Scheckkartenmodul
80C166-CAN

Stand 11.07.94

Inhaltsverzeichnis

1. Einleitung

Mit dem Modul 80C166-CAN erhält man ein leistungsfähiges Gespann im Scheckkartenformat, bestehend aus der Siemens CPU SAB 80C166 bzw. der CPU SAB 88C166 und einem Intel CAN-Controller AN82527. Das Modul enthält eine große Anzahl an Peripheriefunktionen und verfügt durch eine leistungsfähige 16-Bit-CPU über ausreichende Reserven in der Verarbeitungsleistung, so daß sich dieses Modul insbesondere für verteilte Echtzeitapplikationen hervorragend eignet.

Das Modul 80C166-CAN ist in zwei Versionen erhältlich, die sich im jeweils unterstützten Busmodus unterscheiden. Diese werden im folgenden mit M (multiplexed) und N (non-multiplexed) bezeichnet. Zwischen diesen Varianten gibt es im wesentlichen zwei Unterschiede. Der erste liegt in der Anzahl der verfügbaren Ports. Die Variante M hat wegen des gemultiplexten Adress- und Datenbus' 16 I/O-Leitungen mehr als die Variante N. Der zweite Unterschied liegt in der Rechenleistung. Die Variante N hat wegen des getrennten Adress- und Datenbus' bei Zero-Wait-States eine um 33% höhere Rechnerleistung als die Variante M. Die Varianten sind fest verdrahtet und die Entscheidung für eine der beiden Varianten muß demnach beim Bestellen des CPU-Moduls getroffen werden.

Als weitere Option ist eine Version mit Flash-EPROM erhältlich. Durch Bestücken des Moduls mit einem 88C166 anstelle des 80C166 steht im Prozessor ein 32Kbyte großer Flashspeicher zur Verfügung. Dieser Speicher kann direkt im System programmiert und gelöscht werden.

Zur einfachen Inbetriebnahme des Modul 80C166-CAN steht ein Evaluationsboard EVA166 zur Verfügung.

Diese Beschreibung gilt für folgende Baugruppen:

Modul 80C166-CAN-N	Non-multiplexed mit CAN
Modul 80C166-CAN-M	Multiplexed mit CAN

2. Übersicht Modul 80C166-CAN

- Prozessor Siemens SAB 80C166
- Quarzoszillator 40 MHz
- 100 ns kürzeste CPU Zykluszeit
- 256 Kbyte linearer Adressraum für Programm und Daten
- 256 Kbyte statisches RAM, 0 Wait-States
- 1 Kbyte schnelles statisches RAM im Prozessor
- 1 Bytewide-Sockel (DIL) für bis zu 256 Kbyte Boot-EPROM
  (27C512, 27C010 oder 27C020 bestückbar)
- Wahlweise Flash-EPROMS bestückbar, "In-System" programmierbar
- Full-CAN Controller Intel AN82527
- Batterie Backupsteuerung
- Resetlogik
- LED Funktionsanzeige
- Sämtliche Prozessorsignale auf Pfostenleiste verfügbar
- Daten-, Adress- und Steuerbus gepuffert
- 2 serielle Schnittstellen, TTL-Pegel
- Programmierbarer Watchdog
- 10 A/D-Kanäle mit 10-Bit Auflösung, 9,75µs Wandlungszeit
- Universelle Timer/Counter mit Auflösung 400/200 ns
- 16 Capture/Compare-Kanäle für z.B. PWM, Pulsweitenmessung, usw.
- 55 I/O-Pins zur freien Verfügung bei Variante M (Multiplexed)
  39 I/O-Pins bei Variante N (Non-multiplexed)
- Optional 32 Kbyte Flashspeicher im Prozessor (88C166)
- Stromaufnahme ca. 5V/250 mA bei 40 MHz
- Abmessungen 53 x 81 mm²

3. Anschluß- und Bestückungsplan

4. Schaltplan

5. Listing des Adressdekoder-GALs (U2)

```
Name       166can;
Partno     ;
Date       14.06.94;
Revision   3;
Designer   Foegele;
Company    FS FORTH-SYSTEME GmbH, 79206 Breisach;
Assembly   Modul 80C166-CAN;
Location   U13;
Device     g26cv12;

/*****************************************************************/
/* Modul Mit 80C166 und CAN-Controller Intel 82527.              */
/*                                                               */
/* Die Anbindung des 82527 erfolgt im 16 Bit Modus.              */
/* MOD0 und MOD4 haben Dopplefunktion.                           */
/* Waehrend eines Resets wird festgelegt, mit welchem Busmodus   */
/* die CPU bootet. MOD0=0 bewirkt, dass mit einem 16 Bit Modus   */
/* gebootet wird. MOD4=0 legt fest, dass mit einem gemultiplexter */
/* Datenbus gebootet werden soll.                                */
/* MOD0..MOD4 legen das Memory Mapping fest                      */
/*                                                               */
/* Die Ausdekodierung der Adressleistungen A17..A11 ergibt eine  */
/* minimale Blockgroesse von 2 Kbyte.                            */
/*                                                               */
/*****************************************************************/

/**  Inputs  **/
Pin 7        =  Vcc      ;  /* Achtung: Fehler im CUPL Vers. 4.2a */
Pin 21       =  GND      ;  /* Achtung: Fehler im CUPL Vers. 4.2a */

Pin [1,2]    = [MOD3..4] ;      /* selection memory map          */
Pin [3,4]    = [A11..12] ;
Pin 5        = !MSE      ;      /* Module Select Enable          */
Pin 6        =  A0       ;      /* EVEN/ODD-SELECTION            */
Pin 8        = !BHE      ;      /* Byte-High-Enable              */
Pin 9        = !BUSACT   ;      /* !BUSACT=1 -> boot from CPU-ROM */
Pin 10       = A13       ;
Pin 11       = !WR       ;
Pin [12..15] = [A14..17] ;
Pin [26..28] = [MOD0..2] ;      /* selection memory map          */
```

```
/**  Outputs  **/
Pin 16       =  EBC0     ;
Pin 17       =  EBC1     ;
Pin 18       = !CSCAN    ;      /* Chip Select CAn-Cintroller     */
Pin 19       = !CSE      ;      /* Chip Select Extern             */
Pin 20       = !CSRAM_L  ;      /* Chip Select RAM low  byte      */
Pin 22       = !CSRAM_H  ;      /* Chip Select RAM high byte      */
Pin 23       = !CSEPR    ;      /* Chip Select EPROM              */
Pin 24       = !WRL      ;      /* write low  for CAN-Controller  */
Pin 25       = !WRH      ;      /* write high for CAN-Controller  */

/** Declarations and Intermediate Variable Definitions **/

fld ADR=[A17..A0];          /* A1..A10 only Dummy, not connected */

RAM    =  ADR:[00000..0FFFF] & MSE & !MOD0
        # ADR:[10000..1FFFF] & MSE & !MOD1
        # ADR:[20000..2FFFF] & MSE & !MOD2
        # ADR:[30000..3EFFF] & MSE & !MOD3;

EPROM  =  ADR:[00000..0FFFF] & MSE &  MOD0
        # ADR:[10000..1FFFF] & MSE &  MOD1
        # ADR:[20000..2FFFF] & MSE &  MOD2
        # ADR:[30000..3EFFF] & MSE &  MOD3 &  MOD4;

CAN    =  ADR:[3F800..3FFFF] & MSE;

EXT    =  ADR:[30000..3EFFF] & MSE &  MOD3 & !MOD4
        # ADR:[3F000..3F7FF];

/**  Logic Equations  **/
CSCAN   = CAN;       /* Chip select CAN-Controller, Real-Time-Clock */
CSEPR   = EPROM;          /* Chip Select EPROM */
CSRAM_L = RAM & !A0;      /* Chip select RAM */
CSRAM_H = RAM & BHE;      /* Chip select RAM */
CSE     =  (EXT # !MSE);  /* Chip Select Extern */

WRL     = WR & !A0;
WRH     = WR &  BHE;

EBC0    =   MOD0 &  BUSACT & !MOD4      /*  8bit multiplexed */
         # !MOD0 &  BUSACT &  MOD4;     /* 16bit non-multiplexed */

EBC1    =  !MOD0 &  BUSACT;             /* 16bit nonmux und mux */
```

6. Beschreibung der Anschlüsse

6.1 Kontaktierung

Das Modul verfügt an der Unterseite über 128 Kontaktstifte, an denen alle für eine Applikation notwendigen Signale, wie Spannungsversorgung, gepufferte Busse, Ports, Steuerleitungen etc. zu finden sind. Es sind nicht alle Kontakte belegt. Die mit NC (not connected) gekennzeichneten Anschlüsse sollten freigelassen werden. Sie sind für spätere Erweiterungen reserviert. In Sonderfällen kann man über diese Anschlüsse weitere Verbindungen vom Modul zur darunterliegenden Platine führen. Außerdem befinden sich auf der Oberseite (neben der CPU) eine 5-polige Stiftleiste, die für die Verbindung zum ROM/RAM-Emulator TIME/SRS vorgesehen ist.

6.2 Spannungsversorgung

Betriebsspannung (VCC (65, 97)):

Die Betriebsspannung des Moduls von +5 Volt muß genauer als +/- 0,25 V stabilisiert werden. Die Stromaufnahme beträgt bei 40 MHz ca. 250 mA.

Batteriespannung (VBAT (21)):

Die Batteriespannung VBAT von ca. 3 Volt dient zur Sicherung des RAM-Inhalts bei fehlender VCC. Wird auf die Batterie verzichtet, so muß der Eingang VBAT mit GND verbunden werden. **Dieser Anschluß darf auf keinen Fall unbeschaltet bleiben und darf auch nicht mit Vcc verbunden werden.** Die Stromaufnahme an VBAT beträgt wenige µA, und bei Anliegen von VCC ca. 100 nA.

GND an Pin 32 und 64:

Masse der Spannungsversorgung.

Bezugspotential A/D-Wandler (VAGND (29)):

Die Masse für den Analog-Digital-Wandler ist auf dem Modul nicht mit GND verbunden, sondern direkt nach außen geführt. Für eine Verbindung zwischen GND und VAGND muß auf der Anwenderplatine gesorgt werden.

Referenzspannung (VAREF (61)):

Die Referenzspannung (4.8 V < VAREF < 5.2 V) für den Analog/Digital-Wandler muß hier angelegt werden. Der Anschluß muß immer beschaltet werden. Im Bedarfsfall kann hierfür die +5 Volt Versorgungsspannung VCC verwendet werden. Zur Ausnutzung der vollen Auflösung des A/D-Wandlers empfiehlt sich jedoch die Verwendung einer separaten stabilisierten Referenzspannung.

6.3 Steuersignale

Rücksetzen Eingang (-RSTIN(25)):

Das Modul kann sowohl über den Eingang -RSTIN als auch an der 5poligen Stiftleiste für den ROM/RAM-Emulator (J1) definiert zurückgesetzt werden. Dieser Eingang ist auf dem Modul entprellt, so daß man hier direkt einen Taster anschliessen kann. Bei Anlegen der Betriebsspannung (VCC) wird auch RESET (an der CPU) ausgelöst. In jedem Fall wird das RAM vor zufälligem Überschreiben geschützt.

Rückksetzen Ausgang (-RSTOUT(16)):

Der 80C166 verfügt über eine spezielle Instruktion (EINIT), die am Ausgang -RSTOUT das Ende der Initialisierung kennzeichnet. Auf dem Modul wird mit dieser Instruktion das Resetsignal des CAN-Controllers deaktiviert. RSTOUT steht ebenfalls auf der 128poligen Stiftleiste zur Verfügung, um ein ordnungsgemäßes Rücksetzen externer Peripherie auch bei internen RESET-Ursachen (Watchdog, SoftWareReSeT) zu ermöglichen.

Auswahl Speicherkonfiguration (-RBOOT(15)):

Über den Anschluß -RBOOT kann zwischen zwei verschiedenen Speicherkonfigurationen gewählt werden. In der Standardkonfiguration (-RBOOT unbeschaltet bzw. mit VCC verbunden) ist nach einem Reset im Codesegment 0 das EPROM selektiert, und die CPU holt sich die ersten Befehle aus dem EPROM. Der Logikbaustein sorgt dafür, daß der Prozessor mit dem richtigen Busmodus bootet. (Siehe auch Kapitel 5 und 7).

Wird der Anschluß -RBOOT dem Restausgang (RSTOUT) verbunden, ist nach einem Reset im Codesegment 0 das RAM selektiert. Diese Konfiguration ist insbesondere dann von Vorteil, wenn man mittels Bootstrap-Loader ein Programm in das RAM des Moduls geladen hat und dieses dann nach einem Reset starten möchte.

Auswahl Busstatus (-BUSACT(8)):

Mit diesem Anschluß kann ebenfalls das Bootverhalten des Prozessors beeinflußt werden. In der Standardkonfiguration (-BUSACT unbeschaltet bzw. mit GND verbunden) bootet der Prozessor aus dem externen RAM bzw. ROM, je nach Zustand der Leitung -RBOOT.
Ist der Anschluß -BUSACT mit +5V verbunden, bootet der Prozessor aus dem CPU-internen ROM bzw. Flash-EPROM.

Programmierspannung (VPP(9)):

Hier kann die Programmierspannung für das CPU-interne Flash-EPROM angeschlossen werden. Zur Programmierung muß hier eine Spannung zwischen 11,6V und 12,4V anliegen. Der Strombedarf beträgt maximal 50 mA.

Es ist unbedingt darauf zu achten, daß die Programmierspannung erst nach Ende des Resets eingeschaltet wird.

Achtung: **Die Programmierspannung darf nur angeschlossen werden, wenn auf dem Modul ein 88C166 bestückt ist. Bei einem 80C166 führt das Anlegen der Programmierspannung zur Zerstörung des Bausteins.**

NMI (-NMI(48):

Der Nicht-Maskierbare-Interrupt-Eingang steht ebenfalls auf der Pfostenleiste zur Verfügung. Er ist durch einen Widerstand mit VCC verbunden und braucht deshalb nicht beschaltet zu werden.

Modulfreigabe (-MSE(47)):

Ist dieser Eingang inaktiv (High), wird die Aktivierung der auf dem Modul integrierten Einheiten (EPROM, RAM, Real-Time-Clock, CAN-Controller) verhindert. Insbesondere kann man hiermit auf die Adressen des EPROM oder RAM externe Bausteine legen. Der Modulausgang -CSE zeigt mit Low-Pegel in diesem Fall an, daß auf den entsprechenden Adressen keine Bausteine auf dem Modul adressiert sind. Bei einfachen Applikationen ohne zusätzliche Peripherie ist -MSE mit GND zu verbinden. Im unbeschalteten Zustand sorgt ein Pull-Down auf dem Modul für eine Einwandfreie Funktion.

Auswahl externer Peripherie (-CSE(41)):

Dieses Signal wird Low, wenn eine Adresse angesprochen wird, die auf dem Modul nicht belegt ist. Betreibt man an dem Modul nur einen externen Baustein, so kann dieser direkt mit -CSE selektiert werden.

Schreibfreigabe (-WR(80)):

Port P3.13 muß auf dem Modul als WRite-enable-Signal verwendet werden. Dieses liegt sowohl gepuffert als -WR an der 128poligen Kontaktleiste, als auch ungepuffert am TIME-Verbinder (J1/Pin4) an. Die Leitung P3.13 steht damit nicht als normales I/O-Signal zur Verfügung.

Lesefreigabe (-RD(111)):

Das ReaD-enable-Signal steht gepuffert an der Kontaktleiste zur Verfügung. Für dieses Signal muß keine I/O-Leitung "geopfert" werden, da die Entwickler der CPU hierfür einen eigenen Pin vorgesehen haben.

Bushälftenauswahl (-BHE(110)):

Um den 16-Bit-Bus auch byteweise verwenden zu können, wird die Leitung P3.12 als ByteHighEnable verwendet. Da dieses Signal auch vom GAL(U13) ausgewertet wird, ist es nicht möglich, diesen Pin anderweitig zu verwenden. -BHE ist gepuffert.
Siehe auch P3.12/-BHE(90)

(System-)Taktausgang (CLKOUT(121)):

Dies ist P3.15, an dem der 80C166 seinen Systemtakt (beim Modul mit 40 MHz Oszillator also 20 MHz) ausgeben kann. Diese Funktion muß aber im Bedarfsfalle erst per Software aktiviert werden, wahlweise ist P3.15 als gewöhnlicher I/O-Port konfigurierbar.

(Echtzeituhr) Taktausgang (TAKT(78)):

Die auf dem Modul verwendete Echtzeituhr kann hier einen (unsymetrischen) Takt von 1/64 Hz oder 1 Hz erzeugen.
Die Echtzeituhr steht nur alternativ zum CAN-Controller zur Verfügung.

ALE ungepuffert (IALE(7))

Das ungepufferte ALE-Signal ist herausgeführt, um die Bootstrapfähigkeiten des 80C166 ausnutzen zu können. (Siehe entsprechendes Kapitel)

6.4 Adressbus

Adressbus (A0-A17):

Hier liegen bei Lese- und Schreib-Zugriffen die Adressen gepuffert an. Sie sind unabhängig von der Version M oder N nicht gemultiplext. Auch bei der Version M müssen diese Signale (und nicht die ungepufferten Signale an Port 1) verwendet werden.

Port 4 (P4.0(39), P4.1(40)):

Hier liegen entweder die ungepufferten obersten beiden Adressleitungen A16 und A17 (siehe auch vorangegangenen Absatz) oder Port P4. Da diese Leitungen auch vom GAL ausgewertet werden, ist die Verwendung als Port problematisch.

AdressLatchEnable (ALE(79)):

Dieser Ausgang kennzeichnet den Beginn eines Buszyklus'. Er ist gepuffert. Dadurch sind die Adressen mit der Aktivierung zeitgleich gültig.

6.5 Port 2

Port 2 (CC0-CC15):

Dies ist ein Allzweck-Port der auch mit der Capture-Compare-Einheit verbunden ist. Damit kann er nicht nur als Ein-/Ausgang (für jeden Pin einzeln), sondern auch als Interrupt-Eingang und zur Impuls-Messung oder Erzeugung verwendet werden.

6.6 Port 3

Da alle Anschlüsse des Port P3 mehrfache Bedeutung haben, werden im folgenden sämtliche Pins einzeln beschrieben. Möchte man eine Funktion nicht nutzen, so kann man den entsprechenden Anschluß natürlich als einen bidirektionalen Port verwenden. Dies betrifft P3.0 bis P3.11 sowie P3.14 und P3.15. Bei den beiden letzten muß das SYSCON-Register entsprechend konfiguriert sein, bei den anderen Leitungen wird der Pin immer sowohl als Port als auch mit Sonderfunktion betrieben und beide Signale (logisch UND) verknüpft. Um beispielsweise P3.10 als Ausgang einer asynchronen Schnittstelle zu betreiben (TxD0), muß P3.10 als Ausgang mit Pegel 1 geschaltet werden. Man kann also auch bei gesonderter Verwendung eines Ports als Eingang, den Pegel stets auch per Software abfragen.

T0IN (P3.0(96)):

Eingang zu CAPCOM-Timer T0 im Zähl-Betrieb. Timer 0 ist einer der beiden Timer, die als Referenz für den CaptureCompare-Block dienen. Er kann sowohl synchron zum Systemtakt arbeiten, als auch von einem externen Signal gespeist werden, das dann an diesem Anschluß liegen muß. Die Frequenz darf 1.25 MHz nicht überschreiten.

T6OUT (P3.1(128)):

GPT2 Timer 6 ist ein systemtakt-synchroner Auf/Abwärtszähler (durch Software), dessen Überläufe den Pegel an T6OUT wechseln können (Pegelwechsel bei jedem Über- oder Unterlauf), damit ist an diesem Anschluß eine Frequenz von 2.5 MHz bis 0.07 Hz erzeugbar. In Verbindung mit CAPIN (P3.2) ist zum Beispiel eine einfache digitale PLL-Schaltung möglich.

CAPIN (P3.2(95)):

Auch dieser Anschluß ist in Verbindung mit Timer 5 und 6 zu sehen. Je nach eingestellter Flanke (in Register T5CON) kann das Register CAPREL durch ein Signal an diesem Anschluß mit dem Inhalt von T5 geladen werden, und eine Interruptbedingung wird erzeugt, also auch als reiner Interrupteingang betreibbar. Außerdem kann beim Laden T5 gleichzeitig gelöscht werden.

T3OUT (P3.3 (127)):

Funktion ganz ähnlich zu T6OUT. Will man Nulldurchgänge von T3 signalisieren, setzt man (mittels Register DP3) P3.3 als Ausgang und T3OE (T3CON.9) auf 1 und der T3OUT folgt T3OTL (T3CON.10).

T3EUD (P3.4 (94)):

Ist T3UDE=1 (T3CON.8) so zählt T3 aufwärts, wenn der Pegel an T3EUD gleich dem Inhalt von T3UD (T3CON.7) ist, sonst abwärts.

T4IN (P3.5 (126)):

Ist T4 im Counter-Modus (T4I =2, 3 oder 4 in T4CON, T4M=1), so werden die Impulse an T4IN gezählt (Frequenz kleiner als 1.25 MHz).
Im Gated-Timer-Modus werden die aus dem Systemtakt abgeleiteten Impulse (f_{OSC}*2^(-16-T4I)) gezählt, wenn an diesem Pin der richtige Pegel liegt (je T4M=2 oder 3 in T4CON)
Im Reload-Mode (T4M=4) wird T3 mit dem derzeitigen Inhalt von T4 geladen, wenn an T4IN der richtige Flankenwechsel erfolgt (T4I=0..3).
Im Capture-Modus (T4M=5) wird umgekehrt T4 mit dem Inhalt von T3 geladen.

T3IN (P3.6(93)):

Ist T3 im Counter-Modus (T3M=1, T3I=1..3, beides in T3CON), so werden die passenden Flanken an T3IN von T3 gezählt. (Über die Zählrichtung siehe auch T3EUD.) Auch hier darf die Frequenz 1.25 MHz nicht übersteigen. Im Gated-Timer-Modus (T3M=2, 3) gilt das bei T4IN geschriebene analog.

T2IN(P3.7(125)): Siehe T4IN.

TxD1 (P3.8(92)):

Ausgang der seriellen Schnittstelle, 2. Kanal. Wird die Schnittstelle im asynchronen Modus betrieben (S0M=1,3,4,5,7 in S0CON, z.B. V24), so ist dies der Ausgang der seriellen Datenübertragung.

Achtung: Dieser Anschluß führt normalen CMOS-Pegel.

Für die meisten Datenübertragungen werden noch externe Treiber benötigt (für V24/RS232 zum Beispiel MAX 233 o.ä.) Im synchronen Modus (S0M=0) liegt hier der Takt, nicht die Daten an!

RxD1 (P3.9(124)):

Eingang der seriellen Schnittstelle im asynchronen Modus. Im synchronen Modus Datenaus- und (!) Eingang. Über Pegel und Formate gilt im Wesentlichen das bei TxD1 geschriebene.

TxD0 (P3.10(91)):

Siehe TxD1, nur Kanal 0.

RxD0 (P3.11(123)):

Siehe RxD1, nur Kanal 0.

-BHE (P3.12 (90)):

-BHE wird bereits auf dem Modul verwendet, und muß deshalb mit einem der ersten Befehle auf Ausgang und High geschaltet und im SYSCON-Register aktiviert werden. Es erlaubt das Byteweise Schreiben und Lesen auf dem 16-Bit-Bus und wird vom GAL ausgewertet, weshalb dieser Pin nur nach Änderung der GAL-Programmierung als I/O-Pin zur Verfügung steht.

-BHE am Modul (110) ist ein gepufferter Ausgang des gleichnamigen Signals vom Prozessor.

-WR (P3.13):

Wie bei -BHE, erlaubt überhaupt erst das Schreiben in RAM, RTC usw.; der mit -WR (80) bezeichnete Pin ist gepuffert, das ungepufferte Signal steht an der Stiftleiste zum Anschluß des TIME/SRS zur Verfügung (siehe auch oben).

-READY (P3.14(122)):

Dieser Anschluß kann auch als Port betrieben werden. Bei langsamer Peripherie kann hier ein Buszyklus verzögert werden (asynchroner Betrieb). Auf dem Modul ist es möglich, die Ready-Funktion zur Steuerung der Zugriffe auf den CAN-Controller zu ververwenden. Hierzu ist auf der Modulunterseite die Lötbrücke BR1 zu schließen.

CLKOUT (P3.15 (121)):

Siehe oben

6.7 Port 5

Port 5 (Analog-Port):

Dies ist entweder ein CMOS-Eingangs-Port mit Schmitt-Trigger Funktion oder der Eingang zum 10-Kanal-Multiplexer für den Analog-Digital-Konverter. Der logische Pegel der Eingänge kann jederzeit in Register P5 gelesen werden. Der Multiplexer wird über ADCH (in ADCON) programmiert, er kann auch auf automatisches Weiterschalten programmiert werden. In ADDAT erhält man in den oberen 4 Bit die Kanalnummer zum Ergebnis. Das Vorliegen eines Ergebnisses wird in ADCIC und Fehler in ADEIC signalisiert. Siehe auch VAGND und VAREF.

7. Die Speicheradressierung (Memory Map)

Die beste Information über die Adressierung erhält man aus dem GAL-Listing (siehe Kapitel 5). Auf dem Modul integriert sind 256 Kbyte RAM und ein Bytewide-Sockel für bis zu 256 Kbyte EPROM und ein CAN-Controller. Zusätzlich ist noch ein Bereich von 62 Kbyte bzw. 2 Kbyte für externe Peripherie vorgesehen. Um alle Möglichkeiten trotz des auf 256 Kbyte begrenzten Adressraums ausschöpfen zu können, ist eine Logik zur Umschaltung der Speicheraufteilung vorgesehen. Die Konfiguration erfolgt hierbei über die Portleitungen des CAN-Controllers.

Prinzipiell kann über die Leitungen MOD0 - MOD4 für jedes Codesegment des 80C166 (64 Kbyte Segmente) getrennt eingestellt werden, ob RAM oder EPROM selektiert wird. Hierbei selektiert ein Low-Pegel das RAM, ein High-Pegel das EPROM. (Siehe Tabelle auf der nachfolgenden Seite.)

Eine Sonderstellung nimmt jedoch das Segment 3 ein. Hier geschieht die Auswahl über die Leitungen MOD3 und MOD4 zusammen. Um im Segment 3 das RAM zu selektieren, genügt es die Leitung MOD3 auf Low-Pegel zu programmieren. Der Pegel von MOD4 spielt dann keine Rolle. EPROM wird sellektiert, wenn MOD3 und MOD4 High-Pegel führen. Mit der Kombination MOD3=1 und MOD4=0 wird im Segment 3 weder EPROM noch RAM selektiert und statt dessen der Adressbereich für die externe Peripherie von 2 Kbyte auf 62 Kbyte vergrößert.

Sollte dieser Adressbereich von 62 Kbyte nicht ausreichen, so können, durch entsprechende Ansteuerung des Pins -MSE (Modul Select Enable), Adress-bereiche für interne Bausteine gesperrt und den externen Bausteinen zur Verfügung gestellt werden.

Im Adressbereich 0FA00h..0FDFFh wird immer das CPU-interne RAM und im Adressbereich 0FE00h..0FFFFh werden immer die Special-Function-Register angesprochen, unabhängig von der Decodierung im GAL.

Speicheraufteilung in Abhängigkeit von MOD4..0

MOD4..0	EPROM	RAM	EXT	CAN
x0000	-	00000..3EFFF	3F000..3F7FF	3F800..3FFFF
x0001	00000..0FFFF	10000..3EFFF	3F000..3F7FF	3F800..3FFFF
x0010	10000..1FFFF	00000..0FFFF 20000..3EFFF	3F000..3F7FF	3F800..3FFFF
x0011	00000..1FFFF	20000..3EFFF	3F000..3F7FF	3F800..3FFFF
x0100	20000..2FFFF	00000..1FFFF 30000..3EFFF	3F000..3F7FF	3F800..3FFFF
x0101	00000..0FFFF 20000..2FFFF	10000..1FFFF 30000..3EFFF	3F000..3F7FF	3F800..3FFFF
x0110	10000..2FFFF	00000..0FFFF 30000..3EFFF	3F000..3F7FF	3F800..3FFFF
x0111	00000..2FFFF	30000..3EFFF	3F000..3F7FF	3F800..3FFFF
01000	-	00000..2FFFF	30000..3F7FF	3F800..3FFFF
01001	00000..0FFFF	10000..2FFFF	30000..3F7FF	3F800..3FFFF
01010	10000..1FFFF	00000..0FFFF 20000..2FFFF	30000..3F7FF	3F800..3FFFF
01011	00000..1FFFF	20000..2FFFF	30000..3F7FF	3F800..3FFFF
01100	20000..2FFFF	00000..1FFFF	30000..3F7FF	3F800..3FFFF
01101	00000..0FFFF 20000..2FFFF	10000..1FFFF	30000..3F7FF	3F800..3FFFF
01110	10000..2FFFF	00000..0FFFF	30000..3F7FF	3F800..3FFFF
01111	0000..2FFFF	-	30000..3F7FF	3F800..3FFFF
11000	30000..3EFFF	00000..2FFFF	3F000..3F7FF	3F800..3FFFF
11001	00000..0FFFF 30000..3EFFF	10000..2FFFF	3F000..3F7FF	3F800..3FFFF
11010	10000..1FFFF 30000..3EFFF	00000..0FFFF 20000..2FFFF	3F000..3F7FF	3F800..3FFFF
11011	00000..1FFFF 30000..3EFFF	20000..2FFFF	3F000..3F7FF	3F800..3FFFF
11100	20000..3EFFF	00000..1FFFF	3F000..3F7FF	3F800..3FFFF
11101	00000..0FFFF 20000..3EFFF	10000..1FFFF	3F000..3F7FF	3F800..3FFFF
11110	10000..3EFFF	30000..3EFFF	3F000..3F7FF	3F800..3FFFF
11111	00000..3EFFF	-	3F000..3F7FF	3F800..3FFFF

x = dont't care

Nach einem Reset sind die Port-Leitungen des CAN-Controllers als Eingänge konfiguriert und erzeugen über interne Pull-Ups einen High-Pegel, womit praktisch im gesamten Adressraum das EPROM selektiert wird. Das trifft allerdings nur zu, wenn der Pin -RBOOT unbeschaltet bleibt. Ist -RBBOT mit -RSTOUT verbunden, wird damit die Selektion des RAMs im Codesegment 0 erzwungen.

8. Inbetriebnahme

8.1 Inbetriebnahme Hardware

Damit das Modul arbeiten kann, sind mindestens folgende Anschlüsse zu beschalten:

Pin	Name	Funktion	Anschluß
32, 64	GND	Masseanschluß	GND
65, 97	+5V	Versorgungsspannung	+5 ± 0,25V, ca. 250 mA
21	VBAT	Batteriespannung	ca. 3 V oder an GND
29	VAGND	Bezugspotential A/D-Wandler	GND
61	VAREF	Referenzspannung A/D-Wandler	z.B. mit +5V verbunden

Für das EPROM sind 3 Typen vorgesehen: 64k*8, 128k*8 und 256k*8. Für die einzelnen Typen muß auf die richtige Stellung des Lötjumpers BR2 (unter dem EPROM-Sockel) geachtet werden. (Siehe Tabelle). Bei Auslieferung ist BR2 in der Stellung +5V, so daß nur bei Verwendung der EPROMs 256k*8 zum Lötkolben gegriffen werden muß.

EPROM-Typ	Stellung BR2
64k*8	+5V
128k*8	egal
256k*8	A17

Die EPROMs der Größe 32k*8 werden nicht explizit unterstützt. CMOS-Typen sollten jedoch normalerweise problemlos arbeiten, sofern BR2 eine Verbindung von Pin 30 nach +5V herstellt.

Werden die 28poligen Speicherbausteine (32k*8, 64k*8) verwendet, müssen diese so in die Fassung eingesetzt werden, daß die Pins 1, 2, 31 und 32 des Sockels frei bleiben.

Der Einsatz von Flash-EPROMs ist ebenfalls möglich, wobei diese dann auch in der Schaltung programmiert werden können. Die 12V Typen werden allerdings nicht unterstützt. Vorgesehen sind folgende Typen: 29F010, 29F040. Beim 29F040 ist jedoch nur die obere Hälfte des Speichers nutzbar.

Bei Anlegen der Versorgungsspannung wird auf dem Modul automatisch ein Resetimpuls mit definierter Länge erzeugt und damit die CPU zurückgesetzt. Anschließend beginnt die CPU mit der Abarbeitung des ersten Befehls an der Adresse 0. Unter dieser Adresse wird im Normalfall (-RBOOT=1) das EPROM angesprochen.

Meistens wird die Software in irgendeiner Art mit einem Hostrechner kommunizieren. Wird dafür eine der seriellen Schnittstellen verwendet, so muß diese mit dem Rechner verbunden werden. Im Allgemeinen wird dies eine V24/RS232-Schnittstelle sein, so daß noch externe Treiber nötig sind (etwa MAX233 o.ä.).

Arbeitet man mit dem ROM/RAM-Emulator TIME/SRS, so kann anstelle der seriellen Schnittstelle die virtuelle Kommunikationsschnittstelle des Emulators zum Informationsaustausch zwischen Modul und Hostrechner verwendet werden. Dazu müssen die Signale Write-Enable und Reset vom TIME bzw. SRS an Stiftreihe J1 (TIME-Verbinder) angeschlossen werden.

Soll der CAN-Controller Interrupts auslösen können, muß auf der Modulunterseite die Lötbrücke BR3 geschlossen werden. Der Port 2.8 steht dann nur noch eingeschränkt zur Verfügung.

8.2 Inbetriebnahme Software

Auf den nachfolgenden Seiten ist beispielhaft eine Initialisierung des 80C166-CAN dargestellt.

Zunächst müssen die Steuerleitungen -WR und -BHE aktiviert werden. Hierzu muß der entsprechende Portpin auf Ausgang und High geschaltet werden. Außerdem muß die Funktion BHE explizit im Register SYSCON aktiviert werden, um später einen Bytezugriff auf das 16 Bit breite RAM zu ermöglichen. Beim Booten aus einem der 8 Bit Modi ist BHE nicht automatisch aktiv.

Anschließend wird eine Routine angesprungen, die den ROM-Inhalt ins RAM kopiert. Dieses Unterprogramm muß unbedingt im CPU-internen RAM ausgeführt werden, was die Routine EX_INTRAM erledigt. Ist darauf zu achten, daß das gesamte Unterprogramm keine absoluten Sprünge oder Unterprogrammaufrufe enthält.

In der Routine ROM2RAM wird zunächst der Befehl EINIT ausfgeführt um den Reset des CAN-Controllers zu beenden. Anschließend folgt eine Teil-Initialisierung des CAN-Controllers, insbesondere der Portleitungen, die die Speicherkonfiguration steuern. Beim Zugriff auf den CAN-Controller ist zu beachten, daß hier, um das Timing des CAN-Controllers einhalten zu können, mit mehreren Wait-States zugegriffen wird. Von der Möglichkeit das Timing über die Ready-Leitung zu steuern, sollte an dieser Stelle abgesehen werden.

Vor dem eigentlichen Kopiervorgang wird überprüft, ob der Prozessor nicht bereits im 16 Bit Modus läuft und damit aus dem RAM gebootet hat. In diesem Fall soll natürlich der RAM-Inhalt nicht überschrieben werden. Das Kopieren erfogt dann in Blöcken zu 16 Kbyte (Unterprogramm ONEPAGE), wobei für jedes Wort (16 Bit) zwischen ROM und RAM umgeschaltet wird.

Im Beispielprogramm wird der Bereich von 0 bis 0F9FF kopiert. Prinzipiell kann der Kopierbreich auf fast den gesamten Adressbereich ausgedehnt werden. Ausgenommen bleibt jedoch der CPU-interne Bereich 0FA00H-0FFFFH und ein Bereich mit 4 Kbyte am oberen Ende des Adressbereichs für die externe Peripherie und den CAN-Controller (3F000H..3FFFFH).

Zum Abschluß der Initialisierung wird dann noch die gewünschte Speicheraufteilung eingestellt, die LED und der CAN-Treiber eingeschaltet und zum Hauptprogramm weiterverzweigt.

```
;*********************************************************************
;* EQUs for CAN-Controller                                           *
;*********************************************************************
CAN_BASE_PAGE  EQU       0FH         ; CAN-Controller is located at
CAN_BASE_OFFS  EQU    03800H         ; 3F800H..3FFFFH
CAN_ADDRSEL    EQU       3F8H        ; value of ADDRSEL1 for CAN
                                     ; 2 Kbyte block
CAN_BUSCON     EQU     0689H         ; ALE length, 16 Bit MUX, no ready
                                     ; 6 wait, no memory tri state
I82527_CR      EQU       0           ; control register
I82527_CPUIR   EQU       2           ; CPU interface register
I82527_P2CONF  EQU    0AFH           ; port 2 config register
I82527_P2IN    EQU    0CFH           ; port 2 input register
I82527_P2OUT   EQU    0EFH           ; port 2 output register
;*********************************************************************
ORG    0             ; CPU starts at address 0
COLD_ENTRY:
       JMPS   0,START        ; jump to program start
                             ; reserve space for vector table
ORG    200H
START: DISWDT
       BSET   P3.13          ; WR = inactive
       BSET   DP3.13         ; WR = output
       NOP                   ; dealy for pipeline
       BSET   P3.12          ; BHE = inactive
       BSET   DP3.12         ; BHE = output
       BCLR   SYSCON.9       ; enable BHE for RAM

       MOV    R0,#ROM2RAM
       MOV    R2,#ROM2RAM_END
       CALLA  CC_UC,EX_INTRAM       ; execute ROM2RAM in internal RAM
       JMP    CC_UC,MON166

;*********************************************************************
;*  For module 80C166-CAN:                                           *
;*  Copies the contence of the EPROM in the address range            *
;*  0h..0F9FFh to RAM.                                               *
;*  The subroutine ROM2RAM must run in the CPU internal RAM.         *
;*  The complete subroutine must be relocatible or linked to         *
;*  address 0FA00h.                                                  *
;*********************************************************************
```

```
ROM2RAM:
       ;************************************************************
       ;* initialize the CAN-Controller first                      *
       ;************************************************************
       EINIT                        ; end of reset for CAN-Controller
WAIT_RES:
       MOV     R4,#I82527_CPUIR     ; pointer to CPU interface register
       CALLR   CAN_BYTE_READ
       JB      R5.7,WAIT_RES        ; wait until end of reset
       MOVB    RL5,#41H             ; set memory cycle
       CALLR    CAN_BYTE_WRITE
       ;
       MOV     R4,#I82527_CR        ; pointer to control register
       MOVB    RL5,#41H
       CALLR    CAN_BYTE_WRITE      ; enable access to config registers
       ;
       MOV     R4,#I82527_P2OUT     ; pointer to port 2 output register
       MOVB    RL5,#1FH             ; LED off, ROM active
       CALLR    CAN_BYTE_WRITE      ; enable CAN-Bus driver
       ;
       MOV     R4,#I82527_P2CONF    ; pointer to port 2 config register
       MOVB    RL5,#7FH             ; P2.0..P2.6 as output
       CALLR    CAN_BYTE_WRITE      ;
       ;
       MOV     R4,#I82527_CR        ; pointer to control register
       MOVB    RL5,#01H
       CALLR    CAN_BYTE_WRITE      ; disable access to config registers

       ;************************************************************
       ;* In case of booting from ROM, copy several pages          *
       ;************************************************************
       JB      SYSCON.7,IS_16BIT    ; skip if already in 16 bit mode
                                    ; in case of booting from RAM
       MOV     R0,SYSCON
       AND     R0,#00C0H            ; mask bus type
       MOV     BUSCON1,R0           ; BUSCON1 disabled
       BSET    BUSCON1.10           ; BUSCON1 enabled
       XOR     SYSCON,#00C0H        ; switch from 8 bit to 16 bit

       SCXT    DPP2,#0
       MOV     R2,#0C000H
       CALLR   ONEPAGE              ; copy page 0
       MOV     DPP2,#1
       CALLR   ONEPAGE              ; copy page 1
       MOV     DPP2,#2
       CALLR   ONEPAGE              ; copy page 2
       MOV     DPP2,#3
       MOV     R2,#0BA00H           ; don't copy internal RAM and SFRs
       CALLR   ONEPAGE              ; copy page 3
       POP     DPP2
```

```
       ;*************************************************************
       ;* end of copy                                               *
       ;*************************************************************
IS_16BIT:
       MOV     R4,#I82527_P2OUT       ; pointer to port 2 output register
       MOVB    RL5,#2EH               ; LED on, 0..0F9FF=RAM
       CALLR   CAN_BYTE_WRITE           ; 10000..2FFFF=ROM
                                      ; 30000..37FFF=external
       MOV     R0,SYSCON
       XOR     R0,#00C0H
       AND     R0,#00C0H
       BSET    R0.10
       MOV     BUSCON1,R0             ; set BUSCON1 fro ROM access
       MOV     ADDRSEL1,#204H         ; upper 128k
       RET

;*********************************************************************
;*  Subroutine ONEPAGE                                               *
;*  Copy one page with up to 16 Kbyte from RAM to ROM                *
;*  Parameter: DPP2 = current page                                   *
;*             R2   = number of bytes + 8000H                        *
;*             SYSCON is used to access to RAM                       *
;*             BUSCON1 is used to access to ROM                      *
;*********************************************************************
ONEPAGE:
       MOV     R1,DPP2                ; get current page
       SHL     R1,#6                  ; calculate coresponding ADDRSEL1
       OR      R1,#1
       MOV     ADDRSEL1,R1            ; block with 16 Kbyte
       MOV     R0,#8000H              ; start pointer
ONEPAGE1:
       MOV     R1,[R0]                ; read word from ROM
       MOV     R4,#I82527_P2OUT       ; pointer to port 2 output register
       MOVB    RL5,#0                  ; activate RAM
       CALLR   CAN_BYTE_WRITE           ;
       BCLR    BUSCON1.10             ; addressing via SYSCON
       ;
       MOV     [R0],R1                ; write word to RAM
       MOV     R4,#I82527_P2OUT       ; pointer to port 2 output register
       MOVB    RL5,#1FH                ; activate ROM
       CALLR   CAN_BYTE_WRITE           ;
       BSET    BUSCON1.10             ; addressing via BUSCON1
       ;
       ADD      R0,#2
       CMP      R0,R2
       JMPR    CC_NE,ONEPAGE1
       RET
```

```
;*********************************************************************
;*  Subroutine CAN_BYTE_READ                                         *
;*  read byte from CAN-Controller                                    *
;*  Parameter: R4  = register offset                                 *
;*             RL5 = ouput value                                     *
;*********************************************************************
CAN_BYTE_READ:
       SCXT    ADDRSEL1,#CAN_ADDRSEL  ;
       SCXT    BUSCON1,#CAN_BUSCON    ; set bus config for CAN
       SCXT    DPP0,#CAN_BASE_PAGE    ; set DPP to CAN-Controller
       NOP                            ; delay for pipeline
       MOVB    RL5,[R4+#CAN_BASE_OFFS] ; read byte
       POP     DPP0
       POP     BUSCON1
       POP     ADDRSEL1
       MOVB    RH5,#0
       RET

;*********************************************************************
;*  Subroutine CAN_BYTE_WRITE                                        *
;*  write byte to CAN-Controller                                     *
;*  Parameter: R4  = register offset                                 *
;*             RL5 = input byte                                      *
;*********************************************************************
CAN_BYTE_WRITE:
       SCXT    ADDRSEL1,#CAN_ADDRSEL  ;
       SCXT    BUSCON1,#CAN_BUSCON    ; set bus config for CAN
       SCXT    DPP0,#CAN_BASE_PAGE    ; set DPP to CAN-Controller
       NOP                            ; delay for pipeline
       MOVB    [R4+#CAN_BASE_OFFS],RL5 ; write byte
       POP     DPP0
       POP     BUSCON1
       POP     ADDRSEL1
       RET

;*********************************************************************
;*  Subroutine EX_INTRAM                                             *
;*  Parameter: R0=start address, R2=end address                      *
;*  Copies a specified routine to internal RAM and executes          *
;*********************************************************************
EX_INTRAM:
       MOV     R1,#0FA00H
EX_INTRAM1:
       MOV     [R1+],[R0]
       ADD     R0,#2
       CMP     R0,R2
       JMPR    CC_NE,EX_INTRAM1
       JMPA    CC_UC,0FA00H
ROM2RAM_END:
```

9. Evaluations-Board

9.1. Eigenschaften

Das Evaluations-Board EVA 166 ist ein Board im Einfach-Europaformat 160 x 100 mm². Es enthält neben den Aufnahmesockeln für das Modul80C166-CAN folgende Komponenten:

- Speisestecker für mitgeliefertes +5 Volt Steckernetzteil
- Reset-Taster
- Lithium Batterie 3,6 Volt für RAM
- RS232 Pegelwandler für beide seriellen Schnittstellen
- Frontseitiger 9-poliger D-SUB Stecker für serielle Schnittstelle Kanal 0
- Frontseitiger 9-poliger D-SUB Stecker für Anschluß an den CAN-Bus nach CIA
- 10 poliger Bergstecker zum Anschluß der 2. seriellen Schnittstelle
- Jumper für Bootstrap-Enable
- Steckbare LED-Zeile an Port P2.0 bis P2.7 inkl. Spannungsanzeige
- Steckbarer DIP-Schalter an Port P2.8 bis P2.15
- Großzügiges Löt- bzw. Wrap-Feld für anwendungsspezifische Erweiterungen
- Lötaugen und Befestigungslöcher für DIN 41612 Stecker

Dank des mitgelieferten Steckernetzteiles erlaubt das EVA-Board die sofortige Inbetriebnahme des Moduls 80C166-CAN ohne zusätzliche Hilfsmittel. Es unterstützt dabei alle Funktionen des CPU-Moduls. Zur einfachen Inbetriebnahme von Software bzw. für Demonstrationszwecke lassen sich 8 LED-Anzeigen sowie 8 DIP-Schalter vom Modul aus ansteuern bzw. einlesen.

Dank des großzügigen Löt- bzw. Wrap-Feldes eignet sich das EVA-Board auch für die Erstellung von Prototypenschaltungen sowie zum Einschub in 19" Systeme.

9.2. Schnittstellenbeschreibung

Folgende Stecker sind auf dem EVA-Board 166 vorgesehen:

Stecker J1:

- CAN-Bus
- D-SUB, 9polig male

Pin	Signal
1	NC
2	GND
3	CAN_L
4	NC
5	NC
6	CAN_H
7	GND
8	NC
9	NC

Stecker J2:

- RS232 Kanal 1
- Pfostenleiste, 10polig

Pin	Signal
1	NC
2	RXD
3	DTR
4	NC
5	TXD
6	NC
7	NC
8	NC
9	GND

Stecker J3:

- RS232 Kanal 0
- D-SUB, 9polig female

Pin	Signal
1	NC
2	TXD
3	RXD
4	NC
5	GND
6	DTR
7	NC
8	NC
9	NC

Stecker J4:

- 5 V Speisung
- Buchse für Steckernetzteil

Pin	Signal
1	+5V
2	GND

Stecker J6:

- 19" Backplane
- VG Steckverbinder nach DIN 41612

Der DIN 41612 Stecker J6 ist vom Anwender zu bestücken und zu verdrahten.

LED-Zeile:

Die steckbare 10-fach LED-Zeile ist wie folgt belegt:

LED1 .. LED8	leuchtet wenn Port P2.0 .. P2.7 = Low
LED9	leuchtet wenn -RSTOUT = Low (bevor die Software-Instruktion EINIT ausgeführt wurde)
LED10	leuchtet wenn die +5 V Speisung des Steckernetzteils angeschlossen ist.

DIL-Schalter:

Der steckbare 8-fach DIL-Schalter ist wie folgt belegt:

DIL1 .. DIL8	wenn Schalter = On ist P2.8 .. P2.15 = Low (sonst Hi über Pull-up´s)

RS232-Schnittstellentreiber:

Die RS232 Schnittstellentreiber wandeln die TTL Signale in die RS232 Norm um.

JP1, RBT: RAM-Boot

JP1 offen	Modul bootet aus dem EPROM
JP1 gesetzt	Modul bootet aus dem RAM

JP2, CNF: RAM-Boot für altes Modul

JP2 offen	Modul bootet aus dem EPROM
JP2 gesetzt	Modul bootet aus dem RAM

JP3, NMI Aktivierung Bootstrap

JP3 offen	Normalmodus
JP3 gesetzt	Bootstrapmodus

JP4 Abschlußwiderstand CAN-Bus

JP4 offen	Abschlußwiderstand inaktiv
JP4 gesetzt	Abschlußwiderstand aktiv

JP5 Verbindung Analog GND

JP5 offen	Analog GND hat keine Verbindung mit GND
JP5 gesetzt	Analog GND ist mit GND verbunden

Achtung: **JP5 ist voreingestellt. Um die Verbindung zwischen Analog GND und GND aufzutrennen, muß eine Leiterbahn durchgetrennt werden.**

JP6 Batterie-Backup

Stellung 1-2	Batterie-Backup aktiv
Stellung 2-3	Batterie-Backup inaktiv

9.3 Bestückungsplan EVA-Board

9.4 Schaltplan EVA-Board

10. Bootstrap mit Modul 80C166-CAN

Bei den Prozessoren mit der Maskenkennzeichnung Cx ist eine Bootstrap-Routine eingebaut. In einem speziellen Modus ist die CPU damit in der Lage selbständig ein kleines Programm ins CPU-interne RAM zu laden und dieses anschließend auszuführen.

Zur Aktivierung der Bootstrap-Funktion müssen zwei Bedingungen erfüllt sein: Zunächst muß der Bootstrap-Modus freigeschaltet werden indem während eines externen Resets der Anschluß ALE auf High-Pegel gehalten wird. Anschließend muß nach Ende des Resets ein NMI ausgelöst werden.

Während eines Resets ist der Prozessoranschluß -ALE als Eingang geschaltet und zum Ende des Resets wird der anliegende Pegel gespeichert. Bleibt der Anschluß unbeschaltet, wird aufgrund des CPU-internen Pull-Down von ca. 8.2 kΩ ein Low-Pegel registriert und der Prozessor beginnt mit der normalen Abarbeitung des Anwenderprogramms. Durch Anlegen eines High-Pegels verzweigt die CPU in eine spezielle Schleife in der auf einen nicht maskierbaren Interrupt (NMI) gewartet wird. Ein NMI innerhalb von 10 ms nach dem Reset aktiviert nun die Bootstap-Routine. Wird die Wartezeit überschritten, führt die CPU einen Softwarereset aus, was zur normalen über die Konfigurations-Pins -BUSACT, EBC0 und EBC1 eingestellten Operation führt.

Zu beachten ist, daß die Bootstrap-Funktion nur durch einen Hardwarereset aktiviert werden kann, nicht über Softwarereset oder Reset durch Überlauf des Watchdog-Timers. Ferner ist zu berücksichtigen, daß die Bootstrap-Funktion unabhängig von -BUSACT, EBC0 und EBC1 aktiviert wird.

Das Modul 80C166-CAN ist für die Nutzung der Bootstrap-Funktion vorbereitet. Am Anschluß IALE muß hierzu ein Pull-Up von 1,5 kΩ angeschlossen und außerdem eine Verbindung zwischen IALE(7) und -NMI(48) hergestellt werden. Der Pull-Up sorgt dafür, daß nach Ende des Resets am IALE-Anschluß ein High-Pegel eingelesen wird. Wenige Taktzyklen später wird ALE als Ausgang geschaltet und erzeugt damit am -NMI-Anschluß ein High-to-Low Flanke, womit der Prozessor in die Bootstrap-Routine verzweigt.

Sobald die Bootstrap-Funktion aktiv ist, wird eine Systeminitialisierung durchgeführt. Hierbei werden einige Prozessorregister initialisiert, z.B. SP=FA40h, CP=FA00h) Danach wartet der Bootstrap-Loader auf ein '00'-Zeichen am Pin RxD0, um daraus die Baudrate zu bestimmen. Der Host muß das Null-Zeichen mit 8 Datenbits und einem Stopbit ohne Parität schicken. Nach Angaben der Fa. Siemens sollte die Übertragungsrate nicht mehr als 9600 Baud betragen, doch wurde in der Praxis bei einem 40 MHz Quartz selbst 57600 Baud noch einwandfrei verarbeitet. Danach wird die serielle Schnittstelle 0 des SAB 80C166 initialisiert und als Antwort das Zeichen '55h' zurück geschickt. Dieses Zeichen kann vom Host ausgewertet werden, um eine einwandfreie Verbindung zu erkennen. (Der SAB C167 schickt an dieser Stelle übringens das Zeichen 'A5', womit der Host die beiden Prozessortypen leicht unterscheiden kann.)

In diesem Zusammenhang ist zu beachten, daß der Pin TxD0 erst nach Empfang des Null-Zeichens als Ausgang geschaltet wird, d.h. daß diese Leitung zunächst floatet. Damit vom Host keine falschen Zeichen empfangen werden, muß diese Leitung unbedingt mit einem Pull-Up versehen werden. Bei einem Pegelwandler MAX233 ist dieser Pull-Up bereits integriert, bei anderen Bausteinen ist unter Umständen mit Problemen zu rechen.

Im nächsten Schritt erwartet dann der Prozessor eine feste Anzahl Zeichen an der seriellen Schnittstelle 0 und überträgt diese ins interne RAM ab der Adresse FA40h. In der Anzahl der übertragenen Bytes unterscheiden sich die Prozessoren mit der Maske CA von denen mit der Maske CB und Nachfolgende. Bei ersteren werden 960 Bytes erwartet, bei letzteren nur 32 Bytes. Sobald alle Zeichen empfangen wurden, beginnt der Prozessor mit der Abarbeitung des Programms an der Adresse FA40h.

Um die beiden Maskenvarianten unter einen Hut zu bekommen, wird zunächst ein 32 Byte langes Programm übertragen, das in der Lage ist, ein Synchronisationszeichen (ASCII V) zum Host zu schicken und anschließend das interne RAM von FA40h bis FDFFh (d.h. insgesamt 960 Bytes) mit Zeichen von der seriellen Schnittstelle zu füllen. (Siehe Listing 1.) Sind diese 32 Bytes übertragen, meldet sich im Falle der Maske CB der Target mit dem Synchronisationszeichen. Meldet sich der Target innerhalb einer bestimmten Zeit nicht, muß es sich um die Maskenvariante CA handeln und es werden 928 Füllzeichen geschickt.

Listing1: erster Teil der Bootstraproutine

```
        ORG             0FA40H
Boot1   PROC NEAR
        MOV             S0TBUF,#'V'         ; send synch char
        DISWDT                              ; switch off watchdog (for 80C166)
        MOV             R0,#0FA60H          ;
Receiv: JNB             S0RIR,Receiv        ; wait for char
        MOVB            R1L,S0RBUF          ; get char
        BCLR            S0RIR               ; clear interrupt flag
        MOVB            [R0],R1L            ; write char to memory
        ADD             R0,#1               ; increment pointer
        CMP             R0,#0FE00H
        JMPR            CC_NE,Receiv        ; go on until end off internal memory
Boot1   ENDP
```

In einem weiteren Schritt wird nun ein Programm von bis zu 928 Bytes ins interne RAM übertragen. Beispielsweise kann dieses Programm eine Applikation ins RAM laden und anschließend ausführen. wie ein derartiges Programm aussehen kann ist im nachfolgenden Listing gezeigt.

Zunächst wird ein Synchronisationszeichen zum Host geschickt. Danach muß der Prozessor initialisiert werden. Zu beachten ist in diesem Zusammenhang, daß bei einem Bootstrap das CPU-interne ROM aktiv ist und deshalb zunächst ausgeblendet werden muß.

Anschließend wird überprüft, ob der CAN-Controller vorhanden ist und dieser gegebenenfalls initialisiert. Jetzt kann der eigentliche Download stattfinden. Um die Übertragung einigermaßen sicher zu gestellten sollte eine Prüfsumme gebildet werden. Insbesondere kann auf diese Weise ein defektes RAM lokalisiert werden. Im Fehlerfall wird das gesamte File zum Host zurückgeschickt um eine Fehleranalyse zu ermöglichen.

Bei erfolgreichem Download, wird noch auf ein Synchronisationszeichen vom Host gewartet, bevor zur Adresse 0 im RAM verzweigt wird. Dies ist erforderlich, weil vor dem löschen des NMI-Flags im Trap-Flag-Register der Bootstrap-Jumper entfernt werden muß. Andernfalls würde der erste Buszyklus nach dem RETI-Befehl sofort einen NMI auslösen.

```
BOOT2:
        BSET    S0TIR           ; transmit buffer empty
        MOVB    RL5,#'X'        ; transmit 'X' for
        CALLR   SEND_CHAR       ; successfull load of download programm
        BSET    P3.13           ; WR inactive level
        BSET    DP3.13          ; WR output
        NOP                     ; delay for pipeline
        BSET    P3.12           ; BHE inactive level
        BSET    DP3.12          ; BHE output
        BCLR    SGTDIS          ; enable segmenatation
        BCLR    BYTDIS          ; enable byte wide write

        MOV     R0,SYSCON       ; direkt and/or to SYSCON here not allowed
        MOV     R1,R0
        AND     R1,#0FB3FH
        OR      R1,#00080H
        MOV     SYSCON,R1       ; disable ROM
;       OR      R0,#000EH       ; set wait states (here 1 wait state)
;       BSET    R0.5            ; switch off tristate time
        MOV     SYSCON,R0       ; set to normal operation

        JB      SYSCON.7,IS_16BIT       ; skip if already in 16 bit mode
        XOR     SYSCON,#00C0H           ; change to 16 bit
IS_16BIT:
        MOV     R4,#I82527_CPUIR        ; pointer to CPU interface register
        CALLA   CC_UC,CAN_BYTE_READ
        CMPB    RL5,#0E1H               ; Reset value of the CAN-Controller
        JMPR    CC_NE,FOR_ALL
```

```
IS_166_CAN:    ; the following lines are for module 80C166-CAN only
       ;************************************************************
       ;* initialize the CAN-Controller first                      *
       ;************************************************************
       EINIT                        ; end of reset for CAN-Controller
WAIT_RES:
       CALLR   CAN_BYTE_READ        ; get CPUIR of the CAN-Controller
       JB      R5.7,WAIT_RES        ; wait until end of reset
       MOVB    RL5,#41H             ; set memory cycle
       CALLR    CAN_BYTE_WRITE
       ;
       MOV     R4,#I82527_CR        ; pointer to control register
       MOVB    RL5,#41H
       CALLR    CAN_BYTE_WRITE      ; enable access to config registers
       ;
       MOV     R4,#I82527_P2OUT     ; pointer to port 2 output register
       MOVB    RL5,#1FH             ; LED off, ROM active
       CALLR    CAN_BYTE_WRITE      ; enable CAN-Bus driver
       ;
       MOV     R4,#I82527_P2CONF    ; pointer to port 2 config register
       MOVB    RL5,#7FH             ; P2.0..P2.6 as output
       CALLR   CAN_BYTE_WRITE       ;
       ;
       MOV     R4,#I82527_CR        ; pointer to control register
       MOVB    RL5,#01H
       CALLR    CAN_BYTE_WRITE      ; disable access to config registers

FOR_ALL:
       EINIT
       MOV     DPP0,#0              ; internal ROM is still active
       MOV     DPP1,#1              ; until DPP0 and DPP1 are set

       CALLR   GET_FILE             ; receive file and write to memory
       MOVB    RL5,#'Z'             ; transmit 'Z' for
       CALLR   SEND_CHAR            ; successfull download

       MOV     R5,R2
       CALLR   SEND_WORD            ; send checksum of receive
       CALLR   CALC_CHECK           ; calculate checksum in memory
       MOV     R5,R2
       CALLR   SEND_WORD            ; send checksum of memory

       CALLR   GET_CHAR             ; get acknowledge to checksum
       CMPB    RL5,#'Y'             ; checksum ok?
       JMPR    CC_EQ,EXIT           ; yes, exit to execute programm

       CALLR   SEND_FILE            ; memory dump to host, if error in download
       MOV     R0,#0AA55H
       MOV     R1,#055AAH
```

```
ENDLESS:
       MOV     R2,[R0]
       MOV     [R0],R1
       JMPR    CC_UC,ENDLESS          ; loop forever

EXIT:  CALLR   GET_CHAR               ; wait for char from host to go on
       BCLR    TFR.15                 ; clear NMI flag from booting
       MOV     R0,#0
       PUSH    R0                     ; PSW = 0
       PUSH    R0                     ; CSP = 0
       MOV     R0,DPP0:2              ; get IP from reset vector
       PUSH    R0                     ;
       RETI                           ; execute programm in RAM
BOOT ENDP


;*********************************************************************
;* SEND_CHAR: Send char to serial channel 0, RL5 = char              *
;*********************************************************************
SEND_CHAR PROC NEAR
WAIT_T: JNB    S0TIR,WAIT_T    ; wait for buffer empty
       MOVB    S0TBUF,RL5      ; send char
       BCLR    S0TIR           ; set buffer not empty
       RET
SEND_CHAR ENDP

;*********************************************************************
;* SEND_WORD: Send word to serial channel 0, R5 = word               *
;*********************************************************************
SEND_WORD PROC NEAR
       CALLR   SEND_CHAR       ; send low byte
       MOVB    RL5,RH5
       CALLR   SEND_CHAR       ; send high byte
       RET
SEND_WORD ENDP

;*********************************************************************
;* GET_CHAR: Receive char from serial channel 0, R5 = char           *
;*********************************************************************
GET_CHAR PROC NEAR
WAIT_R: JNB    S0RIR,WAIT_R    ; wait for char
       MOVBZ   R5,S0RBUF       ; get char
       BCLR    S0RIR           ; clear interrupt flag
       RET
GET_CHAR ENDP
```

```
;*********************************************************************
;* GET_WORD: Receive word from serial channel 0, R5 = word           *
;*********************************************************************
GET_WORD PROC NEAR
       PUSH    R0
       CALLR   GET_CHAR       ; get low byte
       MOVB    RL0,RL5
       CALLR   GET_CHAR       ; get high byte
       MOVB    RH5,RL5
       MOVB    RL5,RL0
       POP     R0
       RET
GET_WORD ENDP


;*********************************************************************
;* GET_FILE: Receive file from serial channel 0 and write to memory *
;*           beginning at address 0,                                 *
;*-------------------------------------------------------------------*
;* out: R3 = file length                                             *
;*      R2 = checksum receive                                        *
;*********************************************************************
GET_FILE PROC NEAR
       CALLR   GET_WORD       ; get length of file first
       MOV     R3,R5
       MOV     R1,#0          ; pointer = 0
       MOV     R2,#0          ; checksum receive = 0
GET_FILE1:
       CALLR   GET_CHAR       ; get char from serial interface
       ADD     R2,R5          ; calculate checksum
       MOV     [R1],RL5       ; write char to memory
       ADD     R1,#1          ;
       CMP     R1,R3          ; check end of file
       JMPR    CC_NE,GET_FILE1
GET_FILE ENDP

;*********************************************************************
;* SEND_FILE: Sends file from memory back to serial channel 0       *
;*            R3=file length                                         *
;*-------------------------------------------------------------------*
;* in: R3 = file length                                              *
;*********************************************************************
SEND_FILE PROC NEAR
       MOV     R1,#0          ; pointer = 0
SEND_FILE1:
       MOV     RL5,[R1+]
       CALLR   SEND_CHAR
       CMP     R1,R3
       JMPR    CC_NE,SEND_FILE1
       RET
SEND_FILE ENDP
```

```
;*********************************************************************
;* CALC_CHECK: Calculate checksum of module memory from 0 to end     *
;*             of file and send to host                               *
;*------------------------------------------------------------------- *
;* in: R3 = file length                                               *
;*********************************************************************
CALC_CHECK PROC NEAR
       MOV     R1,#0          ; pointer = 0
       MOV     R2,#0          ; cheksum = 0
       MOV     R5,#0          ; high byte of R5 mus be 0
CALC_CHECK1:
       MOVB    RL5,[R1+]
       ADD     R2,R5
       CMP     R1,R3
       JMPR    CC_NE,CALC_CHECK1
       CALLR   SEND_WORD      ; send memory checksum
       RET
CALC_CHECK ENDP
```