# HC12compact

Hardware Version 1.0

## Benutzerhandbuch

7. Juli 2008

ELMICRO Computer GmbH & Co. KG
Hohe Str. 9-13 D-04107 Leipzig
Telefon: +49-(0)341-9104810
Fax: +49-(0)341-9104818
Email: leipzig@elmicro.com
Web: http://elmicro.com

# Inhalt

# 1. Überblick

HC12compact ist ein universelles Mikrocontrollermodul auf Grundlage des Motorola Mikrocontrollers MC68HC812A4.

Neben den umfangreichen Möglichkeiten des 16Bit-Mikrocontrollers selbst, stehen dem Anwender auf dem HC12compact folgende Peripherieeinheiten zur Verfügung (z.T. optional):

- 512 KB Flash-Memory und 256 KB (optional: 1024 KB) RAM
- Real Time Clock (batteriegestützt)
- Analog/Digital-Wandler (12 Bit, 11 Kanäle)
- Digital/Analog-Wandler (12 Bit, 2 Kanäle)
- CAN Controller
- RS232 Schnittstellentreiber
- akustischer Signalgeber
- Indikator-LED

Kompakte Bauweise (Halbeuro-Formfaktor), niedrige Stromaufnahme und einfache Handhabung zeichnen diese Baugruppe aus; ebenso die für den HC12 erhältliche, umfassende Softwareunterstützung (Monitor, C-Compiler, BDM-Debugger).

## Lieferumfang

Die Basisversion der Baugruppe (Art.-Nr. HC12CO/1) wird wie folgt geliefert:

- Fertigbaugruppe mit 256 KB RAM und 512 KB Flash
- Optionen RTC, ADC, DAC und CAN *nicht* bestückt
  (die gewünschten Optionen bitte in jedem Fall zusammen mit dem Board bestellen, da es sich um SMT-Bauteile handelt)
- seitliche Stiftleisten unbestückt (können vom Anwender je nach Bedarf nach unten oder nach oben weisend eingelötet werden)
- Handbuch mit CD-ROM (Dokumentation und Software)
- TwinPEEKs Monitorprogramm im EEPROM des HC12

In der voll ausgestatteten Version der Baugruppe (Art.-Nr. HC12CO/FULL) sind folgende Peripheriefunktionen zusätzlich enthalten:

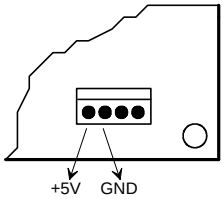
- Real Time Clock und LiMn-Batterie
- A/D-Wandler (12 Bit, 11 Kanäle)
- D/A-Wandler (12 Bit, 2 Kanäle)
- CAN Controller SJA1000

# 2. Schnellstart

Kein Mensch liest gern dicke Handbücher. Daher hier die wichtigsten Hinweise in Kürze. Wenn Sie sich jedoch über ein Detail einmal nicht sicher sind, dann informieren Sie sich am Besten in den nachfolgenden Kapiteln.

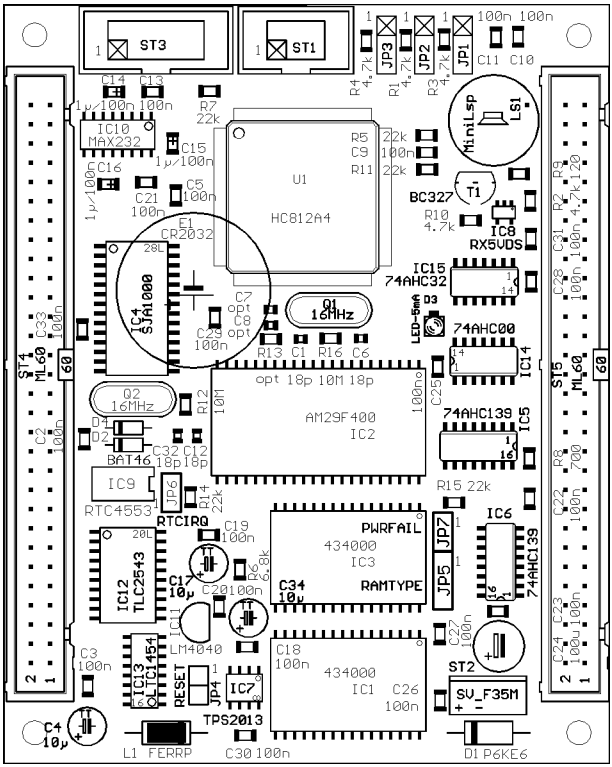
Und so können Sie beginnen:

- Überprüfen Sie die Baugruppe zuerst auf offenkundige Transportschäden.
- Verbinden Sie den Einplatinenrechner via RS232 mit Ihrem PC. Hierzu dient ein 1:1-belegtes Flachbandkabel. Pin 1 des Pfostenverbinders auf der Platine entspricht Pin 1 des 9-poligen Sub-D Verbinders der PC-Schnittstelle.
- Starten Sie auf Ihrem PC ein Terminalprogramm, wie z.B. OC-Console (kostenlos erhältlich auf unserer Website!).
- Stellen Sie die Baudrate auf 19200 Baud. Schalten Sie alle zusätzlichen Protokolle aus.
- Schließen Sie die (stabilisierte!) Versorgungsspannung an den Einplatinenrechner an. Vergewissern Sie sich **zuvor** von der richtigen Spannung und Polarität:




- Daraufhin startet das Monitorprogramm und zeigt eine Systemmeldung an. Mit Ausgabe des Promptzeichens erwartet es Ihre Anweisungen.

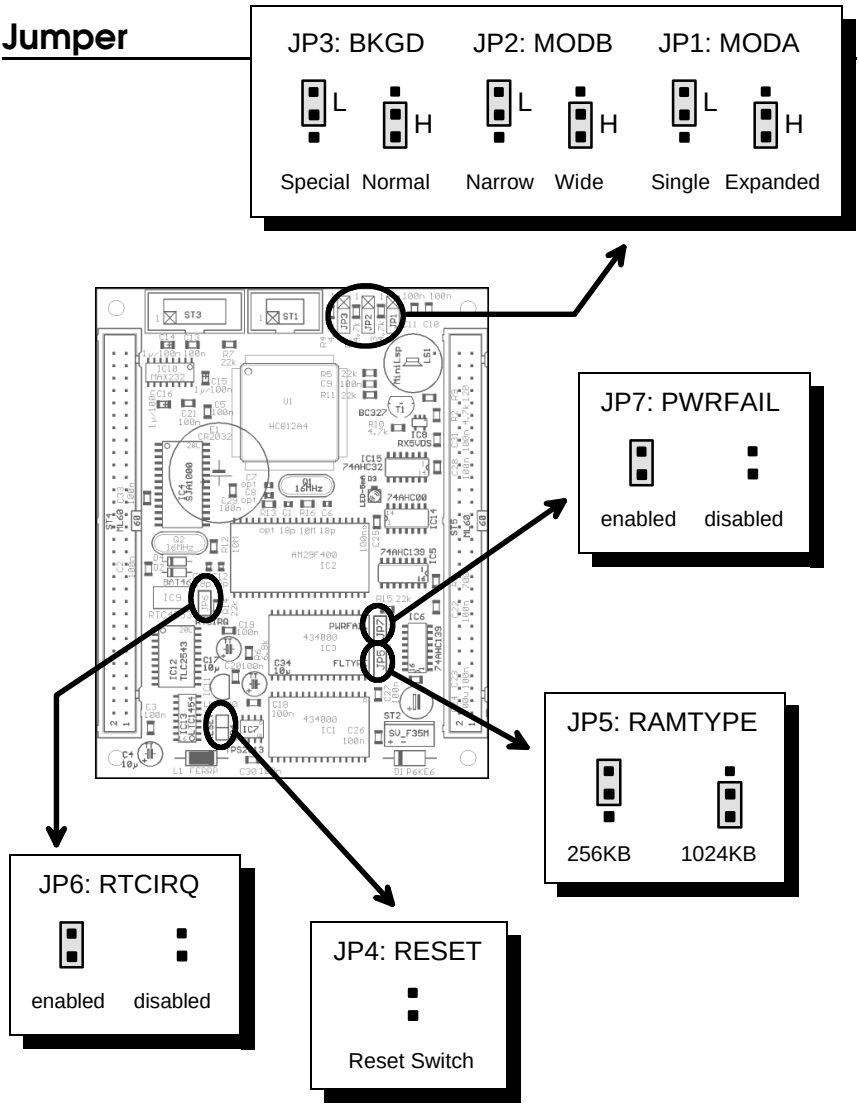
# 3. Bestückungsplan




*Bestückungsplan*

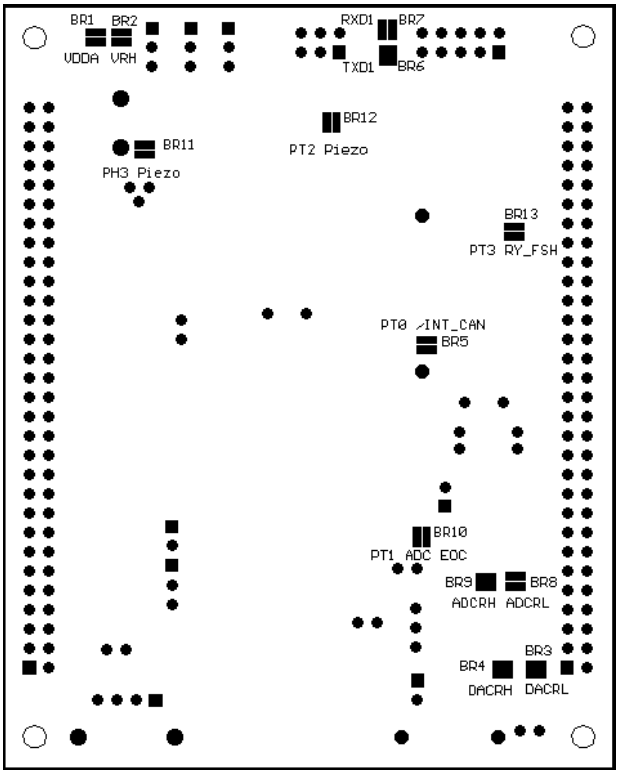
# 4. Jumper und Brücken

## Jumper




*Lageplan und Belegung der Jumper*

## Lötbrücken

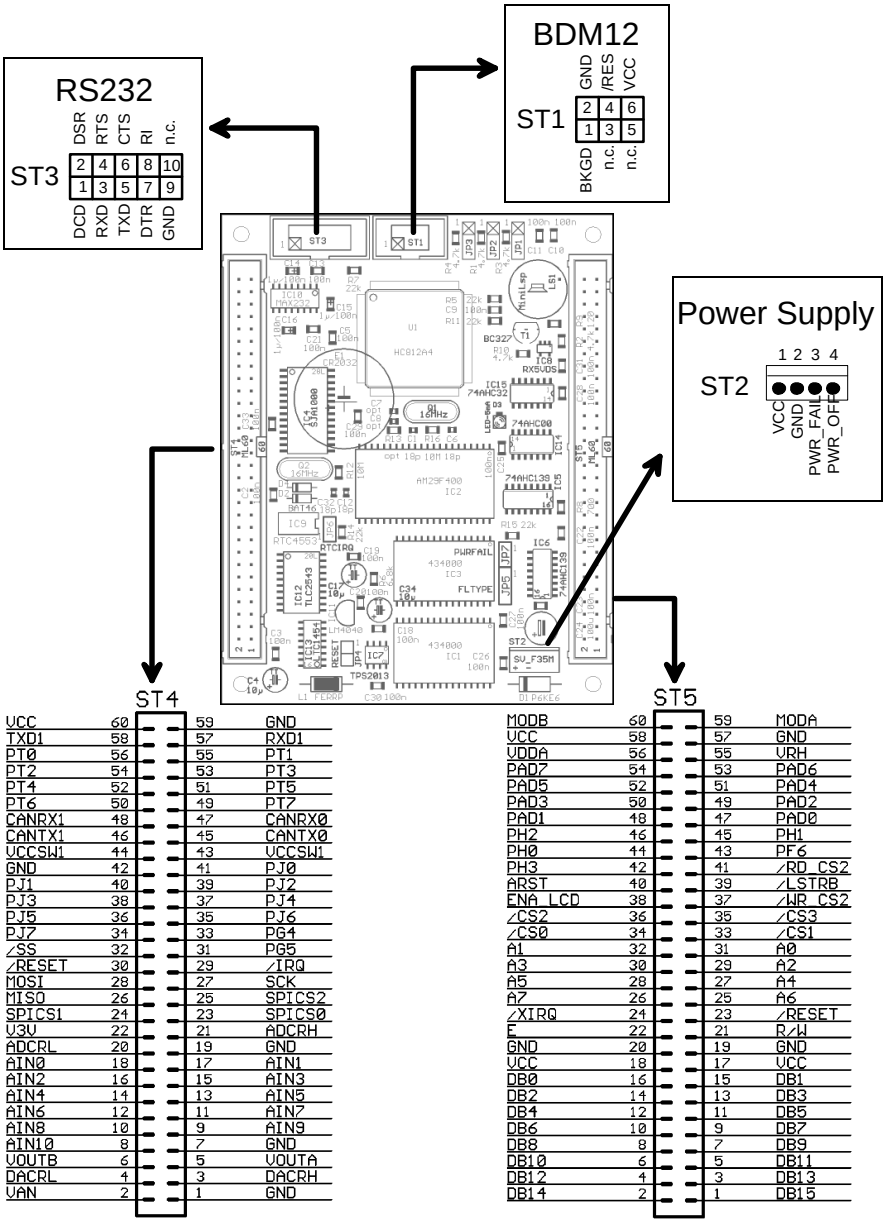



*Lageplan der Lötbrücken auf der Platinenunterseite*

| Brücke | Funktion | Beschreibung | * |
|---|---|---|---|
| BR1 | VDDA | HC12 VDDA = VCC | X |
| BR2 | VRH | HC12 VRH = VCC | X |
| BR3 | DACRL | DAC XA/B = GND | X |
| BR4 | DACRH | DAC REFHA/B = REFOUT | X |
| BR5 | /INT_CAN->PT0 | CAN Interrupt Erkennung | O |
| BR6 | TXD1 | TXD1 treibt DTR | O |
| BR7 | RXD1 | DSR treibt RXD1 | O |
| BR8 | ADCRL | ADC VREF- = GND | X |
| BR9 | ADCRH | ADC VREF+ von IC11 | X |
| BR10 | ADC_EOC->PT1 | ADC End Of Conversion Erkennung | O |
| BR11 | PH3->Piezo | PH3 treibt Signalgeber | O |
| BR12 | PT2->Piezo | PT2 treibt Signalgeber | X |
| BR13 | RY_FSH->PT3 | Flash Ready Erkennung | O |

* Standardeinstellungen: O=offen X=geschlossen

# 5. Steckverbinder



# 6. Speicher

## Speicherinterface

Der Anschluß der Speicherbausteine an den Systembus ist aus Blatt 1 des Schaltplanes (separat beigelegt) ersichtlich. Der Flash-Baustein IC2 (29F400 von AMD bzw. Fujitsu) hat eine Kapazität von 512 KB. Für viele Anwendungsfälle würde auch der kleinere 29F100 mit 128 KB ausreichen. Er ist pinkompatibel zum 29F400, hat nur zwei Adreßleitungen weniger.

Adreß- und Datenleitungen des Flash Memories sind Eins zu Eins mit dem Systembus verschaltet. Die Adreßleitung A0 des Systembusses bleibt frei, da der Flashbaustein im 16 Bit-Modus betrieben wird. Dadurch ist die Numerierung der Adressleitungen am Flashbaustein und am Adreßbus um Eins versetzt.

Der 16 Bit-Modus wird an dem AMD Flash mit dem /BYTE-Pin festgelegt. Es ist permanent mit VCC verbunden, daran erkennt der Flash die Betriebsart "Word-Mode".

Selektiert wird der Flashspeicher über /CE und /OE. Der /CE-Pin wird vom Controller über die Chipselect-Leitung /CSP0 kontrolliert. Diese Chipselect-Leitung des HC12 ist nach Reset im Expanded Mode bereits aktiv, d.h. die Leitung geht auf Low, sobald ein Zugriff im Speicherbereich $8000 bis $FFFF stattfindet und keine höherpriorisierten internen Ressourcen vorgehen.

Die Schreib-/Leserichtung wird über /WE und /OE gesteuert. Das Output-Enable Signal wird aus dem negierten R/W-Signal des Controllers generiert.

Der Ausgang RY/BY kann genutzt werden, um das Ende eines Schreibvorganges zu erkennen. Hierzu ist eine Verbindung zum Port PT3 des Controllers vorbereitet. Schließt man die Lötbrücke BR13 (diese befindet sich, wie alle Lötbrücken, auf der Unterseite der Platine), kann man über den Input Capture Kanal 3 einen Interrupt auslösen, sobald der Flash eine (längerandauernde) Schreib- bzw.

Löschroutine beendet hat. Alternativ zu dieser Hardwarelösung ist es ebenso möglich, den Status des Flash-Bausteins mittels Toggle-Bit Polling herauszufinden (Details zu den Algorithmen siehe AMD Datenbuch). Tatsächlich wird meistens diese Softwaremethode bevorzugt.

In der Schaltung wurde das RAM mit zwei 8 Bit Bausteinen realisiert. IC1 deckt die obere Hälfte des Datenbus ab, IC3 die untere. Da nur ein Chipselektsignal für beide Speicher vorhanden ist, wird dieses (/CSD) zunächst mit der Adreßleitung A0 bzw. /LSTRB verknüpft, um einen der RAMs (oder ggf. beide!) anzusprechen.

Die Verknüpfung erfolgt mit IC15A bzw. IC15B. Ist A0=L wird IC1 selektiert, ist /LSTRB=L wird IC3 selektiert. Bei Bytezugriffen entspricht /LSTRB stets dem negierten A0 Signal. Bei Wortzugriffen sind sowohl A0 als auch /LSTRB Low, die beiden RAMs bedienen dann den ganzen Datenbus mit 16 Bit Breite. Findet der Wortzugriff an einer ungeraden Adresse statt, ist A0=/LSTRB=H. Dieser Sonderfall kann aber nur bei Zugriffen auf das interne RAM auftreten, welches in der Lage ist, Low- und High-byte gegeneinander auszutauschen. Externe Zugriffe finden stets als Aligned Word- oder Byte-Access statt, dafür sorgt die Bussteuerlogik des HC12.

Für die RAMs lassen sich Typen mit 128 KB (Standard) oder 512KB (Option) einsetzen. Sie sind weitgehend pinkompatibel, lediglich eine der RAM-Leitungen muß typspezifisch umgeschaltet werden. Je nach RAM-Typ ermöglicht es JP5, entweder (für 512 KB RAMs) die Adressleitung A17 oder (für 128 KB RAMs mit einem zusätzlichen Chipselektsignal) das Resetsignal an Pin 30 zu führen.

## Speicheradressierung, Bankswitching

Der HC12 verfügt über einen Speicheradressraum von 64 KB. Dadurch ist es möglich, codesparende 2-Byte-Adressen zu verwenden (wie beim HC11). Durch die eingebaute Bankswitching-Hardware des HC812A4 lassen sich darüber hinaus bis zu 4 MB Programmspeicher und bis zu 1 MB Datenspeicher ansteuern.

Die Paging-Hardware wird unterstützt durch Erweiterungen im Befehlssatz. Es stehen spezielle Befehle zum Unterprogrammaufruf und

zur Rückkehr zur Verfügung, die die verwendeten Programmspeicherseiten automatisch wechseln. Die CALL Instruction arbeitet wie ein JSR Befehl, enthält jedoch als zusätzlichen Parameter die gewünschte Pagenummer. Der Wert der bisherigen Programmspeicherseite wird, zusammen mit der Returnadresse, auf den Stack gebracht. Zur Rückkehr bedient man sich dann des Befehls RTC, dem Pendant zu RTS.

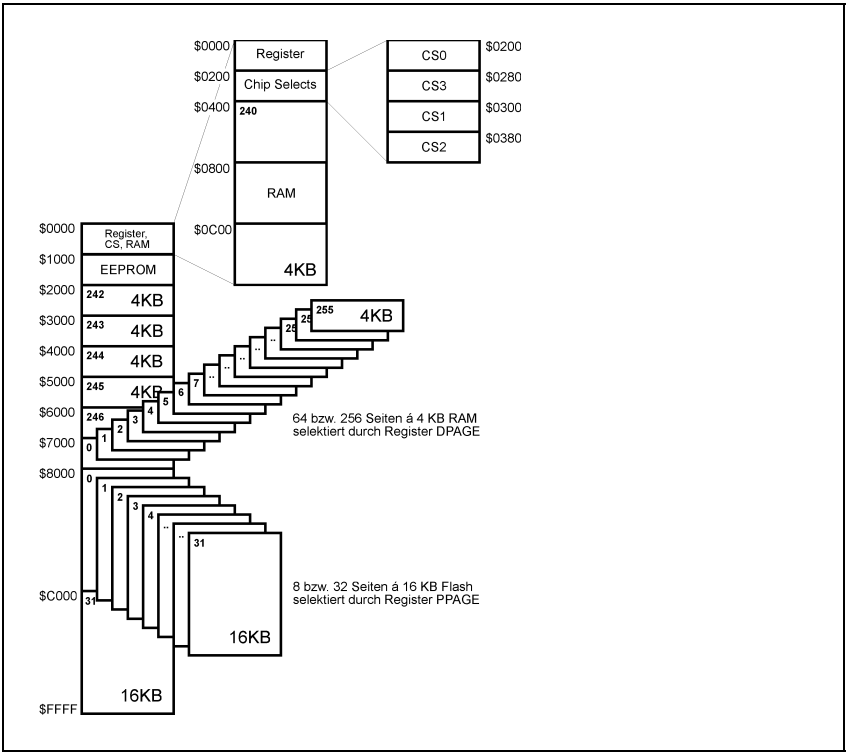
Die Speicherseiten für den Programmspeicher sind 16 KB groß und werden im Adreßfenster $8000 bis $BFFF eingeblendet. Voraussetzung ist natürlich, daß das Paging aktiviert wurde, man muß also zuvor einige Initialisierungen vornehmen.

Die gewünschte Speicherseite wird einfach über das PPAGE Register gewählt. Dieses Register ist 8 Bit breit und erlaubt somit, eine von 256 verschiedenen Speicherseiten zu selektieren.

Für die Datenspeicherseiten wird das Data Window von $7000 bis $7FFF genutzt. Auch hier kann man aus 256 Seiten auswählen, gesteuert über das DPAGE Register. Mit 256 Seiten zu je 4 KB ergibt sich ein maximaler Umfang von einem Megabyte Data Memory. Spezifische Befehle zur Verwaltung des Datenspeichers gibt es, im Gegensatz zum Programmspeicher, nicht. Der Programmierer muß selbst dafür sorgen, im richtigen Moment die richtige Speicherseite einzublenden. Ein großes Problem stellt dies aber nicht dar, schließlich muß nur der entsprechende Wert in das DPAGE Register geschrieben werden.

Mit einem dritten Fenster namens Extra Window lassen sich nochmals 256 Blöcke zu 1 KB selektieren. Inwiefern auch das Extra Window genutzt werden kann/soll, muß man anhand der geplanten Applikation abwägen. Meistens wird man sich wohl eher auf das Paging im Program und/oder Data Window beschränken. Die gesamte Generierung von Chip-Selects und Memory Pages ist beim HC812A4 recht leistungsfähig, aber auch komplex. Für Einzelheiten sei daher auf das Datenbuch "Technical Summary" von Motorola verwiesen.

Die Memory Map des HC12compact Einplatinenrechners im Normal Expanded Wide Mode sieht - nach geeigneter Initialisierung - wie folgt aus:




*Memory Map*

Die obere Hälfte des Speicheradreßraumes ($8000 bis $FFFF) wird als Programmspeicher verwendet, er wird von dem Flash-Memory Baustein bereitgestellt. Der Controller benutzt zunächst die obersten 16 KB des Speicherbausteins (dies ist gleichzeitig Page $FF), um den Adreßbereich von $C000 bis $FFFF damit zu füllen. Darunter ($8000 bis $BFFF) befindet sich das Program Window. In dieses Fenster kann *jede* 16 KB Page des Flash-Bausteins eingeblendet werden. Sinnvoll ist es, an diese Stelle standardmäßig die Page $FE einzublenden, denn dann ist das Program Window mit dem selben Speicherbereich belegt, wie *nach* einem Reset und *vor* der Aktivierung des Program Window.

In der unteren Hälfte des Speicheradreßraumes ($0000 bis $7FFF) ergibt sich folgende Situation: Der Bereich ist grundsätzlich mit 32 KB des externen RAM (256 KB bis 1 MB) belegt. Es gibt jedoch einige

höher priorisierte Ressourcen, die bestimmte externe RAM-Adressen verdecken. Im Bereich von $0000 bis $01FF liegen die Steuerregister des Controllers, danach schließt sich ein Bereich an (bis $03FF), der zur Verwaltung von Chipselects für Peripheriebausteine vorgesehen ist. Zwischen $0400 und $07FF "schaut" wieder ein Stück externes RAM durch. Ab $0800 bis $0BFF liegt das interne RAM (1 KB) und überdeckt wiederum den externen Speicherbereich. Dann ist "Luft" bis $1000, dort beginnt der interne EEPROM-Bereich (4 KB). Ab $2000 bis $6FFF geht es ganz linear mit RAM-Adressen weiter.

Im 4 KB Block von $7000 bis $7FFF liegt das Data Memory Window. Mit 1 MB RAM sind bis zu 256 Pages möglich, mit 256 KB RAM sind es 64 Pages. Bestimmte Seiten (die mit den Nummern [n-16] bis [n-10], wenn [n] die maximal zur Verfügung stehende Anzahl Seiten bezeichnet) sind gleichzeitig im linearen Adreßbereich sichtbar, die Abbildung oben soll dies illustrieren.

## Flash-Programmierung

Im Gegensatz zu einem Lesezyklus, bei dem unter Verwendung der angelegten Adresse auf das gewünschte Wort im Speicher-Array zugegriffen wird, kann man mit einem Schreibzyklus nicht auf den eigentlichen Speicher des Flash Bausteins zugreifen. Statt dessen gelangen die Daten beim Schreiben in das Command Register einer internen State Machine. Nur wenn eine bestimmte, genau vorgeschriebene Sequenz zu dieser State Machine gelangt, löst diese letztendlich den gewünschten Schreibvorgang aus.

Tabelle 1 zeigt den Ablauf einer solchen Schreibsequenz ("Embedded Program Algorithm"). Die ersten drei Schreibzyklen schalten die interne State Machine frei. Erst im vierten Zyklus wird die Programmieradresse und das zu programmierende Datenwort übergeben. Während dieses vier Zyklen umfassenden Ablaufs darf kein zusätzlicher Schreibzugriff auf den Flash Baustein stattfinden, sonst wird der Vorgang durch die State Machine abgewiesen. In diesem Falle müßte man mit Punkt 1 neu beginnen.

| Schreibzyklus | Flash_Adresse | Daten |
|---|---|---|
| 1. | $5555 | $00AA |
| 2. | $2AAA | $0055 |
| 3. | $5555 | $00A0 |
| 4. | Programmieradresse | Datenwort |

*Tab.1: Embedded Program Algorithm für den Am29F400 (Word Mode)*

Das erscheint zunächst kompliziert, dient aber letztlich dazu, die Sicherheit gegen versehentliches Überschreiben von Speicherzellen zu erhöhen. Zudem wird das gesamte Timing für den eigentlichen Programmiervorgang vom Flash Baustein selbständig gebildet. Dadurch hat man als Anwender dieser Embedded Algorithmen bei erträglichem Aufwand ein nur noch geringes Risiko, sich Fehler einzuhandeln.

Bevor man den Ablauf gem. Tabelle 1 als Software implementieren kann, sind noch einige Umrechnungen nötig. Im Schaltplan (siehe Anhang) erkennt man, daß die Adreßleitungen des Flash Bausteins gegenüber dem Adreßbus um eins versetzt sind. Das ergibt sich einfach aus der Beschaffenheit des Speicherinterface: Beim wortweisen Ansprechen des Flash Bausteins ist die Adreßleitung A0 stets Null. Diesem Umstand verdanken wir, daß die von der State Machine erwarteten Adressen gem. Tabelle 1 nicht identisch sind mit denen, die man durch den HC12 aussenden muß. Aus der Sicht der MCU erscheinen alle Adreßwerte verdoppelt, Tabelle 2 zeigt die neue Belegung. Nur die vom Anwender gewünschte Programmieradresse muß nicht korrigiert werden, vorausgesetzt man rechnet in Byte-Schritten und schreibt Worte (Doppelbytes) ausschließlich an gerade Adressen.

| Schreibzyklus | MCU_Adresse | Daten |
|---|---|---|
| 1. | $AAAA | $00AA |
| 2. | $5554 | $0055 |
| 3. | $AAAA | $00A0 |
| 4. | Programmieradresse | Datenwort |

*Tab. 2: Embedded Program Algorithm - aus Sicht der MCU*

Die Notwendigkeit, Schreibzugriffe auf den Flash ausschließlich nur wortweise ausführen zu können, klingt nach einer erheblichen Einschränkung. Eine Anwendung (wie z.B. ein Monitorprogramm) kann diese Klippe jedoch leicht umschiffen, indem die zu schreibende Einzelbytes automatisch zu Worten aufgefüllt werden. Hierzu liest man zunächst das Original-Wort an der betroffenen Adresse ein, ersetzt eine Hälfte durch das zu schreibende Byte und schreibt schließlich das komplette Wort zurück.

Nun könnte man also ein kurzes Test-Programm gemäß Tabelle 2 implementieren und wäre schon in der Lage, Flash-Zellen "in-circuit" zu programmieren - aber nur, wenn der gesamte 64KB Adreßraum der MCU dem Flash-Memory gewidmet wäre! Da dem nicht so ist (vergl. Memory Map), muß man nochmals ein wenig rechnen.

Innerhalb des Program Memory Window im Adreßbereich von $8000 bis $BFFF kann jeder beliebige, 16KB große Abschnitt des Flash Bausteins eingeblendet werden. Diese Abschnitte bzw. Speicherseiten werden durch das PPAGE Register des HC12 selektiert. Die erste Speicherseite des Flash Bausteins (Flash-Adresse $00000 bis $03FFF) erscheint nach Schreiben der Pagenummer $00 ins PPAGE Register somit im Program Memory Window (MCU-Adresse $8000 bis $BFFF).

Zur Bedienung der State Machine muß man diese Rechnung für die beiden Adressen $AAAA und $5554 durchführen. Tabelle 3 zeigt den Embedded Program Algorithm mit den endgültigen Zugriffsadressen.

| Schreib-zyklus | PPAGE-Register | MCU_Adresse | Daten |
|---|---|---|---|
| 1. | $02 | $8000+$2AAA | $00AA |
| 2. | $01 | $8000+$1554 | $0055 |
| 3. | $02 | $8000+$2AAA | $00A0 |
| 4. | Userpage | $8000+Offset | Datenwort |

*Tab. 3: Embedded Program Algorithm im Paged Memory*

Userpage und Offset ergeben zusammen die Zieladresse im Flash Memory. Ein 29F400 hat eine Größe von 512KB, es gibt also 32 physikalische Seiten zu je 16 KB. Die höchste Seite mit der Nummer 31 ist zusätzlich zu einer eventuellen Einblendung in das Program Memory Window stets im oberen Adreßbereich ($C000 bis $FFFF) sichtbar. Will man diese Seite programmieren, stellt man im PPAGE Register Seite 31 ($1F) ein und vermindert alle zu programmierenden Adressen um $4000. Bei allen anderen Seiten ist davon auszugehen, daß die Adreßlage bei Programmierung wie bei Ausführung des Programms identisch ist (innerhalb des Program Memory Window). Die Useradresse setzt sich dann zusammen aus Userpage und Offset (kleiner $4000) plus Anfangsadresse im Program Memory Window ($8000). Als Formel dargestellt würde das in etwa so aussehen:

```
PPAGE := Flash_Adresse DIV $4000
MCU_Adresse := (Flash_Adresse MOD $4000) + $8000
```

Das Listing (s.u.) zeigt die fertige Flash Schreibfunktion für den HC12. Vor Aufruf der Funktion muß das PPAGE Register auf die gewünschte Speicherseite gesetzt werden und das Register Y muß mit der MCU-Adresse (wie in Tabelle 3) geladen sein:

```
;---------------------------------------------
; Func: Write Flash Word
; Args: Y = Destination Address (must be even!)
;       D = Word to write
;---------------------------------------------
;
wrFlashWord     pshx
                pshd
                ldaa PPAGE
                psha
                movb #$02,PPAGE
                movw #$00aa,$aaaa
                movb #$01,PPAGE
                movw #$0055,$9554
                movb #$02,PPAGE
                movw #$00a0,$aaaa
                pula
                staa PPAGE
                puld
                std  0,y
                ; /Data Polling Sequence
                ldx  #1000                  ; 1000 x 8 x 0,125æs
_wrflcploop     cpd  0,y                    ; [3]
                beq  _wrflwdone             ; [1]
                dex                         ; [1]
                bne  _wrflcploop            ; [3]
_wrflwdone      pulx
                rts

;---------------------------------------------
```

*Flash Write Algorithmus*

Die gezeigte Funktion bedient die State Machine des Flashs in der besprochenen Art und Weise und führt nach dem eigentlichen Schreiben einen "Toggle Bit Polling" genannten Algorithmus aus, der dazu dient, das Ende der Schreibzeit (vom Flash vorgegeben) abzuwarten. Wer seine kostbare Rechenzeit nicht für profane Aufgaben wie derlei Polling verschwenden will, kann statt dessen auch einen Interrupt auswerten, den der Flash generiert, sobald er das Schreiben beendet hat. Hardwareseitig muß man dazu lediglich das RY/BY-Signal des Flash Bausteins (IC2) mit dem interruptfähigen Eingang PT3 der MCU verbinden, indem man die Lötbrücke BR13 schließt.

Ergänzend sei noch darauf hingewiesen, daß die hier verwendeten AMD Flash-Bausteine neben dem "Embedded Program Algorithm" noch eine Reihe weiterer Funktionen kennen. Am wichtigsten ist sicherlich der Erase Algorithmus, der zum Löschen einzelner Sektoren oder des gesamten Chips dient. Auch diese Löschoperationen führen die 29Fxxx Bausteine übrigens mit einer einzigen Betriebsspannung von 5V aus.

# 7. RS232-Schnittstelle

Der HC812A4 verfügt über zwei interne asynchrone serielle Schittstellen (SCI0, SCI1). Jede Schnittstelle umfaßt zwei Signale (RXD, TXD). Handshakeleitungen sind ggf. durch Hinzunahme anderer Portpins zu realisieren.

Die beiden Schnittstellen sind voneinander weitgehend unabhängig. Dadurch ist es z.B. möglich, das zweite SCI abzuschalten. Die beiden freiwerdenden Leitungen können nun als Handshakeleitungen für SCI0 verwendet werden (hierzu muß der Benutzer aber entsprechende Software implementieren).

In vorgenanntem Fall, oder aber bei gleichzeitiger Nutzung beider SCI-Kanäle sind die Lötbrücken BR6 und BR7 zu schließen. PS2 bzw. PS3 des HC12 werden dadurch mit einem Receiver- bzw. Transmitterkanal des RS232-Pegelwandlers IC10 verbunden.

Die Firmware für das SCI-Modul ist relativ unproblematisch, wie das folgende Programmierbeispiel beweist:

```
;=============================================================================;
; File: SCI12.A
; Func: Serial-I/O via the Serial Communication Interface (SCI0)
;=============================================================================;

;       CPU 68HC12
;       include "reghc12.inc"


SC0BD   equ SC0BDH
SC0CR   equ SC0CR1


; Func: Initialize SCI (9600 Baud, 8N1, Polling Mode)
; Args: -
; Retn: -
; Dest: D
;
initSCI         ldd   #26               ; 19200 Bd
                std   SC0BD
                ldd   #$000c            ; 8N1, TE + RE
                std   SC0CR             ; A:B -> SC0CR1:SC0CR2
                rts


; Func: Test if any character available (received)
; Args: -
; Retn: A =  0 (Z = 1) -> no char
;       A <> 0 (Z = 0) -> char available
; Dest: -
;
testSCI         ldaa  SC0SR1            ; read status
                anda  #$20              ; receive data reg full?
                rts                     ; returns 0, if no data
```

```
; Func: Get character from SCI (wait for)
; Args: -
; Retn: A = char
; Dest: -
;
getSCI          brclr SC0SR1,$20,getSCI ; receive data reg full?
                ldaa  SC0DRL            ; read out data
                rts


; Func: Send a character via SCI
; Args: A = char
; Retn: -
; Dest: -
;
putSCI          brclr SC0SR1,$80,putSCI ; transmit data reg empty?
                staa  SC0DRL            ; send data
                rts

;===============================================================================;
```

## IF-Module

Der Anschluß von 10-pol. IF-Modulen (serielle Treibermodule bzw. Medieninterfaces versch. Hersteller) ist **nicht unmittelbar** über einen entsprechenden Steckverbinder vorbereitet!

Gleichwohl kann man diese Module am SCI-Kanal 1 betreiben, indem man die Signale RXD1, TXD1, GND und VCC vom Steckverbinder ST4 bezieht. Die Brücken BR6 und BR7 sind in diesem Falle offen zu halten!

# 8. Peripheriebausteine am SPI-Bus

Über die SPI-Schnittstelle des HC12 werden drei Peripheriebausteine auf der Platine angesprochen: Real Time Clock, A/D-Wandler und D/A-Wandler. Die MCU ist stets SPI-Master. Zur Auswahl der Slaves wird je Slave eine Chipselectleitung benötigt.

Die benötigten SPI-Signale (MISO, MOSI, SCK) liegen auf Leitungen des Port S (PS4/5/6) des HC812A4. Zur Slave-Auswahl werden in der vorliegenden Schaltung die Port H-Signale PH4 und PH5 verwendet. Über den Dekoder IC6B lassen sich die folgenden vier Zustände selektieren:

| PH4 | PH5 | Select-Leitung | selektierter Slave |
|---|---|---|---|
| 0 | 0 | /SPICS0 | RTC |
| 1 | 0 | /SPICS1 | DAC |
| 0 | 1 | /SPICS2 | ADC |
| 1 | 1 | alle inaktiv | keiner |

PH4=PH5=H ist der Defaultzustand (kein Slave selektiert).

Beispiele für die Initialisierung des SPI-Moduls des HC12 und einer geeigneten Sende-/Empfangsfunktion zeigt das folgende Listing. Die Initialisierung erwartet keine Argumente, die Transferfunktion erwartet das zu sendende Zeichen in Akku A und gibt das empfangene Zeichen ebenda zurück:

```
;===============================================================================
; File: SPI12.A
; Func: HC12 SPI Routines
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 08.02.98
;===============================================================================

; SPI Slave Chip Select Decoder:
; SPICOD0 SPICOD1
;   PH4     PH5
; -----------------------------
;    0       0     SPICS0    RTC
;    1       0     SPICS1    DAC
;    0       1     SPICS2    ADC
;    1       1     none selected

initSPI         bset PORTH,$30          ; SPICOD0/1 (PH4,5) = H
                bset DDRH,$30           ; PH4,5 Output
```

```
                bset DDRS,$e0           ; /SS,SCK,MOSI Output
                movb #$03,SP0BR         ; SPI Rate = ECLK/16 (500kHz)
                movb #$54,SP0CR1        ; SPE+MSTR+/CPOL+CPHA
                movb #$08,SP0CR2        ; as default
                rts


xferSPI         staa SP0DR
_xfs1           tst  SP0SR
                bpl  _xfs1
                ldaa SP0DR
                rts

;===============================================================================
```

## Real Time Clock

Zusätzlich zu den SPI-Signalen benötigt die Real Time Clock (Typ RTC4553 von Seiko/Epson) ein Schreibsignal. Hierzu wird PH6 des HC812A4 benutzt. Der Defaultzustand ist H (Read). Selektiert wird die RTC mit PH4=PH5=L.

Im Listing sind grundlegende Ein-/Ausgabe-Routinen gezeigt, höhere Funktionen wie z.B. Uhrzeit setzen/lesen sind nicht abgedruckt, jedoch auf der beiliegenden Diskette enthalten. Da die Programmierung der RTC relativ komplex ist, sollte man diese Beispiele, neben dem RTC-Datenblatt, als Vorlage für eigene Routinen benutzen.

```
;===============================================================================
; File: RTC12.A
; Func: HC12 Driver Routines for RTC4553
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 08.02.98
;===============================================================================

; SPICOD0 = PH4 = 0   selects
; SPICOD1 = PH5 = 0   the RTC
; /WRRTC  = PH6

initRTC         bset PORTH,$40          ; /WRRTC (PH6) = H
                bset DDRH,$40           ; PH6 Output
                rts

getRTC          movb #$5d,SP0CR1        ; SPE+MSTR+CPOL+CPHA+LSBF
                bset PORTH,$40          ; /WRRTC = H
                bclr PORTH,$30          ; SPICOD0/1 = L (enable SPICS0)
                bsr  xferSPI            ; send Register Address
                bsr  xferSPI            ; receive Data from RTC
                bset PORTH,$30          ; SPICOD0/1 = H (disable SPI slaves)
                rts

putRTC          movb #$5d,SP0CR1        ; SPE+MSTR+CPOL+CPHA+LSBF
                bclr PORTH,$40          ; /WRRTC = L
                bclr PORTH,$30          ; SPICOD0/1 = L (enable SPICS0)
                psha
                bsr  xferSPI            ; send Register Address + Data
                pula
                bset PORTH,$30          ; SPICOD0/1 = H (disable SPI slaves)
                rts

;===============================================================================
```

Die RTC kann über einen Alarmtimer einen Interrupt auslösen, z.B. um den Prozessor nach einer längeren (untätigen/stromsparenden) "Schlafphase" wieder aufzuwecken. Um dieses Feature zu nutzen, muß JP6 (RTCIRQ) gesteckt werden.

Die RTC wird bei Stromausfall von einer Lithium-Zelle (E1) gepuffert. Der installierte Typ ist auf eine Lebensdauer von mindestens 5 Jahren konzipiert.

## A/D-Wandler

Der A/D-Wandler (Typ TLC2543 von TI) verarbeitet 11 Eingangskanäle mit 12 Bit Auflösung. Selektiert wird der ADC mit PH4=L und PH5=H. Eine separate Initialisierung des ADC ist nicht notwendig, abgesehen von dem Setup des SPI-Interface des HC12 (s.o.). Der Erfassungsroutine ist die Nummer des gewünschten Meßkanals in Akku A zu übergeben. Sie liefert daraufhin den Meßwert (12 Stellen signifikant) im Doppelakku D zurück:

```
;===============================================================================
; File: ADC12.A
; Func: HC12 Driver Routines for TLC2543 11ch 12bit ADC
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 10.02.98
;===============================================================================

; SPICOD0 = PH4 = 0   selects
; SPICOD1 = PH5 = 1   the ADC


initADC         ; nothing to do!
                rts


; Func: Get ADC result
; Args: A=Channel ($00..$0A)
;       Special channels:
;       $0B: VREF/2 ($800)
;       $0C: VREF-  ($000)
;       $0D: VREF+  ($FFF)
;       $0E: enter power save mode
; Retn: D=Value (12 Bit significant)
;
getADC          movb #$50,SP0CR1        ; SPE+MSTR+/CPOL+/CPHA
                bclr PORTH,$10          ; SPICOD0 = L (enable SPICS2)
                lsla
                lsla                    ; move channel no. to
                lsla                    ; upper 4 bits of control word
                lsla
                oraa #$0c               ; lower nibble: DL=16Bits,MSB 1st,Unipolar
                psha                    ; save control word
                jsr  xferSPI            ; send control word
                jsr  xferSPI            ; send bits 8..15 (ignored by ADC)
                bset PORTH,$30          ; SPICOD0/1 = H (disable SPI slaves)
```

```
                ;
                ldaa #100               ; wait for conversion result
_gadcwt         deca                    ; 4 cyc x 125 ns x 100 = 50µs (>10µs)
                bne  _gadcwt
                ;
                bclr PORTH,$10          ; SPICOD0 = L (enable SPICS2)
                pula                    ; restore control word
                jsr  xferSPI            ; get data (high)
                tab
                jsr  xferSPI            ; get data (low)
                exg a,b                 ; order h/l
                lsrd
                lsrd                    ; move result 4 bits to the right
                lsrd
                lsrd
                bset PORTH,$30          ; SPICOD0/1 = H (disable SPI slaves)
                rts

;===============================================================================
```

Der A/D-Wandler arbeitet mit einer externen Spannungsreferenz von 4096 mV (IC11). Ein Schritt entspricht somit 1 mV, der Nullpunkt ist bei 0 mV, Endwert bei 4095 mV. Durch Auftrennen von BR9 und/oder BR8 lassen sich auch andere Meßbereiche einstellen.

Das Ende einer A/D-Umsetzung vermag der ADC auch über eine Interruptleitung mitzuteilen. Soll diese Funktion genutzt werden, ist BR10 zu schließen. Der "End-Of-Conversion" Interrupt kann dann via PT1 des HC12 generiert werden.

## D/A-Wandler

Der D/A-Wandler verfügt über zwei Kanäle mit 12 Bit Auflösung. Zum Einsatz kommt der Typ LTC1454 (Linear Technology).

Auswahl des DAC über PH4=H und PH5=L, Initialisierung entfällt. Die beiden DAC-Kanäle werden stets gleichzeitig aktualisiert, daher muß der Ausgaberoutine ein Feld mit den zwei Ausgabeworten (insgesamt 4 Byte) bereitgestellt werden:

```
;===============================================================================
; File: DAC12.A
; Func: HC12 Driver Routines for LTC1454 2ch 12bit DAC
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 09.02.98
;===============================================================================

; SPICOD0 = PH4 = 1   selects
; SPICOD1 = PH5 = 0   the DAC

initDAC         rts


; Update both DAC channels
; Args: X points to buffer[4] containing new DAC values
;       X[0,1] for DACA, X[2,3] for DACB
;
updateDACx      pshd
                movb #$50,SP0CR1        ; SPE+MSTR+/CPOL+/CPHA
                bclr PORTH,$20          ; SPICOD1 = L (enable SPICS1)
                ldab 0,x
                lslb
                lslb
                lslb
                lslb
                ldaa 1,x
                lsra
                lsra
                lsra
                lsra
                aba
                jsr  xferSPI            ; send DACA highest 8 Bit
                ldab 1,x
                lslb
                lslb
                lslb
                lslb
                ldaa 2,x
                anda #$0f
                aba
                jsr  xferSPI            ; send DACA lowest 4 Bit, DACB highest 4 Bit
                ldaa 3,x
                jsr  xferSPI            ; send DACB lowest 8 Bit
                bset PORTH,$30          ; SPICOD0/1 = H (disable SPI slaves)
                puld                    ; releasing /CS will latch new DAC output!
                rts

;===============================================================================
```

Der DAC hat eine eingebaute Spannungsreferenz (ergibt einen Ausgangspegel zwischen 0 und 4095 mV ). Diese ist über BR4 zurückgeführt auf die Referenzeingangspins der beiden Kanäle. Die untere Referenzspannung liegt über BR3 auf Masse. Nach Auftrennen der beiden Lötbrücken kann man auch andere Ausgangsbereiche erzielen.

# 9. Peripherie am Systembus

Der HC812A4 stellt einige Chipselect-Signale zur Verfügung. Darunter sind die vier Signale /CS0../CS3 (Port F), welche zur Ansteuerung von Peripheriebausteinen vorgesehen sind:

| Signal | Adresse | Funktion |
|---|---|---|
| /CS0 | $0200 | CAN-Controller (Schreib-/Lesezugriff auf das gewählte Register) |
| /CS1 | $0300 | CAN-Controller (Register/Adresse auswählen) |
| /CS2 | $0380 | Ansteuerung LC-Display (alphanum.) bzw. universelles Chipselect (verknüpft mit R/W und E-Clock) |
| /CS3 | $0280 | freies Chipselect |

In der Initialisierungsphase des Cotrollers sind die Chipselects - je nach Erfordernis - zu verlängern. Es ist i.d.R. ratsam, den maximalen Stretchingwert (3 E-Clocks) einzustellen (s.a. Beispiel Startup-Initialisierung). Dadurch ergibt sich eine Buszykluszeit von 1µs bei 16 MHz Quarztakt.

## CAN-Controller

Zur Vernetzung von Steuerungstechnikkomponenten wird der CAN-Bus immer beliebter, und längst nicht nur in der Automobiltechnik.

Das CAN-Protokoll ist mehrschichtig und bzgl. der höheren Schichten in verschiedensten Varianten gebräuchlich. Dank der Implementierung der hardwarenahen Protokollschichten in dem Stand-Alone CAN Controller SJA1000 (kompatibel zum Vorgänger PCA82C200) ist es dennoch verhältnismäßig einfach, eine zuverlässige Kommunikation zwischen den Busteilnehmern zu realisieren. Zu Einzelheiten der Programmierung von CAN-Systemen sei hier aus Gründen der Komplexität auf die Fachliteratur verwiesen. Im folgenden sollen

lediglich die grundlegende Kommunikation mit dem CAN-Baustein gezeigt werden:

```
;===============================================================================
; File: CAN12.A
; Func: HC12 Driver Routines for SJA1000 CAN2.0B Controller
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 14.02.98
;===============================================================================

;-------------------------------------------------------------------------------
; Func: Target dependend hardware access functions
; Rem.: Modify this according to the actual bus interface!
;-------------------------------------------------------------------------------
CAN_ADDR        equ  $0301      ; write to this address to select a register
CAN_DATA        equ  $0383      ; read/write data from/to selected register
;-------------------------------------------------------------------------------


; Func: Read CAN Controller Register
; Args: X = Register Number ($00..$1f)
; Retn: A = Data
;
readSJA1000     stx  CAN_ADDR-1
                ldaa CAN_DATA
                rts


; Func: Write CAN Controller Register
; Args: X = Register Number ($00..$1f)
;       A = Data
;
writeSJA1000    stx  CAN_ADDR-1
                staa CAN_DATA
                rts

;===============================================================================
```

Der eingesetzte CAN-Controller von Philips ist nicht unmittelbar für den Anschluß an den Systembus des HC12 ausgelegt. Als Interfacehardware sind zusätzlich die Dekoder IC5B und IC6A erforderlich. Ein Zugriff wird stets in zei Phasen durchgeführt: Zunächst wird über CAN_ADDR das gewünschte Register (Registeradressen siehe Datenblatt zum SJA1000) selektiert. Danach kann lesend bzw. schreibend auf den Inhalt dieses Registers zugegriffen werden. Die Zugriffsadresse ist stets CAN_DATA.

Der CAN-Controller kann über seinen Ausgang /INT Systeminterrupts auslösen. Um dieses Feature zu benutzen, muß Lötbrücke BR6 geschlossen werden. Das Interruptsignal kann dann über PT0 des HC12 ausgewertet werden.

Zwischen CAN-Controller und Busleitung ist noch ein Medieninterface zu schalten. Der Baustein PCA82C250 (ebenfalls Philips) ist ein

gängiger Vertreter dieser Gattung. Fertige Interfacemodule mit diesem Baustein stellen wir Ihnen auf Anfrage gern bereit.

## LCD-Anschluß

Für alphanumerische LC-Displays hat sich ein Quasi-Standard auf Grundlage des Displaycontrollers HD44780 (Hitachi) etabliert. Es gibt viele Hersteller solcher Module, z.B. Philips, Seiko, Toshiba, Batron, Picvue, Optrex, Sharp usw. Die meisten dieser Module verfügen über einen 14-poligen Anschluß nach gleichem Schema. In der Tabelle ist die Zuordnung zu den Signalen auf dem Controllermodul eingetragen:

| LCD Pin | LCD Signal | Funktion | HC12compact Signal | ST5 Pin |
|---|---|---|---|---|
| 1 | VSS | Masse | GND | 19/20 |
| 2 | VDD | +5V | VCC | 17/18 |
| 3 | VEE | Kontrastspannung | ./. | ./. |
| 4 | RS | Register Select | A1 | 32 |
| 5 | R/W | Read/Write | R/W | 21 |
| 6 | E | Enable | ENA_LCD | 38 |
| 7 | D0 | Datenbit 0 | DB0 | 16 |
| 8 | D1 | Datenbit 1 | DB1 | 15 |
| 9 | D2 | Datenbit 2 | DB2 | 14 |
| 10 | D3 | Datenbit 3 | DB3 | 13 |
| 11 | D4 | Datenbit 4 | DB4 | 12 |
| 12 | D5 | Datenbit 5 | DB5 | 11 |
| 13 | D6 | Datenbit 6 | DB6 | 10 |
| 14 | D7 | Datenbit 7 | DB7 | 9 |

Es folgt ein Ausschnitt aus den LCD-Treiberroutinen (siehe Diskette). Die Routinen einen vom Hauptprogramm bereitzustellenden Puffer (zwei Byte) namens PosXY. Im Header (vergl. Beispiel) muß zudem

definiert sein, über wieviel Zeilen und Zeichen/Zeile das Display verfügt. Gängig sind Typen mit 2x16, 2x20, 2x40 und 4x20 Zeichen.

In der Initialisierungssequenz wird die Betriebsart des LC-Displays eingestellt.

```
;===========================================================================
; File: ALCD12.A
; Func: HC12 Driver Routines for alphanum. LC-Displays
;       (HD44780 based, from 1x8 up to 2x40/4x20 characters)
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.2
; Date: 07.02.98
;===========================================================================
;-- defines ----------------------------------------------------

PosX            equ  PosXY+0    ; RAM location PosXY must be
PosY            equ  PosXY+1    ; defined in the main program!

;---------------------------------------------------------------
; Func: LCD definitions
; Rem.: Modify this according to LCD type used!
;---------------------------------------------------------------

MAX_X           equ  40         ; number of characters per line (8..40)
MAX_Y           equ  2          ; number of lines (1,2 or 4)

RowAddrTable    dc.b $00        ; DDRAM address for 1st line
                dc.b $40        ; DDRAM address for 2nd line
                dc.b $10        ; DDRAM address for 3rd line
                dc.b $50        ; DDRAM address for 4th line

;---------------------------------------------------------------
; Func: Target dependend hardware access functions
; Rem.: Modify this according to the actual bus interface!
;---------------------------------------------------------------

LCD_CMD         equ  $0381
LCD_DATA        equ  $0383

;-- write a command byte ---------------------------------------

_writeCmd       brset LCD_CMD,$80,_writeCmd
                nop
                staa LCD_CMD
                rts

;-- write a data byte ------------------------------------------

_writeData      brset LCD_CMD,$80,_writeData
                nop
                staa LCD_DATA
                rts

;---------------------------------------------------------------
; Func: Initialize LCD
; Args: -
; Retn: -
;---------------------------------------------------------------
initLCD         psha
                pshy
                ldy  #_iniTableStart
_iniCmdLoop     ldaa 0,y
                bsr  _writeCmd
                iny
                cpy  #_iniTableEnd
                bne  _iniCmdLoop
                clr  PosX
                clr  PosY
                puly
                pula
```

```
                rts

_iniTableStart  dc.b $38        ; Function Set (8Bit,5x7)
                dc.b $38        ; dito
                dc.b $0C        ; Display on, Cursor off, Blink off
                dc.b $01        ; Clear display
                dc.b $06        ; Entry Set: Increment, No shift
                dc.b $80        ; Reset Address counter
_iniTableEnd
;===============================================================================
```

# 10. Power Management

Bei der Auswahl der Bauelemente für die Controllerbaugruppe HC12compact wurden solche Typen bevorzugt, die eine Suspend- bzw. Power-Down Funktion haben. Dadurch können die einzelnen Funktionsbaugruppen innerhalb wie außerhalb des Controllers gezielt ein- bzw. ausgeschaltet werden. Je nachdem, auf welche Funktionen verzichtet werden kann, läßt sich die Baugruppe mit sehr geringem Strombedarf betreiben. Folgende Möglichkeiten stehen zur Verfügung:

- Abschaltung HC12-interner Module (z.B. ADC, Timer)
- Abschaltung des externen Businterface (Programm läuft in internen Speicherbereichen des Controllers)
- Optional: Einsatz der PLL des Controllers
- Nutzung der Power-Down Modi von ADC, DAC, CAN-Controller

Eine Batteriepufferung der RAM-Bausteine ist in Hinblick auf die vielfältigen Möglichkeiten zur Reduzierung der Gesamtstromaufnahme nicht vorgesehen.

# 11. Anwendungshinweise

In diesem Abschnitt sollen einige Besonderheiten der Programmierung des HC12 im allgemeinen und der HC12compact Baugruppe im besonderen genannt werden.

Dieses Hardwarehandbuch kann nur einige spezifische Hinweise geben. Die Behandlung allgemeiner Techniken zur Programmierung des Controllers in Assembler bzw. Hochsprachen würden Umfang und Ziel dieses Handbuchs sprengen. Die meisten Antworten finden Sie beim (leider unerläßlichen) Studium der Daten- und Referenzhandbüchern der Halbleiterhersteller.

## Controller-Betriebsarten

Die Startbetriebsart des Controllers wird über die drei Leitungen MODA, MODB und BKGD festgelegt. Folgende Tabelle zeigt alle acht denkbaren Kombinationen (s.a. Jumperbelegung). Drei der genannten Modi sind dabei besonders interessant: Normal Expanded Wide Mode, Normal Single Chip Mode und Special Single Chip Mode.

| MODA | MODB | BKGD | Startbetriebsart | Erläuterung |
|---|---|---|---|---|
| 0 | 0 | 0 | Special Single Chip | Background Debugging sofort aktiv, kein externes Businterface |
| 1 | 0 | 0 | Special Expanded Narrow | Wie Normal Exp. Narrow Mode, einige Schutzmaßnahmen sind jedoch deaktiviert |
| 0 | 1 | 0 | Special Peripheral | Testbetriebsart - CPU disabled |
| 1 | 1 | 0 | Special Expanded Wide | Wie Normal Exp. Wide Mode, einige Schutzmaßnahmen sind jedoch deaktiviert |
| 0 | 0 | 1 | Normal Single Chip | kein externes Businterface, interner EEPROM als Programmspeicher gemappt |
| 1 | 0 | 1 | Normal Expanded Narrow | Externer 8 Bit Datenbus aktiv, langsamer als 16 Bit Zugriff |
| 0 | 1 | 1 | reserviert | |
| 1 | 1 | 1 | Normal Expanded Wide | Externer 16 Bit Datenbus aktiv, volle Geschwindigkeit möglich |

*Betriebsarten vs MODE-Pins*

Für die Entwicklungsphase ist besonders der Normal Single Chip Mode geeignet. Der HC812A4 blendet in dieser Betriebsart seinen internen EEPROM im Bereich $F000 bis $FFFF ein. Da bei der Fertigung des HC12compact stets der TwinPEEKs Monitor im EEPROM abgelegt wird, kann der Anwender sofort nach dem ersten Einschalten mit diesem Monitorprogramm arbeiten. Im Normal Single Chip Mode ist das externe Businterface zunächst disabled. Der Monitor schaltet aber automatisch per Softwarebefehl in den Normal Expanded Wide Mode, wodurch die externen Speicher- und Peripheriebausteine zugänglich werden.

Verfügt man über ein Background Debug Interface, erfolgt der Start i.d.R. im Special Single Chip Mode. Nur in dieser Betriebsart ist der Background Debug Mode, die Nabelschnur des Controllers, vom Start weg aktiv - die CPU versucht also gar nicht erst, ein Anwenderprogramm zu starten. Das ist zu Beginn sehr sinnvoll, denn wo kein User-Code ist, kann man schwerlich ein Programm abarbeiten.

Ein Irrtum wäre es, den Jumper für das BKGD Signal auf Low umzustellen, um den Special Single Chip Mode einzustellen. Belassen Sie diesen Jumper auf High, denn das BDM-Pod (also das Interface zwischen Host-PC und Targetcontroller) wird sich um dieses Signal selbst kümmern! Jedes BDM-Pod ist mit der BKGD-Leitung des Targetcontrollers verbunden, denn über diese Leitung findet der Datenaustausch der Background Debug Schnittstelle statt. Daher ist das Pod auch in der Lage, dafür zu sorgen, daß während und kurz nach Reset dieses Signal auf Low liegt. Die MCU startet dann im Special Mode und das Pod kann die BKGD-Leitung anschließend wieder auf den Defaultzustand (High) zurückbringen, denn das ist die Voraussetzung für die BDM-Kommunikation. Anders gesagt: Legte man BKGD mit Jumper JP3 permanent auf Low, würde man jeglichen Zugriff über die BDM Schnittstelle verhindern.

Der Normal Expanded Wide Mode ist spätestens dann von Interesse, wenn man die Programmentwicklung abgeschlossen hat. In dieser Betriebsart startet der Controller sofort mit externem Businterface. Der Datenbus ist volle 16 Bit breit und die Chipselectleitung für den Programmspeicher (CSP0) ist aktiv. Das Anwenderprogramm residiert

im Flash-Memory und der interne EEPROM ist in niederen Adreßgefilden zu finden, nämlich ab $1000. Um vernünftig mit dem HC12compact arbeiten zu können, sind aber auch im Normal Expanded Wide Mode noch etliche Initialisierungen in das Anwenderprogramm einzubauen.

## Power-Up Initialisierung

Sobald die Resetleitung des Controllers freigegeben wird, holt sich der HC12 die Information, wo das Programm des Anwenders beginnt. Der Controller liest hierzu den Resetvektor von den Speicherzellen $FFFE und $FFFF. Unmittelbar danach springt er an die dort eingetragene Startadresse in der Hoffnung, der User möge etwas sinnvolles programmiert haben.

Jedes Programm beginnt mit einer Reihe von Initialisierungen, so verhält es sich auch beim HC12. Die im Listing (s.u.) gezeigte Initialisierungssequenz zeigt beispielhaft den Programmstart im Expanded Wide Mode auf dem HC12compact:

```
main  lds  #StackPtr            ; Init Stack Pointer
      ; REM: MCU starts in Normal Expanded Wide Mode
      clr  COPCTL               ; Disable Watchdog
      movb #$0c,PEAR            ; LSTRE+RDWE, ECLK
      movb #$3F,CSCTL0          ; Enable CSP0+CSD+CS3+CS2+CS1+CS0
      movb #$10,CSCTL1          ; CSD covers $0000-$7fff
      movb #$30,CSSTR0          ; CSP0+CSD not stretched (CSP1:3x)
      movb #$FF,CSSTR1          ; CS0..3 stretched (3x)
      movb #$fe,PPAGE           ; Select Program Page $FE
      movb #$0f,MXAR            ; A16E...A19E
      movb #$c0,WINDEF          ; DWEN+PWEN
      movb #$80,DDRE            ; PE7=Out (LED driver)
      movb #$00,PORTE           ; LED on
```

Die Initialisierung beginnt mit dem Setzen des Stackpointers. Diese Aktion wird häufig "vergessen".

Überaus wichtig ist die nächste Programmzeile, denn hier wird der Watchdog deaktiviert. Geschieht dies nicht innerhalb von ca. 1048 ms ab Reset, so wird das System zurückgesetzt. Das Ergebnis ist dann ein periodisches Signal auf der Resetleitung - nur leider funktioniert das eigentliche Programm nicht. Eine Sekunde ist für einen Mikrocontroller eine Menge Zeit, trotzdem sollte man sich gleich zu Beginn der Initialisierung um den Watchdog kümmern. Vorsicht ist auch deshalb angeraten, weil Motorola zukünftige HC12-Derivate mit einer viel kürzeren

Watchdog-Resetrate starten läßt. Nur in den Special Modi kann man sich damit Zeit lassen - hier ist der Watchdog nach Reset zusätzlich über das DISR Bit im COPCTL Register disabled.

Das MODE Register enthält nach einem Reset im Normal Expanded Wide Mode den Wert $F0. Folgende Tabelle zeigt die Belegung dieses Registers:

| Bit | Name | Funktion |
|---|---|---|
| 7 | SMODN | entspr. BKGD nach Reset |
| 6 | MODB | entspr. MODB nach Reset |
| 5 | MODA | entspr. MODA nach Reset |
| 4 | ESTR | E-Clock gestreckt |
| 3 | IVIS | Interne Zugriffe extern sichtbar |
| 2 | 0 | stets 0 |
| 1 | EMD | Port D emulation |
| 0 | EME | Port E Emulation |

*Belegung des MODE Registers*

Die oberen drei Bit repräsentieren die Betriebsart. Gleichzeitig wird in diesem Register das Verhalten des E-Clock Bussignals und einiger Settings mit untergeordneter Bedeutung eingestellt.

Weiter im Programm: Nun wird das Port E Assignment Register (PEAR) initialisiert. Port E führt einige wichtige Signale für den externen Systembus. Die Konfigurationsbits NECLK, LSTRE und RDWE konfigurieren das Bus- (Speicher-) Interface. NECLK muß auf Null gesetzt werden, damit der Systemtakt (E-Clock) auf Port E4 ausgegeben wird. Das Signal LSTR wird - zusätzlich zu A0 - für die Ansteuerung der 8 Bit breiten RAMs benötigt, daher muß LSTRE auf Eins gesetzt werden. Last but not least ist auch das Read/Write-Signal für die angeschlossenen Busteilnehmer überaus "hilfreich". RDWE gleich Eins sorgt für die Ausgabe des gewünschten Signals.

Die folgenden Zeilen dienen zur Festlegung der zugeordneten Adreßbereiche und des Timings für die diversen Chip Select Signale. Für eigene Anwendungen sollte man die gezeigten Einstellungen Eins zu Eins übernehmen, soweit man nicht spezielle Pläne hat.

Das PPAGE Register hat einen Defaultwert von $00. Diesen Wert sollte man auf $FE ändern, damit nach Einschalten des Paging der gleiche Speicherbereich wie zuvor (im linearen Adreßmodus) eingeblendet wird.

In der darauffolgenden Zeile wird im Memory Expansion Assignment Register (MXAR) festgelegt, in welchem Umfang Leitungen des Port G als zusätzliche Adreßleitungen verwendet werden sollen. In unserer Schaltung werden die Adreßleitungen A16 bis A19 benutzt, somit verbleiben die oberen zwei Bits des (nur 6 Bit breiten) Port G als Universal-I/O's.

Schließlich wird durch das Setzen von zwei Bits im WINDEF Register das Paging für Programm- und Datenspeicher aktiviert.

An Port E7 ist ein Gatter angeschlossen, welches seinerseits eine LED treibt. PE7 wird als Ausgang konfiguriert und die angeschossene LED eingeschaltet. Damit endet das Initialisierungsbeispiel.

## Schaltplan

Damit alle Details gut lesbar bleiben, liegt der Schaltplan (zwei Seiten A4) separat bei.

## Zusatzinformationen im Web

Beim Vorliegen zusätzlicher bzw. aktualisierter Informationen zur Hard- und Software der Baugruppe, veröffentlichen wir diese auf unserer Website:

http://elmicro.com/de/hc12compact.html

# 12. Monitorprogramm TwinPEEKs

Softwareversion 1.3d

## Allgemeine Hinweise

Das Monitorprogramm TwinPEEKs dient - neben dem Betrachten und Modifizieren von Speicherbereichen - in erster Linie zum Laden und Ausführen von Anwenderprogrammen innerhalb der Testphase.

Für TwinPEEKs ist der **komplette interne EEPROM**-Bereich des HC812A4 **reserviert**. Zudem benötigt TwinPEEKs etwa 512 Byte RAM. Das Anwenderprogramm wird i.d.R. in einem externen Speicherbereichen (RAM oder Flash) abgelegt. Einzelheiten zur Speicheraufteilung siehe Abschnitt "Memory Map".

Der HC12compact wird für den Start mit TwinPEEKs auf "**Normal Single Chip Mode**" eingestellt (siehe Jumperbelegung). Die Umschaltung in den Expanded Mode geschieht per Software im Laufe der Initialisierungssequenz der Monitorfirmware.

Die Kommunikation mit dem Host-PC erfolgt über die erste serielle Schnittstelle (SCI0) mit 19200 Baud, das Übertragungsprotokoll ist 8N1 (keine weiteren Protokolleinstellungen notwendig). Der HC12compact wird dazu mit 16 MHz Quarztakt betrieben.

Als Monitorprompt wird die aktuell gültige Program Memory Page (also der Inhalt des PPAGE Registers) ausgegeben.

## Notation

Alle Zahlenangaben erfolgen hexadezimal ohne weitere Zusätze. Groß-/Kleinschreibung spielt keine Rolle.

Ein Monitorkommando besteht aus einem einzelnen Buchstaben, ggf. gefolgt von einer Liste von Argumenten.

Die Argumenteliste enthält bis zu sechs Adreßargumente, die voneinander durch Leerzeichen oder Komma getrennt sind.

Der Adreßraum umfaßt 64KB, die Adressen sind demzufolge maximal 4-stellig hexadezimal. Beim HC12 beinhaltet dies (wie schon beim HC11) auch die Adreßbereiche der Ports und Steuerregister.

### Wichtig:

**Endadressen beziehen sich stets auf das dem Adreßbereich folgende (nicht enthaltene) Byte. Der Befehl "D 1000 1200" zeigt so z.B. den Adreßbereich von $1000 bis inkl.. $11FF an.**

Eingaben des Benutzers werden über einen Zeilenpuffer abgewikkelt. Gültige ASCII-Zeichencodes liegen im Bereich $20 bis $7E. Mittels Backspace ($08) kann das Zeichen links des Cursors gelöscht werden. Die <ENTER> Taste ($0A) schließt die Eingabe ab.

## Schreibzugriffe

### Flash Memory

TwinPEEKs kann auch auf Flash-Bereiche schreiben. Die Speicherzelle wird zuvor jedoch **nicht automatisch gelöscht**, da der Flash-Memory blockweise (nicht byteweise) gelöscht werden muß. Zum blockweisen Löschen wurde das Monitorkommando X implementiert.

Obwohl der Flash-Baustein wortweise (16 Bit) organisiert ist, kann byteweise geschrieben werden. Der Monitor ergänzt Byte-Daten entsprechend den Erfordernissen des Flash-Bausteins automatisch zu Informationen in Wortbreite.

Im Bereich $F000 bis $FFFF überdeckt der interne EEPROM (enthält das Monitorprogramm) den externen Flashspeicherbereich. Schreibzugriffe auf den internen EEPROM sind weder sinnvoll noch möglich, da der EEPROM schreibgeschützt ist, sobald nach einem Reset die Monitorinitialisierung durchlaufen ist. TwinPEEKs erkennt, wenn ein Schreibzugriff auf ein Byte im Bereich $F000 bis $FFFF

erfolgen soll, und programmiert statt des internen EEPROM den externen Flashspeicher.

Dadurch ist es möglich, ein Anwenderprogramm inkl. Reset- und Interruptvektoren mittels des Monitors in den Flash-Speicher zu laden. Danach muß man nur noch die Betriebsart auf Normal Expanded Wide Mode umstellen, um das Anwenderprogramm "in Echtzeit" zu starten. Schaltet man danach zurück auf Normal Singlechip Mode, startet wieder TwinPEEKs.

Lassen Sie sich aber nicht verwirren: Wenn Sie sich den Speicherbereich $F000 bis $FFFF z.B. mittels Memory Dump anschauen, sehen Sie den Monitorcode in dem für die CPU momentan tatsächlich "sichtbaren" EEPROM, nicht den (verdeckten) Flash-Inhalt! Sie können diesen Bereich im Flash dennoch kontrollieren, indem Sie die Program Page $FF aktivieren (Monitor Befehl "P"). Sie finden dann den Adreßbereich $F000 bis $FFFF innerhalb des Program Window von $B000 bis $BFFF wieder.

### Ports und Steuerregister

Nicht alle Ports bzw. Steuerregister sind gleichermaßen lesbar *und* schreibbar. Ein erfolgreich verlaufener Schreibzugriff kann daher dennoch zu einer Fehlermeldung beim Kontrolllesen führen. Ursache dafür kann sein, daß evtl. von dem betroffenen Register kein Rücklesen möglich ist, bzw. die Belegung des Registers für Lese- und Schreibzugriffe unterschiedlich ist. Ein Beispiel dafür sind Interruptflags des HC12, die meistens dadurch gelöscht werden, indem ein Eins-Bit geschrieben wird.

## Autostart-Funktion

Nach Reset und einigen grundlegenden Initialisierungen stellt das Monitorprogramm fest, ob die MCU-Signale PAD0 und PH1 miteinander verbunden sind. Ist dies der Fall, wird nicht das Monitorprogramm weiter abgearbeitet, sondern ein Sprung zur Adresse **$8000** ausgeführt. Durch diese Funktion ist der Autostart eines bei $8000 beginnenden Anwenderprogramms möglich.

Um die Autostart-Funktion zu aktivieren, müssen lediglich die benachbarten Pins 45 (PH1) und 47 (PAD0) des Steckverbinders ST5 mittels Jumper verbunden werden.

## Interruptvektoren

Die Interruptvektoren des HC12 liegen am Ende des 64 KB umfassenden Adreßraumes, d.h. innerhalb des schreibgeschützten Monitorcodes. Um dennoch Interruptfunktionen in einem Anwenderprogramm zu ermöglichen, leitet der Monitor alle Interruptvektoren (außer den Resetvektor) auf Adressen im internen RAM um. Das Verfahren entspricht der Vorgehensweise des HC11 im Special Bootstrap Mode.

Das Anwenderprogramm setzt den benötigten Interruptvektor, indem es einen Sprungbefehl in den RAM-Pseudovektor einträgt. Um z.B. den SPI-Interrupt nutzen zu können, muß ein Anwenderprogramm folgende Schritte ausführen:

```
ldaa #$06     ; JMP Opcode
staa $0BC7    ; SPI Pseudo Vector
ldd  #isrFunc ; Jump Address
std  $0BC8    ; SPI Pseudo Vector + 1
```

Der folgende Ausschnitt aus dem Assemblerlisting des Monitors dokumentiert die Adressen der umgeleiteten Interruptvektoren:

```
FFCE : 0B B8    dc.w  RAMTOP-72         ; KWUH
FFD0 : 0B BB    dc.w  RAMTOP-69         ; KWUJ
FFD2 : 0B BE    dc.w  RAMTOP-66         ; ATD
FFD4 : 0B C1    dc.w  RAMTOP-63         ; SCI1
FFD6 : 0B C4    dc.w  RAMTOP-60         ; SCI0
FFD8 : 0B C7    dc.w  RAMTOP-57         ; SPI
FFDA : 0B CA    dc.w  RAMTOP-54         ; Pulse Accu Input Edge
FFDC : 0B CD    dc.w  RAMTOP-51         ; Pulse Accu Overflow
FFDE : 0B D0    dc.w  RAMTOP-48         ; Timer Overflow
FFE0 : 0B D3    dc.w  RAMTOP-45         ; TC7
FFE2 : 0B D6    dc.w  RAMTOP-42         ; TC6
FFE4 : 0B D9    dc.w  RAMTOP-39         ; TC5
FFE6 : 0B DC    dc.w  RAMTOP-36         ; TC4
FFE8 : 0B DF    dc.w  RAMTOP-33         ; TC3
FFEA : 0B E2    dc.w  RAMTOP-30         ; TC2
FFEC : 0B E5    dc.w  RAMTOP-27         ; TC1
FFEE : 0B E8    dc.w  RAMTOP-24         ; TC0
FFF0 : 0B EB    dc.w  RAMTOP-21         ; RTI
FFF2 : 0B EE    dc.w  RAMTOP-18         ; IRQ / KWUD
FFF4 : 0B F1    dc.w  RAMTOP-15         ; XIRQ
FFF6 : 0B F4    dc.w  RAMTOP-12         ; SWI
FFF8 : 0B F7    dc.w  RAMTOP-9          ; Illegal Opcode
FFFA : 0B FA    dc.w  RAMTOP-6          ; COP Fail
FFFC : 0B FD    dc.w  RAMTOP-3          ; Clock Monitor Fail
FFFE : F8 00    dc.w  main              ; Reset
```

## Memory Map

Die folgende Tabelle gibt Auskunft über die Speicherbelegung des HC12compact bei Verwendung des TwinPEEKs Monitorprogramms (nicht dargestellt sind externe Speicherbereiche):

| Start | Ende | Speicher | Belegung |
|---|---|---|---|
| $0800 | $0A00 | int. RAM | Frei für Anwenderprogramm |
| $0A00 | $0Axx | int. RAM | Monitor Variablen |
| $0Axx | $0BB8 | int. RAM | Monitor Stack |
| $0BB8 | $0C00 | int. RAM | Umgeleitete Interruptvektoren |
| $F000 | $0000 | int. EEPROM | Monitor Code |

# Monitorkommandos

## Analog/Digital Converter

### Syntax: A

Führt Messungen auf allen Kanälen des A/D-Wandlers durch und zeigt die Ergebnisse fortlaufend an. Abbruch mit beliebiger Taste.

Neben den eigentlichen A/D-Kanälen 0 bis 10 werden auch die internen Referenzkanäle 11 (Vref/2), 12 (Vref) und 13 (GND) angezeigt. Der Wertebereich ist jeweils $0000 bis $0FFF (12 Bit). $0000 entspricht 0V, $0FFF entspricht 4,095V.

Der Befehl erfordert keine Argumente.

| | |
|---|---|
| A | Zeigt fortlaufend alle ADC Kanäle an |

## Dump Memory

### Syntax: D (<AADR> (<EADR>))

Zeigt den Speicherinhalt an. In jeder Bildschirmzeile werden 16 Bytes in hexadezimaler Form dargestellt. Hinzu kommt die Darstellung der ASCII-Interpretation dieser 16 Zeichen. Zeichen außerhalb des druckbaren Bereiches ($20 bis $7E) werden durch einen Punkt ersetzt.

Für den Befehl sind ein, zwei oder kein Argument zulässig, z.B.:

| | |
|---|---|
| D F000 F800 | Zeigt den Speicherbereich von $F000 bis $F7FF an |
| D F000 | Zeigt den Speicherbereich ab $F000 an. Es werden $0040 Bytes angezeigt. |
| D | Zeigt *die nächsten* $0040 Bytes an (ausgehend von der aktuellen Adresse) |

## Edit Memory

### Syntax: E (<AADR>)

Ermöglicht das Ändern von Speicherstellen, beginnend bei der angegebenen Adresse. Für den Befehl sind ein oder kein Argument zulässig, z.B.:

| | |
|---|---|
| E F000 | Edit Memory ab $F000 |
| E | Edit Memory (ausgehend von der zuletzt erreichten Endadresse) |

Der Monitor zeigt daraufhin die aktuelle Belegung der jeweiligen Adresse an und erwartet als Eingabe einen neuen Wert. Statt Eingabe eines neuen Wertes kann auch einer der folgenden Befehle eingegeben werden:

| | | |
|---|---|---|
| | <ENTER> | Zeigt die nächste Speicherstelle an. |
| | - | Zeigt die vorhergehende Speicherstelle an. |
| | = | Zeigt die aktuelle Speicherstelle erneut an. |
| | . | Beendet den Edit Mode |
| | Q | Ende (wie zuvor) |

Wurde durch den Benutzer ein neuer Wert eingegeben, so schreibt der Monitor diesen Wert (ggf. unter Berücksichtigung des innerhalb des Flash-Bereiches anzuwendenden Programmieralgorithmus) in den Speicher und zeigt daraufhin die nächste Speicherstelle an..

## Fill Memory

**Syntax: F <AADR> <EADR> <BY>**

Füllt einen Speicherbereich mit dem angegebenen Bytewert. Für den Befehl sind drei Argumente erforderlich, z.B.:

| | |
|---|---|
| F F000 F800 FF | Füllt den Speicherbereich von $F000 bis $F7FF mit dem Wert $FF |

Wenn sich ein Schreibfehler ereignet, wird die Funktion an der augenblicklichen Adresse abgebrochen.

## Goto Address

**Syntax: G (<AADR>)**

Führt einen Unterprogramm-Sprung (mittels JSR) zur angegebenen Adresse aus. Der Monitor gibt eine Meldung aus, an welcher Adresse die Programmausführung fortgesetzt wird. Das Anwenderprogramm kann mittels RTS zum Monitor zurückkehren.

Für den Befehl ist ein Argument zulässig (oder Eingabe ohne Argumente) z.B.:

| | |
|---|---|
| G F100 | Ruft das Unterprogramm bei Adresse $F100 auf |
| G | Führt das Unterprogramm an der aktuellen Adresse aus |

Um die Ausführung des Monitorprogramms fortsetzen zu können, darf das Anwenderprogramm keine Störungen verursachen. Z.B. sollte das Anwenderprogramm den Stack korrekt wiederherstellen, bevor es die Kontrolle an den Monitor zurückgibt. Auch die serielle Schnittstelle SCI0 darf nicht neu initialisiert werden, wenn der Monitor unbeeinflußt weiterarbeiten soll. Darüber hinaus sollte das Anwenderprogramm alle

Interruptquellen, die es freigibt, vor Beendigung auch wieder sperren. Eine globale Sperrung der Interrupts (mittels SEI) darf nicht erfolgen, da der interruptgesteuerte serielle Zeichenempfang des Monitors dadurch unterbrochen wird.

## Help

### Syntax: H

Gibt einen kurzen Hilfe-Text aus. Der Befehl erfordert keine Argumente:

| | |
|---|---|
| H | Zeigt die verfügbaren Kommandos an |

## Flash-ID

### Syntax: I

Zeigt die Device-ID des Flash Bausteins an. Die Device-ID für den Vorzugstyp 29F400BT lautet 2223 (siehe AMD Datenblatt). Der Befehl erfordert keine Argumente:

| | |
|---|---|
| I | Zeigt die Device-ID des Flash Bausteins an |

## Load S-Records

### Syntax: L (<OFFS>)

Lädt Daten im S-Record Format in den Speicher. Die erforderlichen Speicheradressen sind in der S-Record Datei vermerkt, können aber durch Angabe eines Offsets verschoben werden.

Für den Befehl ist maximal ein Argument (Offsetangabe) zulässig, beispielsweise:

| | |
|---|---|
| L | Lädt eine S-Record Datei in den Speicher |
| L E000 | Lädt eine S-Record-Datei und verschiebt die enthaltenen Adreßangaben um den Offset $E000 (z.B. werden Daten aus der S-Record Datei im Bereich $1000 bis $10FF in den Speicher von $F000 bis $F0FF geladen) |

Das PC-seitig verwendete Terminalprogramm muß es dem Benutzer ermöglichen, nach Ausgabe der Meldung "Loading..." des Monitors, die gewünschte S-Record Datei zu senden. Diese Funktion wird in der Terminalsoftware oft als "Text-" bzw. "ASCII-Upload" bezeichnet.

Die gesendeten Zeichen werden vom Monitor nicht geechot. Bei Auftreten eines Schreibfehlers bricht die Funktion ab. Der Benutzer kann die Übertragung (nach Verlassen der Upload-Funktion) durch Eingabe von <ESC> beenden.

Der Monitor verarbeitet S-Record Zeilen vom Typ S0, S1 und S9. Der Aufbau dieser S-Records ist im Anhang ausführlich beschrieben.

### Wichtig:

Beim Laden von S-Records direkt in den Flash-Memory ist zu beachten, daß Schreibzugriff eine gewisse Ausführungszeit benötigen. Die Daten dürfen nicht schneller gesendet werden, als sie der Monitor verarbeiten kann.

Zur Synchronisierung wurde ein einfaches Protokoll implementiert: Nach jeder verarbeiteten S-Record Zeile sendet der Monitor das Zeichen * (Stern). Damit signalisiert er die Bereitschaft zum Empfang der nächsten S-Record Zeile. Viele Terminalprogramme (z.B. das auf unserer Website frei erhältliche OC-Console) können diese Minimalprotokoll auswerten.

Alternativ gibt es folgende Möglichkeit: Die S-Record Datei wird mit voller Geschwindigkeit zunächst in den RAM geladen. Zur automatischen Umrechnung der Ladeadressen in den RAM Bereich nutzt man die Offset-Option des Ladebefehls. Anschließend wird der Inhalt mit dem Move Kommando in den Flash-Bereich kopiert.

## Move Memory

### Syntax: M <AADR> <EADR> <ADR2>

Kopiert den Speicherbereich von <AADR> bis <EADR> nach <ADR2>. Daraus ergibt sich die Länge <LEN> des zu kopierenden Speicherbereiches mit:

<LEN> = <EADR> - <AADR>

Für den Befehl sind drei Argumente erforderlich, z.B.:

| | |
|---|---|
| M 1000 1800 F000 | Kopiert den Speicherbereich von $1000 bis $17FF nach $F000 (bis $F7FF) |

Eventuelle Überschneidungen von Quell- und Zielbereich werden korrekt behandelt, indem das Kopieren entweder mit der niedrigsten oder der höchsten Adresse beginnt. Ist die Zieladresse <ADR2> kleiner als die Quelladresse <AADR> wird mit steigenden Adresse kopiert, andernfalls mit sinkenden Adressen. Bei Auftreten eines Schreibfehlers bricht die Funktion ab.

## Page Select

### Syntax: P (<PAGE>)

Selektiert eine Speicherseite (PPAGE-Register) aus dem Speicherbereich des Flash-Memories. Die selektierte Page wird in das Program Window ab $8000 eingeblendet (Länge: 16 KB).

Die unbenutzten oberen Bits der Pagenummer sind unrelevant. Um die letzte Page des 29F400 Flash Bausteins auszuwählen, kann man statt $1F als Pagenummer z.B. auch $FF angeben.

Wird keine Pagenummer als Argument angegeben, gibt der Monitor lediglich die augenblicklich ausgewählte Pagenummer aus.

| | |
|---|---|
| P | Zeigt die aktuell selektiert PPAGE Nummer an |
| P 22 | Selektiert PPAGE=$22 |

## Real Time Clock

### Syntax: T (<HH> <MM> <SS> <DD> <MM> <YY>)

Ohne Angabe von Argumenten werden Uhrzeit und Datum von der Echtzeituhr geladen und angezeigt.

Setzen der Uhr erfolgt mit dem gleichen Kommando unter Angabe von sechs Argumenten: Stunde, Minute, Sekunde, Tag, Monat, Jahr. Die Sekundenangabe wird durch die verwendete RTC nicht ausgewertet (wird stets auf Null gesetzt). Die Argumente werden als BCD-Werte interpretiert.

| | |
|---|---|
| T | Zeigt die aktuelle Uhrzeit und das Datum an |
| T 14 30 00 18 10 98 | Setzt die Uhr auf 14:30 Uhr am 18. Oktober '98 |

## Digital/Analog Converter

### Syntax: V <CH0> <CH2>

Spricht den Analog/Digital Converter an. Die beiden Argumente werden auf den DAC-Kanälen ausgegeben. Der Wertebereich ist jeweils $0000 bis $0FFF (12 Bit). $0000 entspricht 0V, $0FFF entspricht 4,095V.

| | |
|---|---|
| V 800 FFF | Gibt 2,048V auf DAC Kanal 0 und 4,095V auf DAC Kanal 1 aus |

## Sektor/Bulk Erase Flash

### Syntax: X (<SECTOR>)

Löscht einen einzelnen Sektor des Flash-Bausteins. Die verwendete Sektoradresse <SECTOR> entspricht der Nomenklatur im Datenblatt der 29Fx00 Flash Memories. Die Sektoradresse ergibt sich aus den für das Erase-Kommando am Flash Memory anzulegenden Adressen A12..A17.

Fehlt die Angabe der Sektornummer, wird der ganze Chip gelöscht (Bulk-Erase).

Zum Schutz vor versehentlichem Löschen wird eine Rückfrage an den Benutzer gerichtet. Erst nach Eingabe der Bestätigung 'Y' wird der Löschvorgang ausgeführt.

Dies ist die Sektortabelle des Am29F400T (inkl. Umrechnung in PPAGE-Nummern):

| Sektor Addresse | Sektor Länge | Speicher-Bereich | PPAGE |
|---|---|---|---|
| 00 | 64KB | 00000–0FFFF | 00-03 |
| 08 | 64KB | 10000–1FFFF | 04-07 |
| 10 | 64KB | 20000–2FFFF | 08-0B |
| 18 | 64KB | 30000–3FFFF | 0C-0F |
| 20 | 64KB | 40000–4FFFF | 10-13 |
| 28 | 64KB | 50000–5FFFF | 14-17 |
| 30 | 64KB | 60000–6FFFF | 18-1B |
| 38 | 32KB | 70000–77FFF | 1C-1D |
| 3C | 8KB | 78000-79FFF | 1E (untere Hälfte) |
| 3D | 8KB | 7A000-7BFFF | 1E (obere Hälfte) |
| 3E | 16KB | 7C000-7FFFF | 1F |

# Anhang

## S-Record Format

Das von Motorola publizierte S-Record Format ist ein Dateiformat zur Definition von Objektdateien (Maschinencode, Executables) unter Verwendung einer textuellen (ASCII-) Notation, die es erlaubt, diese Objektdateien mit jedem beliebigen Texteditor zu betrachten oder zu ändern. Eine S-Record Datei besteht aus einer beliebigen Anzahl S-Records bzw. Zeilen. Eine jede Zeile hat die folgende logische Struktur:

| ID | LEN | ADDR | DATA | CS | <EOL> |
|---|---|---|---|---|---|

Das Feld ID gibt den S-Record Typ an. Relevant sind die Typen "S1", "S9" und gelegentlich "S0" (Kommentarrecord). Außer dem ID Feld bestehen alle weiteren Felder aus Paaren von Hexziffern, beispielsweise "A9", "55" oder "0F".

Das Feld LEN besteht aus einem derartigen Paar und bestimmt die Anzahl der folgenden Ziffernpaare (enthält die Ziffernpaare der Felder ADDR, DATA und CS).

ADDR ist die Anfangsadresse der Datenbytes dieser Zeile. Das Feld besteht aus zwei Byte (erst H-, dann L-Byte), d.h. aus zwei Ziffernpaaren.

DATA enthält die eigentlichen Codebytes, die das Maschinenprogramm bilden. DATA umfaßt (LEN - 3) Bytes bzw. Zeichenpaare.

Im Feld CS ist eine Prüfsumme enthalten. Sie wird gebildet aus den Werten der Zeichenpaare der Felder LEN, ADDR und DATA. CS ist das (niederwertigste Byte des) Einerkomplement der Summe aller vorgenannten Werte. EOL schließlich steht symbolisch für den durch CR, LF ($0D, $0A) gebildeten Zeilenvorschub.

Ein Beispiel soll die Handhabung verdeutlichen:

| S1 | 13 | 2000 | 13A400262741010167CC10FF05C7A501 | D1 | <EOL> |
|---|---|---|---|---|---|

Dieser S1-Record definiert $13-3 = $10 Bytes ab Adresse $2000 des Zielsystems. Die Ziffernpaare des DATA Feldes ergeben eine Summe von $04FB. Addiert man die $13 aus dem LEN Feld sowie $20 und $00 aus dem ADDR Feld hinzu, ergibt sich ein Wert von $052E. Das Einerkomplement des LSB ($2E) ergibt $D1. Dies ist der korrekte Wert für das Prüfsummenfeld.

Neben den S1-Records, welche die eigentlichen Daten enthalten, wird auch der S9-Typ verwendet. Dieser Typ beendet eine Serie von S1-Records. Abgesehen von dieser Terminierungs-Funktion kann in einem S9-Record die Startadresse des Programms vermerkt werden. Der Aufbau des S9-Records entspricht dem S1 Typ, wobei jedoch das Feld DAT leer bleibt. Das Feld ADDR spezifiziert die Startadresse des Programms. Ist hier $0000 eingetragen, wird angenommen, daß die Adresse des ersten geladenen Codebyte gleichzeitig die Startadresse des Programms ist. Ein typischer S9-Record sieht wie folgt aus:

| S9 | 3 | B600 | | 46 | <EOL> |
|---|---|---|---|---|---|

## EMV Hinweise

Die Baugruppe entspricht den EMV-Vorschriften. Zur Stromversorgung ist sie an einer Batteriespannungsquelle mit 5,0 Volt (Einhaltung der Spannungsgrenzwerte beachten!) oder an ein Netzteil mit CE-Kennzeichnung anzuschließen. Der Einsatz einer Mikrocontrollerplatine geht stets einher mit einer mehr oder minder umfangreichen Modifikation der Baugruppe (spezielle Firmware, angeschlossene Peripheriebauteile). Der Hersteller kann den vom Kunden geplanten Einsatz der Baugruppe nicht vorhersehen und daher auch keine Vorhersagen über die EMV-Eigenschaften der modifizierten Baugruppe machen. Anwender ohne Zugriff auf ein EMV-Prüflabor sollten die folgenden Richtlinien beachten, die in der Regel eine einwandfreie Funktion der modifizierten Baugruppe gewährleisten:

Um sicherzustellen, daß die Baugruppe auch dann den EMV-Vorschriften entspricht, wenn Verbindungsleitungen zu anderen Geräten (z.B. Personalcomputer) angeschlossen werden oder die Baugruppe vom Kunden selbst mit weiteren Bauteilen nachgerüstet wird (z.B. Meßadapter oder Leistungsendstufen), empfehlen wir, die komplette Baugruppe in ein allseitig geschlossenes Metallgehäuse einzusetzen.

Wird ein LC-Display angeschlossen (ebenfalls auf CE-Kennzeichnung achten), so darf das Verbindungskabel nicht länger als 10 cm sein; hier ist auf jeden Fall ein Metallgehäuse vorzusehen. Wenn für die Programmentwicklung oder die spätere Anwendung die RS232 Schnittstelle benötigt wird, so ist ein max. 10cm langes Kabel zur Verbindung mit der Anschlußbuchse zu verwenden. Die geschirmte Anschlußbuchse ist fest mit dem Metallgehäuse zu verschrauben. Extern zur Verbindung verwendete Anschlußkabel müssen, ebenso wie der Hostrechner (PC), mit dem CE-Prüfzeichen versehen sein.

Es wird darauf hingewiesen, daß der Anwender selbst dafür verantwortlich ist, daß eine veränderte, erweiterte, mit anderen als vom Hersteller gelieferten IC´s bestückte oder mit Anschlußkabeln versehene Baugruppe den EMV-Vorschriften entspricht.