOs im System
BYTE slio_stat                    ' SLIO status
WORD slio1_id, calib_id           ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask             ' Access-Code und Access-Mask
LONG slio_equip                   ' ein Bit fuer jede SLIO
LONG alive_wait                   ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                  ' Kalibrierungsbotschaft
STRING slio1_dpm1$(6), slio1_doe$(6)' DPM1 out, data out enable
STRING slio1_dout$
STRING can_out$(13)               ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' Endlosschleife
  WORD fi                         ' Pufferfuellung
  WORD value, tmp                 ' Analogwert
  LONG t                          ' 'keep_alive' Zeit
  STRING tmp$(6)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
     27 1F FF FF &                ' access mask
     10 45 &                      ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                      ' single filter mode, outctrl

                                  ' ID dieses SLIO: 0=standard frame
                                  ' + 2 ID-Bytes um 5 geshiftet
  slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO-Identifier
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  can_out$ = ""                   ' muss initialisiert sein
```

```
  call find_slios                 ' stellt Liste aller im System
                                  ' befindlichen SLIOs zusammen
  if slio_equip = 0 then          ' wenn keine SLIO gefunden
    goto no_slio_found            ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif
  wait_duration 1500
  print #LCD, "<1>";

  run_task show_slio              ' zeigt an, was die SLIO(s) senden

  slio1_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO_OUT_ENABLE
  slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
  can_out$ = slio1_doe$
  wait_duration 1000

  slio1_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO_DATA_OUT
  slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID high-Byte
  can_out$ = slio1_dout$
  wait_duration 1000
  slio1_dout$ = "<0><0><0><3><0><0>" ' SLIO_DATA_OUT   SLIO_DATA_OUT
  slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID einbauen
  can_out$ = slio1_dout$
  wait_duration 1000

  slio1_dpm1$ = "<0><0><0><6><0><0>" ' SLIO_DPM1
  slio1_dpm1$ = ntos$ ( slio1_dpm1$, 1, -2, slio1_id )' ID high-Byte
  value = 0
  t = ticks()
  stop_task keep_alive            ' Hauptschleife uebernimmt das jetzt
  for ever = 0 to 0 step 0        ' Endlosschleife ----------
    tmp = value shl 5             ' 10-Bit rechtbuendig in WORD
    tmp$ = ntos$ ( slio1_dpm1$, 4, -2, tmp )' value high-Byte zuerst
    can_out$ = tmp$
    put #CAN, tmp$
    value = modulo_inc ( value, 0, 7FFh, 7Fh ) ' inc step 7Fh
    if diff_ticks ( t ) > 50 then      ' Wert haengt von Bitzeit ab
      put #CAN, calib$                 ' sende Kalibrier-Botschaft
      t = ticks()
    endif
  next                            ' Endlosschleife ----------

no_slio_found:                    ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                        ' Schleifenvariable
  WORD ibu_fill                       ' Eingangspufferfuellung
  BYTE frameformat, msg_len           ' Frame-Format, Botschaftslaenge
  LONG r_id                           ' Empfangs-IDreceive ID
```

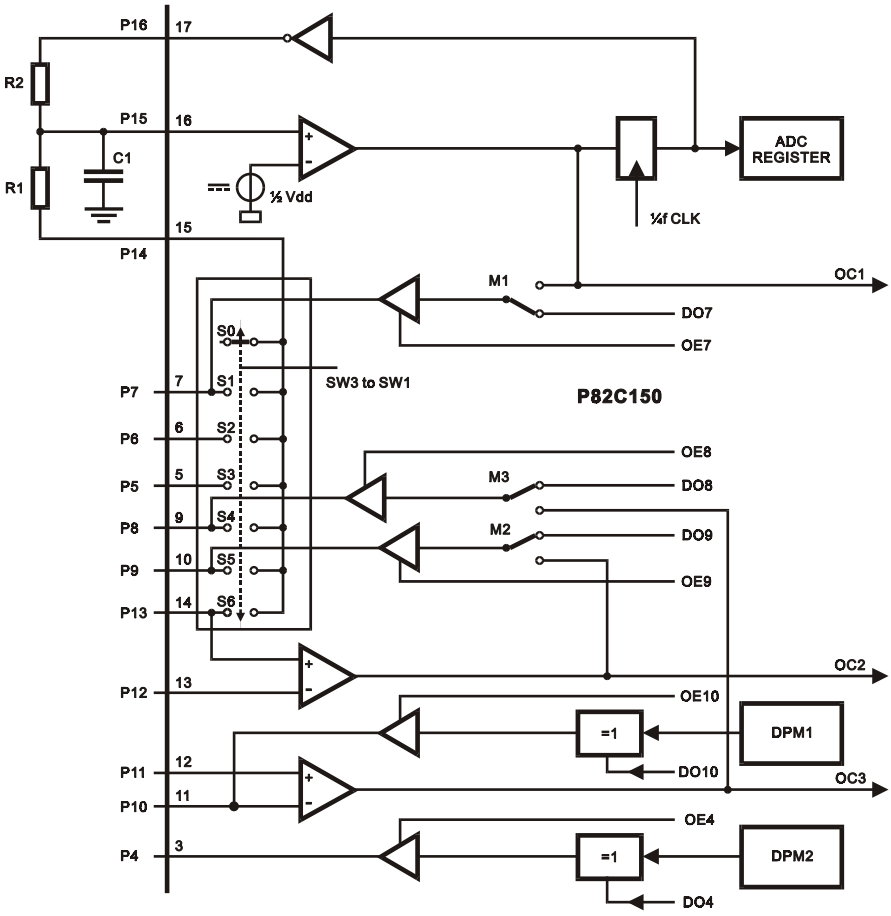
```
  for ever = 0 to 0 step 0              ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                  ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                    ' warte auf naechste Botschaft
      endif                             ' ----------------------------
                                        ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"      ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      print #LCD, "                     <13>DATA:";' loesche Datenanz.
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie und zeige an
        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
        next
      endif
    endif
  next
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'--------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                    ' Schleifenvariable
  WORD ibu_fill                   ' Eingangspufferfuellung
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  LONG r_id                       ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  slio_equip = 0                  ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
```

```
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2        ' in der Initphase kuerzer
  run_task keep_alive             ' sendet Kalibrier-Botschaften
                                  ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                     ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont                ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont                  ' warte auf naechste Botschaft
      endif
'-------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
                                        ' maskiere DIR-Bit weg
      r_id = r_id bitand SLIO_ID_MASK   ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then       ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h ' erzeuge 4-Bit-Adr.
      endif
      r_id = r_id shr 3                 ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id          ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1     ' zaehle die gefundene SLIO
                                        ' ignoriere hier Info in Daten
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END
```

```
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'-------------------------------------------------------------------
TASK keep_alive
  BYTE ever
  LONG t

  t = ticks()
  for ever = 0 to 0 step 0              ' Endlosschleife
    if diff_ticks ( t ) > 50 then       ' Wert haengt von Bitzeit ab
      put #CAN, calib$                  ' sende Kalibrier-Botschaft
      t = ticks()
    endif
    if can_out$ <> "" then
      put #CAN, can_out$                ' Sendebotschaft
      can_out$ = ""                     ' leer = gesendet
    endif
  next
END
```

Analog-Konfiguration

Der SLIO-Chip stellt einen recht komplexen Aufbau zur Erfassung von analogen Eingangssignalen zur Verfügung. Es wird eine Auflösung von 7..8 Bit erreicht. Die Wandelrate ist genauso hoch wie die Ausgaberate bei der quasi-analogen Ausgabe. Eine Wandlung benötigt 1024 Bitzeiten.



Der A/D-Wandler im oberen Teil der Abbildung verwendet den Pin P15 als Analog-Eingang sowie P16 als Feedback-Ausgang. Die Eingangsbeschaltung besteht aus R1 und R2 (je 100k) und C1 (3,3nF). Auf dem CAN-SLIO-Board steht am Pin Ai15 ein solcher Eingang zur Verfügung. Der Pin P15 muß offen bleiben, wenn Ai15 verwendet wird.

Wird mehr als nur ein Analogeingang benötigt, dann können die Pins P5, P6 P7, P8, P9 und P13 über einen Analogschalter auf den Analogeingang geschaltet werden. Der Ausgang des Analogschalters P14 wird dazu auf den Pin Ai15 gebrückt.

Anmerkung: der Pin Vref auf der gleichen 3-poligen Stiftleiste hat nichts mit dem Analogteil des SLIO zu tun. Vref gehört zu der CAN-Schnittstelle.

Eine weitere Möglichkeit des Analogteils besteht darin, mit Hilfe von Komparatoren nur das Über- oder Unterschreiten eines Schwellwertes anzuzeigen und, wenn gewünscht, auch gleichzeitig auf einem Ausgangspin auszugeben. Der Aufbau kann so konfiguriert werden, daß eine CAN-Nachricht erzeugt wird, wenn der Schwellwert über- oder unterschritten wird, oder daß eine selbstätige Regelung läuft, ohne daß CAN-Nachrichten erzeugt werden. Folgende Komparatoren stehen zur Verfügung:

Eingangspins	Ausgang	schaltbar auf Ausgangspin	Bemerkung
P15(+)	OC1	P7	(-) an interner Vref=1/2 VCC (2.5V) Verwendet den Eingang des ADC
P12(+), P13(-)	OC2	P8	
P11(+), P10(-)	OC3	P9	P10 ist auch analog-Ausgang DMP1, der dann nicht mehr verfügbar ist.

Die Ausgänge der Komparatoren können im Register ‚Analog Configuration' gelesen werden (Bits 12, 13 und 14). Ob diese Signale auf Ausgänge durchgeschaltet werden, wird in den Bits 8, 9 und 10 des Registers ‚Analog Configuration' eingestellt. Außerdem müssen dazu natürlich die Ausgangstreiber der betroffenen Pins aktiviert sein. CAN-Nachrichten werden dann automatisch erzeugt, wenn in den Registern ‚Event Positive Edge' und/oder ‚Event Negative Edge' die entsprechenden Bits für P8, P9 und P10 gesetzt sind.

Das ‚Analog Configuration'-Register (Adresse 5):

ADC	OC3	OC2	OC1	0	M3	M2	M1	S3	S2	S1	0	0	0	0	0

Bit(s)			Funktion
0...4			ohne Funktion
S3	S2	S1	Position des Analogschalters
0	0	0	kein Schalter geschlossen
0	0	1	P7 an P14
0	1	0	P6 an P14
0	1	1	P5 an P14
1	0	0	P8 an P14
1	0	1	P9 an P14
1	1	0	P13 an P14
1	1	1	reserviert
			Position der Monitoringschalter
M1			1: OC1 an P7
M2			1: OC3 an P9
M3			1: OC2 an P8
11			ohne Funktion
			Ergebnis der Analog-Komparatoren
OC1			1: P15 > Vref-intern (2.5V)
OC2			1: P13 > P12
OC3			1: P11 > P10
ADC			1: startet A/D-Wandlung

Es ergeben sich einige Regeln (in jedem Fall Kurzschlußgefahr):

- Wenn der Analogschalter gesetzt ist, darf P14 nicht Ausgang sein.
- Wenn M1 gesetzt ist, d.h. OC1 auf P7 geschaltet ist, darf P7 kein Ausgang sein.
- Wenn M2 gesetzt ist, d.h. OC3 auf P9 geschaltet ist, darf P9 kein Ausgang sein.
- Wenn M3 gesetzt ist, d.h. OC2 auf P8 geschaltet ist, darf P8 kein Ausgang sein.

Starten der A/D-Wandlung

Eine Wandlung benötigt 1024 Bitzeiten und liefert eine Genauigkeit von 7...8 Bit.

Der A/D-Wandler wird gestartet, wenn

- das Register ‚A/D-Conversion' beschrieben wird. Die Antwort erfolgt nach der Conversion automatisch.
- das Register ‚Analog Configuration' mit gesetztem Bit ‚ADC' beschrieben wird. Die Antwort erfolgt nach der Conversion automatisch.

Das Beispielprogramm liest auf dem Pin P15 gemessene Analogwerte. Die SLIO(s) werden mit Hilfe des Unterprogrammes ‚**find_slios**' zusammen mit der Task ‚**keep_alive**' gefunden, so wie unter ‚SLIOs auf dem Bus finden' beschrieben. Die Task ‚**show_slio**' zeigt die ankommenden SLIO-Nachrichten und den Identifier auf dem LCD in HEX zur Kontrolle an. Fremde Nachrichten, dazu zählen automatisch extended Frames, werden von ‚show_slio' verworfen, es wird nur der Empfangspuffer bereinigt.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_ANA1.TIG
' Liest Analogeingang des SLIO an P15 und zeigt den Wert als
' HEX-Zahl an (0...7FFh).
' Verbinde CAN-SLIO-Board
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50              ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000             ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210          ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung dieses SLIO

                                  ' globale Variable
BYTE no_of_slios                  ' Zaehler SLIOs im System
BYTE slio_stat                    ' SLIO status
WORD slio1_id, calib_id           ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask             ' Access-Code und Access-Mask
LONG slio_equip                   ' ein Bit fuer jede SLIO
LONG alive_wait                   ' 1/2 Wartezeit in 'keep_alive'
WORD acfg                         ' Analog-Konfigurationsbits
STRING calib$(5)                  ' Kalibrierungsbotschaft
STRING slio1_doe$(6)              ' data out enable
STRING slio1_acfg$(6), slio1_ain$(6)' analog config, analog in
STRING can_out$(13k)               ' Sendebotschaft zur SLIO

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' Endlosschleife
  WORD fi                         ' Pufferfuellung
  STRING tmp$(6)

  install_device #LCD, "LCD1.TDD"

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
     27 1F FF FF &                ' access mask
     10 45 &                      ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                      ' single filter mode, outctrl

                                  ' ID dieses SLIO: 0=standard frame
                                  ' + 2 ID-Bytes um 5 geshiftet
  slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO-Identifier
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  can_out$ = ""                   ' muss initialisiert sein
```

```
  call find_slios                 ' stellt Liste aller im System
                                  ' befindlichen SLIOs zusammen
  if slio_equip = 0 then          ' wenn keine SLIO gefunden
    goto no_slio_found            ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif
  wait_duration 1500
  print #LCD, "<1>";

  run_task show_slio              ' zeigt an, was die SLIO(s) senden

  slio1_doe$ = "<0><0><0><4><0><0>" ' SLIO_OUT_ENABLE alles Inputs
  slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
  can_out$ = slio1_doe$           ' ausgeben (lassen)
  wait_duration 100
                                  ' bereite die Analogkonfiguration vor
  slio1_acfg$ = "<0><0><0><5><0><0>" ' SLIO_OUT_ENABLE alles Inputs
  slio1_acfg$ = ntos$ ( slio1_acfg$, 1, -2, slio1_id)'ID high-Byte erst
'        aooo-mmmsss-----
  acfg = 1000000000000000b        ' nur 'start conversion bit' gesetzt
                                  ' ooo = 0, mmm = 0: kein Comparator
                                  ' sss = 0, kein Analogschalter
  for ever = 0 to 0 step 0        ' Endlosschleife ----------
    tmp$ = ntos$ ( slio1_acfg$, 4, -2, acfg)' Konfig.wort einfuegen
    can_out$ = tmp$               ' ausgeben (lassen), startete A/D
    wait_duration 1000
  next                            ' Endlosschleife ----------

no_slio_found:                    ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                    ' Schleifenvariable
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  WORD ibu_fill                   ' Eingangspufferfuellung
  WORD value                      ' Analogwert
  LONG r_id                       ' Empfangs-ID
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  for ever = 0 to 0 step 0        ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then              ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat    ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                      ' wenn keine SLIO-Botschaft
```

```
          if msg_len > 0 then          ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                 ' warte auf naechste Botschaft
        endif
      else                             ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then            ' wenn Daten
          get #CAN, #0, msg_len, data$ ' hole sie aus dem Puffer
        endif
        goto rx_cont                   ' warte auf naechste Botschaft
      endif                            ' ----------------------------
                                       ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"     ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      if msg_len = 3 then              ' wenn Daten
        get #CAN, #0, msg_len, data$   ' hole sie und zeige an
        if nfroms ( data$, 0, 1 ) = 8 then ' wenn Analogregister if anal
          value = nfroms ( data$, 1, 2 )
          value = byte_mirr ( value, 2 ) shr 5
          using "UH<3><3>   0.0.0.0.3"   ' Format HEX-Ausgabe
          print_using #LCD, " P15:";value;
        endif
      endif ' msg_len
    endif ' ibu_fill
  next
END


'---------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'---------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                     ' Schleifenvariable
  WORD ibu_fill                    ' Eingangspufferfuellung
  BYTE frameformat, msg_len        ' Frame-Format, Botschaftslaenge
  LONG r_id                        ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)        ' Botschaft und Daten

  slio_equip = 0                   ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5            ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"    ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2         ' in der Initphase kuerzer
  run_task keep_alive              ' sendet Kalibrier-Botschaften
                                   ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                      ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
```

```
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                         ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then            ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont                 ' warte auf naechste Botschaft
        endif
      else                               ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then              ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont                   ' warte auf naechste Botschaft
      endif
'-------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
                                         ' maskiere DIR-Bit weg
      r_id = r_id bitand SLIO_ID_MASK    ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then        ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h
      endif
      r_id = r_id shr 3                  ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id           ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1      ' zaehle die gefundene SLIO
                                         ' ignoriere hier Info in Daten
      if msg_len > 0 then                ' wenn Daten
        get #CAN, #0, msg_len, data$     ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"     ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'-------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'-------------------------------------------------------------------
TASK keep_alive
  BYTE ever

  for ever = 0 to 0 step 0             ' Endlosschleife
    put #CAN, calib$                   ' sende Kalibrier-Botschaft
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
    if can_out$ <> "" then
```

```
      can_out$ = ""                    ' leer = gesendet
    endif
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
  next
END
```

Zwei SLIOs an einem Bus

Das folgende Beispiel ist eine Kombination aus SLIO_DPM1.TIG und SLIO_HIGH_LOW.TIG. Es erzeugt DPM-Outputs auf SLIO1 (digital->analog) und digitale Rechteck-Ausgabe auf SLIO2. Stellen Sie auf einem CAN-SLIO-Board alle DIP-Schalter auf OFF, auf dem anderen CAN-SLIO-Board nur DIP-Schalter ID3 auf ON.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_2.TIG
' 2 SLIOs im System
' erzeugt DPM-Outputs auf SLIO1 (digital->analog)
' erzeugt digitale Ausgabe auf SLIO2 (Rechteck)
' Verbinde
' 1 CAN-SLIO-Board, alle DIP-Schalter aus
' 1 CAN-SLIO-Board, nur DIP-Schalter ID3 an
' messe am Pin P10
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50              ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000             ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210          ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung des SLIO1
#define ID_SLIO2 00000001000b     ' ID-Bit-Einstellung des SLIO2

BYTE no_of_slios                  ' Zaehler SLIOs im System
BYTE slio_12_found                ' Flag ueber gefundene SLIOs 1 und 2
BYTE slio_stat                    ' SLIO status
WORD slio1_id, slio2_id, calib_id ' ID der SLIOs und der Kalib.-Botsch.
LONG ac_code, ac_mask             ' Access-Code und Access-Mask
LONG slio_equip                   ' ein Bit fuer jede SLIO
LONG alive_wait                   ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                  ' Kalibrierungsbotschaft
STRING slio1_dpm1$(6), slio1_doe$(6)' DPM1 out, data out enable
STRING slio2_doe$(6)              ' data out enable SLIO2
STRING slio1_dout$
STRING slio2_dout$
STRING can_out$(13)               ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' Endlosschleife
  WORD fi                         ' Pufferfuellung
  WORD value, tmp                 ' Analogwert
  LONG t                          ' 'keep_alive' Zeit
    LONG digi                     ' fuer digitalen Ausgang
  STRING tmp$(6)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
```

```
    10 45 &                         ' Bitzeit 20usec (moegl.:8...50usec)
    08 1A"%                         ' single filter mode, outctrl

                                    ' ID dieses SLIO: 0=standard frame
                                    ' + 2 ID-Bytes um 5 geshiftet
calib_id = 0aah shl 5               ' ID der Kalibrier-Botschaften
can_out$ = ""                       ' muss initialisiert sein

call find_slios                     ' stellt Liste aller im System
                                    ' befindlichen SLIOs zusammen
if slio_equip = 0 then              ' wenn keine SLIO gefunden
  goto no_slio_found                ' dann Programm abbrechen
else
  print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
endif
if slio_equip bitand 000000011b <> 3 then ' wenn SLIO1/2 n. gefunden
  print #LCD, "<1Bh>A<0><3><0F0h>SLIO1/2 not found";
endif
wait_duration 1500
print #LCD, "<1>";

slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO1-Identifier
slio2_id = (SLIO_FIX_ID bitor ID_SLIO2) shl 5 ' SLIO2-Identifier
run_task show_slio                  ' zeigt an, was die SLIO(s) senden

slio1_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO1_OUT_ENABLE
slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
can_out$ = slio1_doe$
wait_duration 1000

slio1_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO1_DATA_OUT
slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID high-Byte
can_out$ = slio1_dout$
wait_duration 1000
slio1_dout$ = "<0><0><0><3><0><0>" ' SLIO1_DATA_OUT
slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID einbauen
can_out$ = slio1_dout$
wait_duration 1000

slio2_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO2_OUT_ENABLE
slio2_doe$ = ntos$ ( slio2_doe$, 1, -2, slio2_id)'ID high-Byte zuerst
can_out$ = slio2_doe$
slio2_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO2_DATA_OUT
slio2_dout$ = ntos$ ( slio2_dout$, 1, -2, slio2_id )' ID einbauen
wait_duration 1000

slio1_dpm1$ = "<0><0><0><6><0><0>" ' SLIO1_DPM1
slio1_dpm1$ = ntos$ ( slio1_dpm1$, 1, -2, slio1_id )' ID high-Byte
value = 0
digi = 0                            ' Wert fuer digitalen Ausgang
t = ticks()
stop_task keep_alive                ' Hauptschleife uebernimmt das jetzt
for ever = 0 to 0 step 0            ' Endlosschleife ----------
  tmp = value shl 5                 ' 10-Bit rechtbuendig in WORD
  tmp$ = ntos$ ( slio1_dpm1$, 4, -2, tmp )' value high-Byte zuerst
  put #CAN, tmp$
  tmp$ = ntos$ ( slio2_dout$, 4, -2, digi )' value high-Byte zuerst
  put #CAN, tmp$
  digi = digi + 1                   ' dig. Ausgangswert aendern
  value = modulo_inc ( value, 0, 7FFh, 7Fh ) ' inc step 7Fh
```

```
      put #CAN, calib$                ' sende Kalibrier-Botschaft
      t = ticks()
    endif
  next                             ' Endlosschleife ----------

no_slio_found:                     ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                          ' Schleifenvariable
  WORD ibu_fill                         ' Eingangspufferfuellung
  BYTE frameformat, msg_len             ' Frame-Format, Botschaftslaenge
  LONG r_id                             ' Empfangs-IDreceive ID
  STRING msg$(13), data$(8)             ' Botschaft und Daten

  for ever = 0 to 0 step 0               ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                 ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat       ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                         ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then            ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                   ' warte auf naechste Botschaft
        endif
      else                               ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then              ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                     ' warte auf naechste Botschaft
      endif                              ' ----------------------------
                                         ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"       ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      print #LCD, "                      <13>DATA:";' loesche Datenanz.
      if msg_len > 0 then                ' wenn Daten
        get #CAN, #0, msg_len, data$     ' hole sie und zeige an
        using "UH<2><2>   0.0.0.0.2"     ' Format HEX-Ausgabe
        for i = 0 to len ( data$ ) - 1   ' alle Bytes des Strings
          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
        next
      endif
    endif
```

```
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'--------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                    ' Schleifenvariable
  WORD ibu_fill                   ' Eingangspufferfuellung
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  LONG r_id                       ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  slio_equip = 0                  ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2        ' in der Initphase kuerzer
  run_task keep_alive             ' sendet Kalibrier-Botschaften
                                  ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                     ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then          ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                  ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then     ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont          ' warte auf naechste Botschaft
        endif
      else                        ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then       ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont            ' warte auf naechste Botschaft
      endif
'--------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
```

```
      r_id = r_id bitand SLIO_ID_MASK   ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then       ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h ' erzeuge 4-Bit-Adr.
      endif
      r_id = r_id shr 3                 ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id          ' setze Bitnummer in Liste
'      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      using "UH<1><1>   0 0 0 0 1"      ' fuer slio_equip Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>SLIO_equip ";slio_equip
      no_of_slios = no_of_slios + 1     ' zaehle die gefundene SLIO
                                        ' ignoriere hier Info in Daten
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'---------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'---------------------------------------------------------------------
TASK keep_alive
  BYTE ever
  LONG t

  t = ticks()
  for ever = 0 to 0 step 0             ' Endlosschleife
    if diff_ticks ( t ) > 50 then      ' Wert haengt von Bitzeit ab
      put #CAN, calib$                 ' sende Kalibrier-Botschaft
      t = ticks()
    endif
    if can_out$ <> "" then
      put #CAN, can_out$               ' Sendebotschaft
      can_out$ = ""                    ' leer = gesendet
    endif
  next
END
```