CardS12

Hardware Version 1.10

Benutzerhandbuch

16. November 2006

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

CardS12 ist ein einfach anzuwendendes Controller Modul im Scheckkartenformat auf Basis der 16-Bit Mikrocontrollerfamilie HCS12 von Freescale Semiconductor (ehem. Motorola). Das CardS12 Modul erleichtert die Evaluierung des Mikrocontrollerbausteins und ist eine schnell verfügbare, kostengünstige Ausgangsbasis für die Realisierung kleiner bis mittlerer Serienanwendungen.

Auf dem Modul CardS12.D64 kommt eine leistungsstarke MCU vom Typ MC9S12D64 zum Einsatz. Dieser Mikrocontroller enthält die 16-Bit HCS12 CPU, 64KB Flash, 4KB RAM, 1KB EEPROM und eine große Menge integrierter Peripheriefunktionen, wie SCI, SPI, CAN, IIC, Timer, PWM, ADC und Input-/Output-Kanäle. Der MC9S12D64 ist vollständig mit 16 Bit breiten internen Datenpfaden ausgestattet. Die integrierte PLL-Schaltung ermöglicht es, Performance und Strombedarf auf einfache Weise den jeweiligen Anforderungen anzupassen.

CardS12 ist mit zwei weiteren Controllervarianten erhältlich. Statt des MC9S12D64 kann ein MC9S12DG256 oder ein MC9S12DP512 eingesetzt werden.

Die für die HCS12-Controller erhältliche umfassende Softwareunterstützung (Monitor, C-Compiler, BDM-Debugger) erleichtert die Entwicklung von Embedded Systemen jeglicher Art.

Technische Daten

- MCU MC9S12D64 im LQFP112 Package (SMD)
- HCS12 16-Bit CPU, Programmiermodell und Befehlssatz wie beim HC12
- 16 MHz Quarztakt, bis zu 25 MHz Bustakt über PLL
- 64KB Flash
- 1KB EEPROM
- 4KB RAM
- 2x SCI - asynch. serial Interface (z.B. RS232, LIN)

- 1x SPI - synch. serial Interface
- 1x IIC - Inter-IC-Bus
- 1x msCAN-Module (CAN 2.0A/B-kompatibel)
- 8x 16-Bit Timer (Input Capture/Output Compare)
- 8x PWM (Pulse Width Modulator)
- 16-Kanal 10-Bit A/D-Wandler
- BDM - Background Debug Mode Schnittstelle
- Spezieller LVI-Schaltkreis (Reset Controller)
- BDM-Anschluß für Download und Debugging
- Zwei serielle Interfaces mit RS232-Treiber ausgerüstet, z.B. für PC-Verbindung, der zweite serieller Port ist auch zur Direkt-ansteuerung serieller LC-Displays geeignet
- Indikator-LED
- High-Speed phys. CAN-Interfacetreiber
- Resettaster
- bis zu 87 freie Ein-/Ausgabeleitungen, alle I/O-Anschlüsse sind auf seitliche Steckverbinder herausgeführt
- Betriebsspannung 5V, typ. Stromaufnahme ca. 50 mA
- Abmessungen 54mm x 86mm

CardS12.DG256, abweichend zur D64-Version:

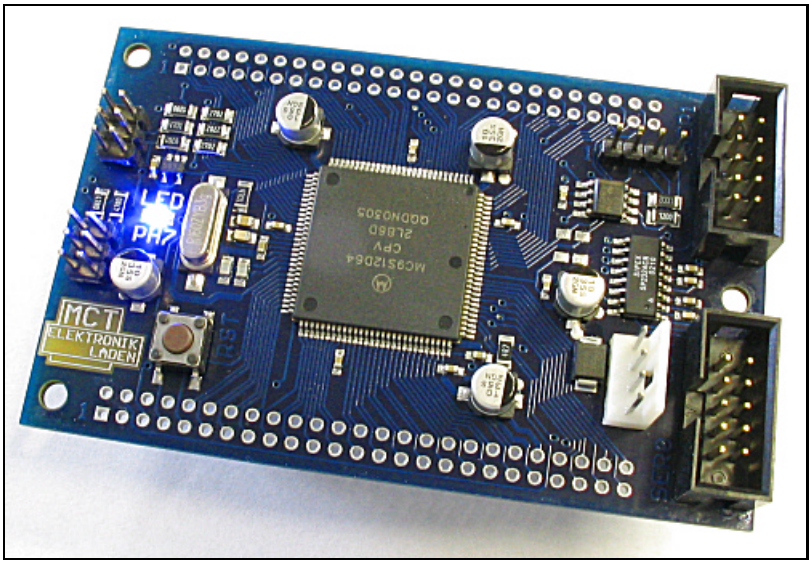
- MCU MC9S12DG256
- 256KB Flash
- 4KB EEPROM
- 12KB RAM
- drei SPI-Module
- ein zusätzliches msCAN Modul (ohne on-board Bustreiber)

CardS12.DP512, abweichend zur D64-Version:

- MCU MC9S12DP512
- 512KB Flash
- 4KB EEPROM
- 14KB RAM
- drei SPI-Module
- vier zusätzliche msCAN Module (ohne on-board Bustreiber)

Lieferumfang

- Controller Modul mit MC9S12D64 (CardS12.D64) bzw. MC9S12DG256 (CardS12.DG256) bzw. MC9S12DP512 (CardS12.DP512)
- TwinPEEKs Monitorprogramm (im Flash Speicher der MCU)
- RS232 Anschlußkabel (Sub-D9)
- Randsteckverbinder (zwei 50-polige Stiftleisten)
- Hardwarehandbuch (dieses Dokument)
- Schaltplan
- CD-ROM: enthält Assemblersoftware, verschiedene Datenblätter, CPU12 Reference Manual, Softwarebeispiele, C-Compiler Demoversion u.v.m.



Controller Modul CardS12.D64

2. Schnellstart

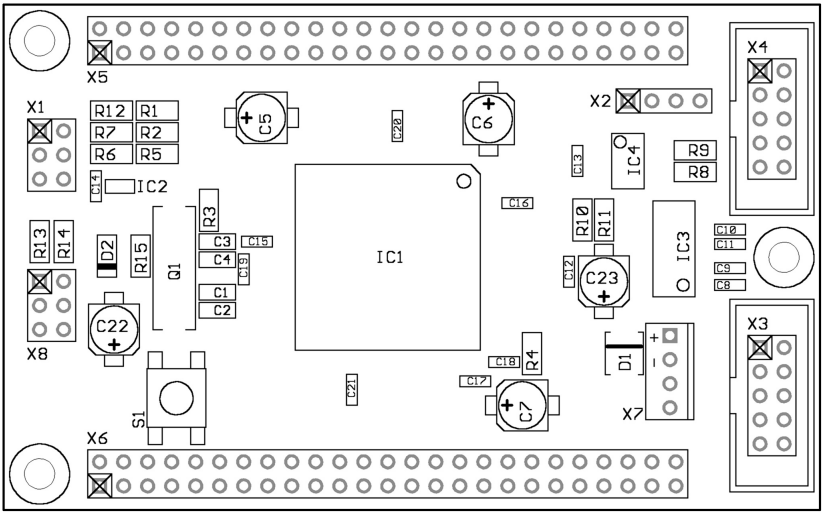
Kein Mensch liest gern dicke Handbücher. Daher hier die wichtigsten Hinweise in Kürze. Wenn Sie sich jedoch über ein Detail einmal nicht sicher sind, dann informieren Sie sich am Besten in den nachfolgenden Kapiteln.

Und so können Sie beginnen:

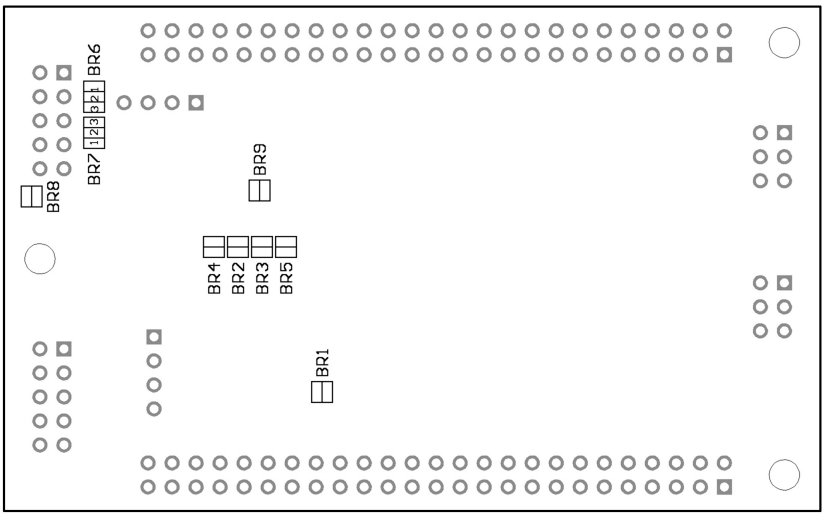
- Überprüfen Sie die Baugruppe zuerst auf offenkundige Transportschäden.
- Verbinden Sie das Controller Modul via RS232 mit Ihrem PC. Die Verbindung zwischen CardS12 (Schnittstelle SER0, Steckverbinder X3) und PC erfolgt über das mitgelieferte 10-pol. Flachbandkabel.
- Starten Sie auf dem PC ein Terminalprogramm. Ein einfaches Programm wie OC-Console (kostenlos auf unserer Website!) reicht aus.
- Stellen Sie die Baudrate auf **19200** Baud. Schalten Sie alle zusätzlichen Protokolle (Hard- und Softwarehandshake) aus.
- Schließen Sie die (stabilisierte!) Versorgungsspannung an den Einplatinenrechner an, z.B. hier:
- Masse an X7 Pin 2
- +5V an X7 Pin 1
- Vergewissern Sie sich **zuvor** von der richtigen Spannung und Polarität!
- Daraufhin startet das Monitorprogramm und zeigt eine kurze Systemmeldung an. Mit Ausgabe des Promptzeichens erwartet es Ihre Anweisungen.

Wir wünschen Ihnen viel Erfolg bei Ihrer Arbeit mit der CardS12!

3. Bestückungsplan



Lageplan Bestückungsseite (V1.10)



Lötbrücken auf der Platinenrückseite (V1.10)

4. Jumper und Lötbrücken

Jumper

Auf dieser Baugruppe sind keine Jumper vorhanden.

Lötbrücken

Die folgenden Lötbrücken befinden sich auf der Unterseite der Platine (vergl. Lageplan auf vorhergehender Seite):

BR1: VRH

offen	externe Einspeisung VRH erforderlich
geschl.*	VRH on-board mit VDDA (VCC) verbunden

BR2: T1IN

offen	Portpin TXD0 (PS1) frei verfügbar
geschl.*	TXD0 mit RS232-Pegelwandler IC3 verbunden

BR3: T2IN

offen	Portpin TXD1 (PS3) frei verfügbar
geschl.*	TXD1 mit RS232-Pegelwandler IC3 verbunden

BR4: R1OUT

offen	Portpin RXD0 (PS0) frei verfügbar
geschl.*	RXD0 mit RS232-Pegelwandler IC3 verbunden

BR5: R2OUT

offen	Portpin RXD1 (PS2) frei verfügbar
geschl.*	RXD1 mit RS232-Pegelwandler IC3 verbunden

* = Standardeinstellung

BR6, BR7: RS232 TxD/RxD Select (SER1)

1-2*	RS232 als "Device" konfiguriert (für Verbindung zu einem PC)
2-3	RS232 als "Host" konfiguriert (für Verbindung zu seriellem LCD o.ä.)

BR8: LCD Power Supply (SER1)

offen*	VCC nicht am RS232 Anschluß SER1 verfügbar (normale Belegung)
geschl.	VCC am RS232 Port SER1 verfügbar (Pin9 des Sub-D Steckers)

BR9: RXCAN0

offen	Portpin RXCAN0 (PM0) frei verfügbar
geschl.*	RXCAN0 mit CAN-Transceiver IC4 verbunden USB-Transceiver aus

* = Standardeinstellung

5. Mechanische Abmessungen

Die folgende Tabelle gibt die mechanischen Dimensionen des CardS12 Controller Moduls wieder. Die Angaben dienen als Orientierung beim Entwurf von Trägerplatinen/-baugruppen (Achtung: Angaben stets an den gelieferten Baugruppen nachprüfen - keine Haftung für Druckfehler!).

Die "südwestliche" Ecke der Platine bildet den Koordinatenursprung. Die Lage der Platine ist horizontal, wie im Bestückungsplan (s.o.) dargestellt.

Alle Angaben zu Bohrungen (B) beziehen sich auf die Mitte, bei Steckverbindern (X) auf die Lage von Pin 1.

	X in Zoll	Y in Zoll
X1	0,150	1,575
X2	2,600	1,700
X3	3,150	0,675
X4	3,150	1,825
X5	0,400	1,900
X6	0,400	0,100
X7	2,775	0,725
X8	0,150	0,950
B1	0,150	0,150
B2	0,150	1,950
B3	3,250	1,050
PCB	3,400	2,100

6. Schaltungsbeschreibung

Bitte beachten Sie: 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 (unerläßlichen) Studium der Datenblätter und Referenzhandbüchern der Halbleiterhersteller.

Die im Text eingestreuten Beispielprogramme dienen lediglich der Demonstration. Für die Korrektheit und die Eignung für eine bestimmte Aufgabe können wir *keine Garantie* geben!

Schaltplan

Damit alle Details gut lesbar bleiben, liegt der Schaltplan im A4-Format separat bei.

Stromversorgung

Der Mikrocontroller (IC1) verfügt über drei Anschlußpaare zur Zuführung der Versorgungsspannung: VDDR/VSSR, VDDX/VSSX und VDDA/VSSA. Die Betriebsspannung beträgt nominal 5 Volt, intern arbeitet der Prozessor jedoch mit 2,5 Volt. Der hierzu erforderliche Spannungsregler ist bereits in der MCU integriert. VREGEN gibt den internen Spannungsregler frei, der Pin ist normalerweise stets mit H-Pegel (5V) zu verbinden.

Die Spannungsreduzierung im Core ist in erster Linie erforderlich durch die geringen Strukturbreiten des Fertigungsprozesses (0,25µm und kleiner). Von außen verhält sich der HCS12 jedoch wie ein 5V-Baustein, da an den Ein-/Ausgabepins Pegelwandler vorhanden sind. Eine Ausnahme stellen die Anschlüsse für Oszillator und PLL dar, näheres dazu unten.

Die drei genannten Versorgungsanschlußpaare müssen sorgfältig entkoppelt werden. In unmittelbarer Nähe der Pins befindet sich daher

je ein 100nF-Keramikkondensator (C15, C16, C17), dem zusätzlich ein 10µF Elektrolytkondensator parallel geschaltet wird (C5, C6, C7). Besonderes Augenmerk muß auf die Entkopplung des VDDA-Pfades gelegt werden, da der interne Spannungsregler aus dieser Spannung seinen Referenzwert (VDDA/2) ableitet.

Die interne 2,5-Volt-Corespannung wird an mehreren Stellen nach außen geführt, um sie dort ebenfalls entkoppeln zu können. Hierzu sind an den Anschlußpaaren VDD1/VSS1, VDD2/VSS2 sowie VDDPLL/VSSPLL weitere Keramikkapazitäten vorgesehen (C19, C20, C21). Eine statische Belastung der internen Betriebsspannung durch externe Schaltungskomponenten ist nicht statthaft! Das gilt grundsätzlich auch für VDDPLL, die als Referenzpunkt für die extern angeschlossene PLL-Filterkombination (R3, C3, C4) dient.

In die Domäne der Versorgungsspannungen fällt auch die Referenzspannung für die integrierten Analog-Digital-Wandler. Die untere Referenzspannungsgrenze wird über den Anschluß VRL festgelegt, welcher hier (wie meist üblich) auf Massepotential liegt. Die obere Referenzspannung VRH ist über die Lötbrücke BR1 mit VDDA verbunden, C18 dient hier zur Entkopplung. Um die Auflösung der internen 10-Bit A/D-Wandler voll auszuschöpfen, kann eine externe Referenzspannung eingespeist werden. In diesem Fall ist BR1 zu öffnen. VRH darf jedoch VDDA niemals übersteigen.

Der TEST-Pin wird nur werkseitig bei Freescale verwendet, in Anwenderschaltungen ist dieser Pin stets mit dem Massepotential zu verbinden.

Reseterzeugung

/RESET ist der bidirektionale, L-aktive Resetpin der MCU. Als Eingang dient er zur Initialisierung der MCU beim Einschalten. Als Open-Drain-Ausgang signalisiert er, dass innerhalb der MCU ein Resetereignis stattgefunden hat. Die HCS12 MCU enthält bereits Schaltungen für Power-On Reset, COP (Watchdog) and Clock Monitor Reset. Es ist dennoch notwendig, zusätzlich einen externen LVI-Schaltkreis vorzusehen, welcher die Aufgabe hat, zuverlässig Reset auszulösen, sobald die Versorgungsspannung der MCU unter den zulässigen Mindestwert gefallen ist.

Der LVI-Schaltkreis IC2 hat einen Open-Drain Ausgang, um Kollisionen mit dem bidirektionalen Resetpin der MCU zu vermeiden. Im inaktiven Zustand stellt sich an /RESET (dank des Pull-Up Widerstands R6) H-Pegel ein. Die Schaltschwelle von IC2 liegt bei typischerweise 4,6V. Das ist geringfügig höher als die Mindestbetriebsspannung der MCU (4,5V).

IC2 ist in der Lage, den Resetimpuls auf eine gewisse Mindestlänge auszudehnen, die über den Kondensator C14 festgelegt wird. Bei 100nF beträgt der Delay ca. 50..80ms.

Es ist wichtig zu bemerken, dass diese Impulsverlängerung nur bei einem Power-On Reset wirksam wird. Die MCU-internen Resetimpulse werden von IC2 hingegen nicht gedehnt, denn sonst wäre die MCU nicht mehr in der Lage, die korrekte Resetquelle zu ermitteln. Die Konsequenz wäre sonst u.U. ein Programmabsturz durch die Verwendung eines falschen Resetvektors. Es ist daher ebenso wichtig, niemals größere Kapazitäten an die Resetleitung des HCS12 anzuschliessen, denn der resultierende Effekt wäre der selbe.

Takterzeugung und PLL

Der On-Chip Oszillator des MC9S12Dxx kann den primären Takt (OSCCLK) mit Hilfe eines Quarzes (Q1) erzeugen, der an die Pins EXTAL und XTAL angeschlossen wird. Der zulässige Frequenzbereich ist 0,5 bis 16 MHz. Wie üblich sind zwei Lastkapazitäten (C1, C2) Teil der Oszillatorschaltung. Die Anordnung ist jedoch modifiziert, wenn man die Schaltung mit der Standard-Pierce Konfiguration vergleicht, wie sie beim HC11 und den meisten HC12-Typen verwendet wurde.

Der MC9S12Dxx verwendet einen Colpitts-Oszillator mit translated Ground. Der Hauptvorteil dieser Oszillatorschaltung ist eine sehr geringe Leistungsaufnahme, dafür ist die Komponentenwahl um einiges kritischer. Auf der CardS12 finden ausgewählte Quartze und Last-Cs mit geringer Kapazität Verwendung. Darüber hinaus wurde beim Design besonders auf die Minimierung von parasitären Kapazitäten geachtet, die sich nachteilig auf die Signale EXTAL und XTAL auswirken könnten.

Mit einem OSCCLK von 16 MHz ergibt sich ein Default-Bustakt (ECLK) von 8 MHz. Zur Erreichung höherer Taktfrequenzen bedient man sich der PLL-Schaltung des HCS12. Der MC9S12Dxx kann intern mit bis zu 25MHz Bustakt arbeiten, wobei die meisten Designs eine Frequenz von 24MHz nutzen, denn dies möglicht eine besonders flexible Festlegung der SCI-Baudraten.

An den Controllerpin XFC wird eine Tiefpassfilterkombination angeschlossen, sie besteht aus den Bauelementen R3, C3 und C4. Ihre Aufgabe ist die Verminderung der Welligkeit des VCO-Signals. Falls die PLL unbenutzt bleibt, kann XFC mit VDDPLL verbunden werden, andernfalls bildet VDDPLL den Bezugspotenzial für den Filter.

Die Wahl der Filterkomponenten ist stets ein Kompromiss zwischen Einschwingzeit und Stabilität der Schleife. 5kHz bis 10kHz Bandbreite und ein Dampingfaktor von 0,9 sind gute Startwerte für die Berechnung. Mit einer Quarzfrequenz von 16MHz und einem gewünschten Busclock von 24MHz ergibt sich eine mögliche Auswahl zu R3=4,7k und C3=22nF. C4 sollte etwa fünf bis zehn Prozent von C3 betragen, hier also 2,2nF. Diese Werte sind passend für eine

Referenzfrequenz von 1MHz (Achtung: diese Vorgabe ist in der Beispieldatei S12_CRG.H zu definieren). Die Einstellung für das Referenzteilerregister ergibt sich zu REFDV=15 und das Synthesizerregister erhält den Wert SYNR=23. Das Kapitel "XFC Component Selection" im MC9S12DP256B Device User Guide illustriert die erforderlichen Rechenschritte im Detail.

Das folgende Listing zeigt die erforderlichen Initialisierungsschritte für die PLL:

```
//=============================================================================
// File: S12_CRG.C - V1.00
//=============================================================================

//-- Includes ---------------------------------------------------------------

#include <hcs12dp256.h>
#include "s12_crg.h"

//-- Code -------------------------------------------------------------------

void initPLL(void) {

    CLKSEL &= ~BM_PLLSEL;               // make sure PLL is *not* in use
    PLLCTL |= BM_PLLON+BM_AUTO;         // enable PLL module, Auto Mode
    REFDV = S12_REFDV;                  // set up Reference Divider
    SYNR = S12_SYNR;                    // set up Synthesizer Multiplier
    // the following dummy write has no effect except consuming some cycles,
    // this is a workaround for erratum MUCTS00174 (mask set 0K36N only)
    // CRGFLG = 0;
    while((CRGFLG & BM_LOCK) == 0) ;    // wait until PLL is locked
    CLKSEL |= BM_PLLSEL;                // switch over to PLL clock
    }

//=============================================================================
```

Alternativ zur Takterzeugung mit Q1 kann über den EXTAL-Pin des MC9S12Dxx ein externer Takt eingespeist werden. /XCLKS muss hierzu während Reset auf L-Pegel gelegt werden. Da von dieser Variante auf dem CardS12 Modul keine Verwendung gemacht wird, dient R5 dazu, /XCLKS während Reset auf H-Pegel zu halten. Achtung: die verschiedenen HCS12-Typen haben z.T. abweichende Funktionalität hinsichtlich des /XCLKS-Pins.

Betriebsarten, BDM-Unterstützung

Drei Pins des HCS12 dienen der Auswahl der MCU-Betriebsart: MODA, MODB und BKGD (=MODC). MODA und MODB werden durch die Widerstände R1 und R2 auf L-Pegel gebracht, um Single Chip Mode auszuwählen. BKGD ist über R7 mit H-Pegel verbunden, damit die MCU im Normal Single Chip Mode startet. Dies ist die übliche Betriebsart zur Abarbeitung von Anwendungsprogrammen.

Die HCS12 Betriebsart, welche für Download und Debugging genutzt wird, heisst Background Debug Mode (BDM). BDM ist direkt nach Reset aktiv, wenn die MCU im Special Single Chip Mode betrieben wird. Dies wird erreicht, indem - zusätzlich zu MODA und MODB - auch die BKGD-Leitung während Reset vorübergehend auf L-Pegel gebracht wird.

Zwischen beiden Modi kann man leicht umschalten, da sich lediglich der Resetzustand der BKGD-Leitung unterscheidet. Ein BDM-Pod, welches am Steckverbinder X1 angeschlossen wird, kann die Umschaltung automatisch vornehmen, und macht einen mechanischen Umschalter überflüssig. Das BDM-Pod wäre ohnehin notwendig zum BDM-basierten Download von Software bzw. als Debugger, gesteuert von Software auf einem (Entwicklungs-) PC.

Integrierter A/D-Wandler

Der MC9S12Dxx verfügt über zwei integrierte Analog/Digital-Wandler Module mit einer Auflösung von max. 10 Bit. Beide Module (ATD0, ATD1) haben jeweils acht gemultiplexte Eingänge.

Die Referenzspannung VRH legt die obere Grenze der Eingangsspannung aller A/D-Kanäle fest, sie ist auf der CardS12 ab Werk über BR1 mit VDDA (5V) verbunden. Durch Öffnen der Lötbrücke BR1 ist es möglich, über X6/42 eine externe Referenzspannung einzuspeisen.

Das folgende Beispielprogramm zeigt die Initialisierungssequenz für das A/D-Wandler Modul ATD0 und eine Routine zum Erfassen des Spannungswertes eines einzelnen Eingangskanals. Weitere

Beispielroutinen für das integrierte ATD-Modul sind in der Quelltext-datei S12_ATD.C enthalten.

```
//=============================================================================
// File: S12_ATD.C - V1.00
//=============================================================================

//-- Includes ---------------------------------------------------------------

#include "datatypes.h"
#include <hcs12dp256.h>
#include "s12_atd.h"

//-- Code -------------------------------------------------------------------

// Func: Initialize ATD module
// Args: -
// Retn: -
//
void initATD0(void) {

    // enable ATD module
    ATD0CTL2 = BM_ADPU;
    // 10 bit resolution, clock divider=12 (allows ECLK=6..24MHz)
    // 2nd sample time = 2 ATD clocks
    ATD0CTL4 = BM_PRS2 | BM_PRS0;
    }

//-----------------------------------------------------------------------------

// Func: Perform single channel ATD conversion
// Args: channel = 0..7
// Retn: unsigned, left justified 10 bit result
//
UINT16 getATD0(UINT8 channel) {

    // select one conversion per sequence
    ATD0CTL3 = BM_S1C;
    // right justified unsigned data mode
    // perform single sequence, one out of 8 channels
    ATD0CTL5 = BM_DJM | (channel & 0x07);
    // wait until Sequence Complete Flag set
    // CAUTION: no loop time limit implemented!
    while((ATD0STAT0 & BM_SCF) == 0) ;
    // read result register
    return ATD0DR0;
    }

//-----------------------------------------------------------------------------
```

Integriertes EEPROM

Der interne EEPROM-Speicher des MC9S12D64 ist 1KB groß und in 256 Sektoren zu je 4 Byte (32 Bit) unterteilt. Gelöscht wird stets sektorweise (4 Byte), während die Programmierung wortweise (2 Byte) erfolgen kann. Lesezugriffe auf den EEPROM erfolgen beliebig, also byte- oder wortweise.

Nach Reset ist der EEPROM Bereich im MC9S12D64 ab Adresse 0x0000 gemappt, wird jedoch dort (0x0000..0x03FF) von den Steuerregistern überlagert. Der selbe EEPROM Bereich ist ein zweites mal ab 0x0400 eingeblendet. Der EEPROM-Bereich kann zudem in 2KB-Schritten verschoben werden (INITEE Register).

Das folgende Beispiel beläßt den EEPROM auf der Defaultposition, in der Initialisierungsroutine wird lediglich der EEPROM Clock Devider entsprechend der Quarzfrequenz der Baugruppe eingestellt. Die Schreibfunktion wrSectEETS() kopiert zwei Worte (4 Byte) von einer beliebigen Quelladresse src auf eine EEPROM-Adresse dest, letztere muß identisch mit einer EEPROM-Sektorgrenze sein (aligned 32 bit). Ist der Inhalt des Zielsektors nicht gelöscht (0xFFFFFFFF), wird zunächst automatisch ein Sector-Erase ausgeführt.

Die Zugriffsfunktionen readItemEETS() und writeItemEETS() verallgemeinern den EEPROM-Zugriff dahin gehend, dass nicht mehr mit EEPROM-Adressen gearbeitet wird, sondern mit einer abstrakten Numerierung von EEPROM-"Items". Jedes dieser EEPROM-"Items" kann 1 bis 4 Byte lang sein.

```
//============================================================================
// File: S12_EETS.C - V1.00
//============================================================================

//-- Includes ----------------------------------------------------------------

#include "datatypes.h"
#include <hcs12dp256.h>
#include "s12_eets.h"

//-- Code --------------------------------------------------------------------

void initEETS(void) {

    ECLKDIV = EETS_ECLKDIV;         // set EEPROM Clock Divider Register
    }

//----------------------------------------------------------------------------
```

```
INT8 wrSectEETS(UINT16 *dest, UINT16 *src) {

    // check addr: must be aligned 32 bit
    if((UINT16)dest & 0x0003) return -1;
    // check if ECLKDIV was written
    if((ECLKDIV & BM_EDIVLD) == 0) return -2;
    // make sure error flags are reset
    ESTAT = BM_PVIOL | BM_ACCERR;
    // check if command buffer is ready
    if((ESTAT & BM_CBEIF) == 0) return -3;
    // check if sector is erased
    if((*dest != 0xffff) || (*(dest+1) != 0xffff)) {
        // no, go erase sector
        *dest = *src;
        ECMD = EETS_CMD_SERASE;
        ESTAT = BM_CBEIF;
        if(ESTAT & (BM_PVIOL | BM_ACCERR)) return -4;
        while((ESTAT & BM_CBEIF) == 0) ;
        }
    // program 1st word
    *dest = *src;
    ECMD = EETS_CMD_PROGRAM;
    ESTAT = BM_CBEIF;
    if(ESTAT & (BM_PVIOL | BM_ACCERR)) return -5;
    while((ESTAT & BM_CBEIF) == 0) ;
    // program 2nd word
    *(dest+1) = *(src+1);
    ECMD = EETS_CMD_PROGRAM;
    ESTAT = BM_CBEIF;
    if(ESTAT & (BM_PVIOL | BM_ACCERR)) return -6;
    while((ESTAT & BM_CCIF) == 0) ;
    return 0;
    }

//-----------------------------------------------------------------------------

INT8 writeItemEETS(UINT16 item_no, void *item) {

    if(item_no >= EETS_MAX_SECTOR) return -7;
    item_no = EETS_START + (item_no << 2);
    return wrSectEETS((UINT16 *)item_no, (UINT16 *)item);
    }

//-----------------------------------------------------------------------------

INT8 readItemEETS(UINT16 item_no, void *item) {

    if(item_no >= EETS_MAX_SECTOR) return -7;
    item_no = EETS_START + (item_no << 2);
    *((UINT16 *)item) = *((UINT16 *)item_no);
    *(((UINT16 *)item)+1) = *(((UINT16 *)item_no)+1);
    return 0;
    }

//=============================================================================
```

Indikator-LED

An Portpin PH7 ist eine Indikator-LED (D2)angeschlossen. Die Steuerung der Indikator-LED kann durch einige einfache Makros erfolgen, wie das folgende Headerfile zeigt.

```
//============================================================================
// File: CARDS12_LED.H - V1.00
//============================================================================

#ifndef __CARDS12_LED_H
#define __CARDS12_LED_H

//-- Macros ------------------------------------------------------------------

#define initLED()   PORTH |= 0x80; DDRH |= 0x80
#define offLED()    PORTH |= 0x80
#define onLED()     PORTH &= ~0x80
#define toggleLED() PORTH ^= 0x80

//-- Function Prototypes -----------------------------------------------------

/* module contains no code */

#endif //__CARDS12_LED_H ==================================================
```

RS232-Schnittstellen

Der MC9S12Dxx verfügt über zwei asynchrone Schnittstellen (SCI0, SCI1). Jede dieser Schnittstellen umfaßt zwei Signalleitungen (RXDx, TXDx). Handshakeleitungen sind nicht Bestandteil der SCI-Module des Controllers, sie sind ggf. durch Einbeziehung zusätzlicher I/O-Ports zu realisieren.

Die Signalleitungen der beiden SCI-Schnittstellen sind über die Lötbrücken BR2 bis BR5 mit dem RS232-Pegelwandler IC3 verbunden. Öffnet man diese Lötbrücken, können die Controllersignale anderweitig verwendet werden. Sie werden dazu am Steckverbinder X6 bereitgestellt.

Der RS232-Anschluß erfolgt über X3 (SCI0). Dieser Steckverbinder ist so gestaltet, dass durch ein Flachbandkabel mit angecrimpter Sub-D9 Buchse eine direkte Verbindung zu einem PC-COM-Port hergestellt werden kann. Dies gilt für X4 (SCI1) analog, vorausgesetzt die Brücken BR6 und BR7 auf der Platinenunterseite sind in Stellung 1-2 verbunden (Default). Der PC fungiert als Host, das CardS12 Modul bildet die Deviceseite.

Der umgekehrte Fall tritt z.B. ein, wenn ein serielles LC-Display am X4 betrieben werden soll. Hierbei ist die CardS12 der Host und das LCD-Modul tritt an die Deviceseite. Die hierzu erforderliche RxD/TxD-Leitungskreuzung wird realisiert durch Konfiguration der Brücken BR6/BR7 in Stellung 2-3. Gleichzeitig kann durch Schließen der Brücke BR8 das serielle LCD mit Betriebsspannung versorgt werden (Achtung: diese Belegung weicht von der RS232 Standardbelegung ab!). Serielle, alphanumerische LC-Displays gibt es von einer Vielzahl verschiedener Anbieter.

Das folgende Codebeispiel zeigt die Ansteuerung von SCI0 mittels Polling:

```
//============================================================================
// File: S12_SCI.C - V1.00
//============================================================================

//-- Includes ----------------------------------------------------------------

#include "datatypes.h"
#include <hcs12dp256.h>
#include "s12_sci.h"

//-- Code --------------------------------------------------------------------

void initSCI0(UINT16 bauddiv) {

    SCI0BD  = bauddiv & 0x1fff; // baudrate divider has 13 bits
    SCI0CR1 = 0;                // mode = 8N1
    SCI0CR2 = BM_TE+BM_RE;      // Transmitter + Receiver enable
    }

//----------------------------------------------------------------------------

UINT8 getSCI0(void) {

    while((SCI0SR1 & BM_RDRF) == 0) ;
    return SCI0DRL;
    }

//----------------------------------------------------------------------------

void putSCI0(UINT8 c) {

    while((SCI0SR1 & BM_TDRE) == 0) ;
    SCI0DRL = c;
    }

//----------------------------------------------------------------------------
```

SPI-Bus

Der MC9S12D64 verfügt über ein integriertes SPI-Modul (SPI0) zur synchronen, seriellen Kommunikation mit externen Peripheriechips.

SPI0 umfasst die Leitungen MISO, MOSI, SCK und /SS, das sind die MCU-Portleitungen PS4 bis PS7. Diese Signale werden in der Schaltung der CardS12 selbst nicht benutzt, sind aber an den seitlichen Stiftleisten präsent.

Das folgende Listing zeigt die Basisfunktionen (Initialisierung, 8-Bit Datentransfer) für den SPI-Port SPI0 (ohne Berücksichtigung von Chipselect-Signalen):

```
//============================================================================
// File: S12_SPI.C - V1.01
//============================================================================

//-- Includes ----------------------------------------------------------------

#include "datatypes.h"
#include <hcs12dp256.h>
#include "s12_spi.h"

//-- Code --------------------------------------------------------------------

void initSPI0(UINT8 bauddiv, UINT8 cpol, UINT8 cpha) {

    DDRS |= 0xe0;                       // SS,SCK,MOSI Output
    SPI0BR = bauddiv;                   // set SPI Rate
    // enable SPI, Master Mode, select clock polarity/phase
    SPI0CR1 = BM_SPE | BM_MSTR | (cpol ? BM_CPOL : 0) | (cpha ? BM_CPHA : 0);
    SPI0CR2 = 0;                        // as default
    }

//----------------------------------------------------------------------------

UINT8 xferSPI0(UINT8 abyte) {

    while((SPI0SR & BM_SPTEF) == 0) ;   // wait for transmitter available
    SPI0DR = abyte;                     // start transfer (write data)
    while((SPI0SR & BM_SPIF) == 0) ;    // wait until transfer finished
    return(SPI0DR);                     // read back data received
    }

//============================================================================
```

IIC-Bus

An den Pins PJ6 und PJ7 bietet der MC9S12Dxx bei Bedarf einen Inter-IC-Bus (IIC/I2C/I²C) Anschluß. Diese Funktion wird von einem integrierten Hardwaremodul des Controllers unterstützt, eine Emulation durch Software erübrigt sich somit.

Soll das IIC-Businterface genutzt werden, sind an den beiden Bussignalen (SDA, SCL) Pull-Up Widerstände vorzusehen. Wenn diese nicht bereits außerhalb des Controllermoduls angesiedelt sind, können sie optional direkt auf dem CardS12 Controller Modul bestückt werden (R10, R11).

Das folgende Listing zeigt eine vereinfachte Master-Mode Implementierung, welche auf die Nutzung von Interrupts verzichtet:

```
//============================================================================
// File: S12_IIC.C - V1.00
// Func: Simplified I2C (Inter-IC Bus) Master Mode implementation
//       using the IIC hardware module of the HCS12
// Rem.: For a real-world implementation, an interrupt-driven scheme should
//       be preferred. See AppNote AN2318 and accompanying software!
// Hard: External pull-ups on SDA and SCL required!
//       Value should be 1k..5k depending on cap. bus load
// Note: Adjust IBFD value if ECLK is not 8MHz!
//============================================================================

//-- Includes ----------------------------------------------------------------

#include "datatypes.h"
#include <mc9s12d64.h>
#include "s12_iic.h"

//-- Code --------------------------------------------------------------------

// Func: Initialize IIC module
// Args: -
// Retn: -
//
void initIIC(void) {

    IBFD = 0x18;                        // 100kHz IIC clock at 8MHz ECLK
//  IBFD = 0x1f;                        // 100kHz IIC clock at 24MHz ECLK
    IBCR = BM_IBEN;                     // enable IIC module, still slave
    IBSR = BM_IBIF | BM_IBAL;           // clear pending flags (just in case...)
    }

//----------------------------------------------------------------------------

// Func: Issue IIC Start Condition
// Args: -
// Retn: -
//
void startIIC(void) {

    while((IBSR & BM_IBB) != 0)         // wait if bus busy
        ;                               // CAUTION! no loop time limit implemented
    IBCR = BM_IBEN | BM_MSSL | BM_TXRX; // transmit mode, master (issue START cond.)
    while((IBSR & BM_IBB) == 0)         // wait for busy state
        ;                               // CAUTION! no loop time limit implemented
    }
```

```
//-------------------------------------------------------------------------

// Func: Issue IIC Restart Condition
// Args: -
// Retn: -
//
void restartIIC(void) {

    IBCR |= BM_RSTA;                    // issue RESTART condition
    }

//-------------------------------------------------------------------------

// Func: Issue IIC Stop Condition
// Args: -
// Retn: -
//
void stopIIC(void) {

    IBCR = BM_IBEN;                     // back to slave mode (issue STOP cond.)
    }

//-------------------------------------------------------------------------

// Func: Transmit byte via IIC
// Args: bval: data byte to transmit
// Retn: if stat==0 then IIC_ACK else IIC_NOACK
//
UINT8 sendIIC(UINT8 bval) {

    UINT8 stat;

//  IBCR = BM_IBEN | BM_MSSL | BM_TXRX; // still transmit mode, still master
    IBDR = bval;                        // transmit byte
    while((IBSR & BM_IBIF) == 0)        // wait for transfer done
        ;                               // CAUTION! no loop time limit implemented
    stat = IBSR & BM_RXAK;              // mask ACK status (0==ACK)
    IBSR = BM_IBIF;                     // clear IB Intr Flag
    return stat;
    }

//-------------------------------------------------------------------------

// Func: Receive byte from IIC
// Args: ack = IIC_ACK / IIC_NOACK
// Retn: byte received
//
UINT8 receiveIIC(UINT8 ack) {

    UINT8 bval;

    IBCR = BM_IBEN | BM_MSSL;           // receive mode (still master)
    if(ack != IIC_ACK) IBCR |= BM_TXAK; // set TXAK to respond with NOACK
    bval = IBDR;                        // dummy read initiates transfer
    while((IBSR & BM_IBIF) == 0)        // wait for transfer done
        ;                               // CAUTION! no loop time limit implemented
    IBSR = BM_IBIF;                     // clear IB Intr Flag
    IBCR = BM_IBEN | BM_MSSL | BM_TXRX; // back to transmit mode, still master
    bval = IBDR;                        // get received byte
    return bval;
    }

//=========================================================================
```

Die IIC-Bussignale stehen an X5/47+48 zur Verfügung.

CAN-Interface

Der MC9S12D64 verfügt über ein integriertes CAN-Modul (CAN0). Es kommuniziert über die Portpins PM0 und PM1 mit einem on-board CAN-Interface Chip (IC4), welcher das physische Businterface bildet. Die CAN-Bussignale CANH und CANL sind dann an X2 abzugreifen.

Wenn das CardS12 Modul der letzte Knoten am CAN-Bus ist, wird eine Terminierung erforderlich. Sie kann durch Verbinden der Pins X2/1 und X2/2 aktiviert werden.

Wird CAN0 in der Anwendung nicht benutzt, kann der Ausgang Pin4 des Treibers IC4 durch Öffnen der Lötbrücke BR9 vom Prozessor abgekoppelt werden. PM0 (wie auch PM1) kann dann als zusätzliches universelles I/O-Portpin verwendet werden (zugänglich an X5/41 bzw. X5/42).

Das folgende Listing illustriert einige grundlegende Basisfunktionen für die Kommunikation über den CAN-Bus:

```
//============================================================================
// File: S12_CAN.C - V1.01
//============================================================================

//-- Includes ----------------------------------------------------------------

#include "datatypes.h"
#include <mc9s12d64.h>
#include "s12_can.h"

//-- Defines -----------------------------------------------------------------

//-- Variables ---------------------------------------------------------------

//-- Code --------------------------------------------------------------------

// Func: initialize CAN
// Args: -
// Retn: -
// Note: -
//
void initCAN0(UINT16 idar, UINT16 idmr) {

    CAN0CTL0 = BM_INITRQ;               // request Init Mode
    while((CAN0CTL1 & BM_INITAK) == 0) ;// wait until Init Mode is established

    // set CAN enable bit, deactivate listen-only mode and
    // use Oscillator Clock (16MHz) as clock source
    CAN0CTL1 = BM_CANE;

    // set up timing parameters for 125kbps bus speed and sample
    // point at 87.5% (complying with CANopen recommendations):
    // fOSC = 16MHz; prescaler = 8 -> 1tq = (16MHz / 8)^-1 = 0.5µs
    // tBIT = tSYNCSEG + tSEG1 + tSEG2 = 1tq + 13tq + 2tq = 16tq = 8µs
    // fBUS = tBIT^-1 = 125kbps
    CAN0BTR0 = 0x07;    // sync jump width = 1tq, br prescaler = 8
    CAN0BTR1 = 0x1c;    // one sample point, tSEG2 = 2tq, tSEG1 = 13tq
```

```
    // we are going to use four 16-bit acceptance filters:
    CAN0IDAC = 0x10;

    // set up acceptance filter and mask register #1:
    // --------------------------------------------------------------------
    //    7   6   5   4   3   2   1   0 |   7   6   5   4   3   2   1   0
    // ID10 ID9 ID8 ID7 ID6 ID5 ID4 ID3 | ID2 ID1 ID0 RTR IDE xxx xxx xxx
    // --------------------------------------------------------------------
    // we are going to detect data frames with standard identifier (11 bits)
    // only, so bits RTR (bit4) and IDE (bit3) have to be clear
    CAN0IDAR0 = idar >> 8;              // top 8 of 11 bits
    CAN0IDAR1 = idar & 0xe0;            // remaining 3 of 11 bits
    CAN0IDMR0 = idmr >> 8;              // top 8 of 13 bits
    CAN0IDMR1 = (idmr & 0xe0) | 0x07;   // remaining 3 bits + RTR + IDE

    // set up acceptance filter and mask register #2,3,4 just as #1
    CAN0IDAR6 = CAN0IDAR4 = CAN0IDAR2 = CAN0IDAR0;
    CAN0IDAR7 = CAN0IDAR5 = CAN0IDAR3 = CAN0IDAR1;
    CAN0IDMR6 = CAN0IDMR4 = CAN0IDMR2 = CAN0IDMR0;
    CAN0IDMR7 = CAN0IDMR5 = CAN0IDMR3 = CAN0IDMR1;

    CAN0CTL0 &= ~BM_INITRQ;             // exit Init Mode
    while((CAN0CTL1 & BM_INITAK) != 0) ;// wait until Normal Mode is established
    CAN0TBSEL = BM_TX0;                 // use (only) TX buffer 0
    }

//--------------------------------------------------------------------------

BOOL testCAN0(void) {

    if((CAN0RFLG & BM_RXF) == 0) return FALSE;
    return TRUE;
    }

//--------------------------------------------------------------------------

UINT8 getCAN0(void) {

    UINT8 c;

    while((CAN0RFLG & BM_RXF) == 0) ;   // wait until CAN RX data pending
    c = *(CAN0RXFG+4);                  // save data
    CAN0RFLG = BM_RXF;                  // clear RX flag
    return c;
    }

//--------------------------------------------------------------------------

void putCAN0(UINT16 canid, UINT8 c) {

    while((CAN0TFLG & BM_TXE0) == 0) ;  // wait until Tx buffer released

    *(CAN0TXFG+0) = canid >> 8;         // destination address
    *(CAN0TXFG+1) = canid & 0xe0;
    *(CAN0TXFG+4) = c;
    *(CAN0TXFG+12) = 1;                 // one byte data
    *(CAN0TXFG+13) = 0;                 // priority = 0 (highest)

    CAN0TFLG = BM_TXE0;                 // initiate transfer
    }

//==========================================================================
```

7. Applikationshinweise

Verhalten nach Reset

Sobald die Resetleitung des Controllers freigegeben wird, holt sich die MCU die Information, an welcher Adresse das Programm des Anwenders beginnt. Der Controller liest hierzu den Resetvektor von den Speicherzellen $FFFE und $FFFF und springt dann an die dort angegebene Programmadresse.

Im Auslieferungszustand der CardS12 ist im Flash-Bootblock ($F000-$FFFF) das Monitorprogramm TwinPEEKs abgelegt. Der Resetvektor verweist auf den Beginn dieses Monitorprogramms. In Folge dessen startet nach jedem Reset automatisch TwinPEEKs (weitere Erläuterungen: siehe Monitorbeschreibung).

Startup-Code

Jede Controllerfirmware beginnt mit einer Reihe von Anweisungen zur Initialisierung der Hardware. Im Fall der CardS12 beschränken sich die *unbedingt notwendigen* Initialisierungen auf das Setzen des Stackpointers.

Die Abschaltung (bzw. ggf. die geeignete Initialisierung) des Watchdogs war bei früheren HC12-Derivaten zwingend notwendig. Beim MC9S12Dxx hingegen ist der Watchdog nach Reset zunächst stets disabled.

Zusatzinformationen im Web

Wenn zusätzliche Informationen zu Hard- und Software der CardS12 vorliegen, veröffentlichen wir diese auf unserer Website:

http://elmicro.com/de/cards12.html

8. Monitorprogramm TwinPEEKs

Software Version 2.3

Serielle Kommunikation

TwinPEEKs kommuniziert über die erste RS232 Schnittstelle ("SER0", X3) mit **19200 Baud**. Weitere Einstellungen: 8N1, kein Hardware- oder Softwarehandshake, kein Protokoll.

Autostart Funktion

Der TwinPEEKs Monitor überprüft nach Reset, ob die Port Pins PH6 und PH7 miteinander verbunden sind. Ist das der Fall, springt der Monitor zur Adresse $8000.

Durch dieses Feature wird es möglich, ein Anwenderprogramm automatisch zu starten, ohne den Resetvektor im geschützten Flash Boot Block ändern zu müssen.

Schreibzugriffe auf Flash und EEPROM

Die CPU kann auf alle Ressourcen des Mikrocontrollers byteweise lesend zugreifen. Der Speichertyp spielt dabei keine Rolle. Bei Schreibzugriffen sind jedoch Besonderheiten zu beachten: Flash und EEPROM müssen vor der Programmierung gelöscht werden, die Programmierung erfolgt wortweise und der Zugriff muss stets auf eine gerade Wortadressen stattfinden.

Deshalb müssen zwei aufeinander folgende Einzelbytes zunächst zu einem Wort zusammengefaßt werden, welches "aligned", d.h. auf eine Wortgrenze ausgerichtet sein muss. TwinPEEKs berücksichtigt dies, kann jedoch das folgende Problem nicht verhindern:

Der Monitor verarbeitet S-Record Daten stets zeilenweise. Falls die letzte belegte Adresse in einer solchen S-Record-Zeile gerade ist, fehlt zunächst das für die Wort-Programmierung erforderliche zweite Byte.

TwinPEEKs ergänzt in dieser Situation ein $FF-Byte und kann nun das Datenwort programmieren.

Setzt sich der Datenstrom in der folgenden S-Record Zeile mit dem zuvor fehlenden Byte fort, müsste der Monitor an der fraglichen Wortadresse einen erneuten Schreibzugriff vornehmen, was jedoch nicht zulässig ist. Es kommt zu einem Schreibfehler ("not erased").

Es ist daher notwendig, S-Record Daten vor der Programmierung auf gerade Adressen auszurichten. Hierzu kann z.B. das frei erhältliche Freescale Tool SRECCVT verwendet werden:

```
SRECCVT -m 0x00000 0xfffff 32 -o <outfile> <infile>
```

Die Syntax ist im SRECCVT Reference Guide (PDF) beschrieben.

Redirected Interrupt Vectors

Die Interruptvektoren des HCS12 liegen am Ende des 64KB 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 zur Laufzeit (vor der globalen Interruptfreigabe!), indem es einen Sprungbefehl in den RAM-Pseudovektor einträgt. Um z.B. den IRQ Interrupt nutzen zu können, muß ein Anwenderprogramm folgende Schritte ausführen:

```
ldaa #$06          ; JMP opcode to
staa $3FEE         ; IRQ pseudo vector
ldd  #isrFunc      ; ISR address to
std  $3FEF         ; IRQ pseudo vector + 1
```

Für C-Programme läßt sich eine Codesequenz nach folgendem Muster verwenden:

```
// install IRQ pseudo vector in RAM
// (if running with TwinPEEKs monitor)
*((unsigned char *)0x3fee) = 0x06;    // JMP opcode
*((void (**)(void))0x3fef) = isrFunc;
```

Der folgende Ausschnitt aus dem Assemblerlisting des Monitorprogramms dokumentiert die Adressen der umgeleiteten Interruptvektoren (erste Spalte von links: ursprüngliche Vektoradresse, zweite Spalte: Adresse im RAM):

```
FF80 : 3F43             dc.w  TP_RAMTOP-189         ; reserved
FF82 : 3F46             dc.w  TP_RAMTOP-186         ; reserved
FF84 : 3F49             dc.w  TP_RAMTOP-183         ; reserved
FF86 : 3F4C             dc.w  TP_RAMTOP-180         ; reserved
FF88 : 3F4F             dc.w  TP_RAMTOP-177         ; reserved
FF8A : 3F52             dc.w  TP_RAMTOP-174         ; reserved
FF8C : 3F55             dc.w  TP_RAMTOP-171         ; PWM Emergency Shutdown
FF8E : 3F58             dc.w  TP_RAMTOP-168         ; Port P
FF90 : 3F5B             dc.w  TP_RAMTOP-165         ; CAN4 transmit
FF92 : 3F5E             dc.w  TP_RAMTOP-162         ; CAN4 receive
FF94 : 3F61             dc.w  TP_RAMTOP-159         ; CAN4 errors
FF96 : 3F64             dc.w  TP_RAMTOP-156         ; CAN4 wake-up
FF98 : 3F67             dc.w  TP_RAMTOP-153         ; CAN3 transmit
FF9A : 3F6A             dc.w  TP_RAMTOP-150         ; CAN3 receive
FF9C : 3F6D             dc.w  TP_RAMTOP-147         ; CAN3 errors
FF9E : 3F70             dc.w  TP_RAMTOP-144         ; CAN3 wake-up
FFA0 : 3F73             dc.w  TP_RAMTOP-141         ; CAN2 transmit
FFA2 : 3F76             dc.w  TP_RAMTOP-138         ; CAN2 receive
FFA4 : 3F79             dc.w  TP_RAMTOP-135         ; CAN2 errors
FFA6 : 3F7C             dc.w  TP_RAMTOP-132         ; CAN2 wake-up
FFA8 : 3F7F             dc.w  TP_RAMTOP-129         ; CAN1 transmit
FFAA : 3F82             dc.w  TP_RAMTOP-126         ; CAN1 receive
FFAC : 3F85             dc.w  TP_RAMTOP-123         ; CAN1 errors
FFAE : 3F88             dc.w  TP_RAMTOP-120         ; CAN1 wake-up
FFB0 : 3F8B             dc.w  TP_RAMTOP-117         ; CAN0 transmit
FFB2 : 3F8E             dc.w  TP_RAMTOP-114         ; CAN0 receive
FFB4 : 3F91             dc.w  TP_RAMTOP-111         ; CAN0 errors
FFB6 : 3F94             dc.w  TP_RAMTOP-108         ; CAN0 wake-up
FFB8 : 3F97             dc.w  TP_RAMTOP-105         ; FLASH
FFBA : 3F9A             dc.w  TP_RAMTOP-102         ; EEPROM
FFBC : 3F9D             dc.w  TP_RAMTOP-99          ; SPI2
FFBE : 3FA0             dc.w  TP_RAMTOP-96          ; SPI1
FFC0 : 3FA3             dc.w  TP_RAMTOP-93          ; IIC
FFC2 : 3FA6             dc.w  TP_RAMTOP-90          ; BDLC
FFC4 : 3FA9             dc.w  TP_RAMTOP-87          ; Self Clock Mode
FFC6 : 3FAC             dc.w  TP_RAMTOP-84          ; PLL Lock
FFC8 : 3FAF             dc.w  TP_RAMTOP-81          ; Pulse Accu B Overflow
FFCA : 3FB2             dc.w  TP_RAMTOP-78          ; MDCU
FFCC : 3FB5             dc.w  TP_RAMTOP-75          ; Port H
FFCE : 3FB8             dc.w  TP_RAMTOP-72          ; Port J
FFD0 : 3FBB             dc.w  TP_RAMTOP-69          ; ATD1
FFD2 : 3FBE             dc.w  TP_RAMTOP-66          ; ATD0
FFD4 : 3FC1             dc.w  TP_RAMTOP-63          ; SCI1
FFD6 : 3FC4             dc.w  TP_RAMTOP-60          ; SCI0
FFD8 : 3FC7             dc.w  TP_RAMTOP-57          ; SPI0
FFDA : 3FCA             dc.w  TP_RAMTOP-54          ; Pulse Accu A Input Edge
FFDC : 3FCD             dc.w  TP_RAMTOP-51          ; Pulse Accu A Overflow
FFDE : 3FD0             dc.w  TP_RAMTOP-48          ; Timer Overflow
FFE0 : 3FD3             dc.w  TP_RAMTOP-45          ; TC7
FFE2 : 3FD6             dc.w  TP_RAMTOP-42          ; TC6
FFE4 : 3FD9             dc.w  TP_RAMTOP-39          ; TC5
FFE6 : 3FDC             dc.w  TP_RAMTOP-36          ; TC4
FFE8 : 3FDF             dc.w  TP_RAMTOP-33          ; TC3
FFEA : 3FE2             dc.w  TP_RAMTOP-30          ; TC2
FFEC : 3FE5             dc.w  TP_RAMTOP-27          ; TC1
FFEE : 3FE8             dc.w  TP_RAMTOP-24          ; TC0
FFF0 : 3FEB             dc.w  TP_RAMTOP-21          ; RTI
FFF2 : 3FEE             dc.w  TP_RAMTOP-18          ; IRQ
FFF4 : 3FF1             dc.w  TP_RAMTOP-15          ; XIRQ
FFF6 : 3FF4             dc.w  TP_RAMTOP-12          ; SWI
FFF8 : 3FF7             dc.w  TP_RAMTOP-9           ; Illegal Opcode
FFFA : 3FFA             dc.w  TP_RAMTOP-6           ; COP Fail
FFFC : 3FFD             dc.w  TP_RAMTOP-3           ; Clock Monitor Fail
FFFE : F000             dc.w  main                  ; Reset
```

Benutzungshinweise

Ein Monitorkommando besteht aus einem einzelnen Buchstaben, ggf. gefolgt von einer Liste von Argumenten. Alle Zahlenangaben erfolgen hexadezimal ohne weitere Vor- oder Nachsätze. Groß- und Kleinschreibung ist gleichermaßen zulässig.

Der für die CPU sichtbare Adreßraum umfaßt 64KB, die Adreßargumente sind demzufolge maximal vierstellig. 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.

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

Monitorbefehle

Blank Check

`Syntax: B`

Prüft, ob der gesamte Flash Memory (exkl. Monitorbereich) gelöscht ist. Falls dies *nicht* der Fall ist, wird die Nummer der ersten Page ausgegeben, in ein Byte ungleich $FF gefunden wurde.

Dump Memory

`Syntax: D [adr1 [adr2]]`

Anzeige des Speicherinhaltes ab Adresse adr1 bis Adresse adr2. Ohne Angabe einer Endadresse werden die folgenden $40 Bytes angezeigt. Der Inhalt von adr1 wird im Listing hervorgehoben.

Edit Memory

Syntax: E [addr {byte}]

Speicher editieren. Nach der Startadresse addr können bis zu vier Datenytes {byte} angegeben werden (ermöglicht Word- und Doubleword-Writes). Die Daten werden unmittelbar geschrieben, danach kehrt die Funktion zur Eingabeaufforderung zurück.

Sind keine Daten {byte} in der Eingabezeile angegeben, wird der interaktive Modus gestartet. Der Monitor erkennt, wenn Speicherbereiche nur wortweise verändert werden können (Flash/EEPROM) und verwendet/erwartet in diesem Fall 16Bit-Daten. Der interaktive Edit-Mode kann durch Eingabe von "Q" beendet werden. Weitere Befehle sind:

```
<ENTER>  nächste Adresse
   -     vorhergehende Adresse
   =     gleiche Adresse
   .     Ende (wie Q)
```

Fill Memory

Syntax: F adr1 adr2 byte

Füllt den Speicherbereich ab Adresse adr1 bis (exklusive) adr2 mit dem Wert byte.

Goto Address

Syntax: G [addr]

Ruft das Anwenderprogramm ab Adresse addr auf. Ein Rücksprung zum Monitor ist nicht vorgesehen.

Help

Syntax: H

Listet eine Kurzübersicht zu allen Monitorkommandos auf.

System Info

Syntax: I

Zeigt die Start- und Endadressen von Registerblock, RAM, EEPROM und Flash der MCU an und gibt die Prozessorkennung (PARTID) aus.

Load

Syntax: L

Lädt eine S-Record Datei in den Speicher. Es werden Daten-Records vom Typ S1 (16-Bit MCU-Adressen) und S2 (lineare 24-Bit Adressen) verarbeitet. S0-Records (Kommentarzeilen) werden übersprungen. S8- bzw. S9 Records werden als End-of-File-Markierung erkannt.

S2-Records verwenden lineare Adressen gemäß Freescale-Empfehlung. Der gültige Adressbereich startet für den MC9S12D64 bei 0xF0000 (0x3C * 16KB) und endet bei 0xFFFFF (0x40 * 16 KB - 1).

Beim Laden in nichtflüchtige Speicher (EEPROM, Flash) muß dieser Speicher zunächst gelöscht werden. Außerdem ist zu beachten, daß der Schreibzugriff nur wortweise erfolgen kann. Die S-Record Daten müssen ggf. entsprechend vorbereitet werden, um das Alignment zu gewährleisten (vergl. Erläuterung oben).

Das sendende Terminal (z.B. OC-Console) muß nach jeder übertragenen S-Record Zeile auf die Empfangsbestätigung (*) warten, um die Übertragungsgeschwindigkeit mit der Programmiergeschwindigkeit zu synchronisieren.

Move Memory

Syntax: M adr1 adr2 adr3

Kopiert den Speicherbereich ab Adresse addr1 bis (exklusive) Adresse addr2 nach Adresse addr3 und folgende.

Select PPAGE

Syntax: P [page]

Selektiert eine Program Page (PPAGE). Diese Page wird daraufhin im 16KB-Page-Window von $8000 bis $BFFF sichtbar.

Erase Flash

Syntax: X [page]

Löscht die angegebene Page (16KB) des Flashspeichers.

Ohne Angabe von page löscht der Befehl den gesamten Flash, abgesehen vom Monitorcode (zum Überschreiben des Monitors ist ein BDM-Tool wie ComPOD12/StarProg erforderlich).

Erase EEPROM

Syntax: Y [sadr]

Löscht den an Adresse sadr liegenden EEPROM-Sektor. Ein EEPROM-Sektor umfasst ein Doubleword (4 Byte). Bit 0 und 1 von sadr sind daher "don't care".

Ohne Angabe von sadr löscht der Befehl den gesamten EEPROM.

9. Memory Map

Die Memory Map des Controllers wird von TwinPEEKs wie folgt initialisiert (Achtung - z.T. abweichend von den Reset-Defaults!):

CardS12.D64

Start	Ende	Belegung
$0000	$03FF	Steuerregister
$0400	**$07FF**	1KB EEPROM (die oberen 16 Bytes sind stets reserviert!)
$3000	$3FFF	4KB RAM (Default nach Reset: $0000-$0FFF) TwinPEEKs verwendet die oberen 512 Bytes
$4000	$7FFF	16KB Flash (identisch mit Page $3E)
$8000	$BFFF	16KB Flash Page $3C (Page **$3C**..$3F mittels PPAGE frei wählbar)
$C000	$FFFF	16KB Flash (identisch mit Page $3F) TwinPEEKs verwendet die oberen 4KB

CardS12.DG256

Start	Ende	Belegung
$0000	$03FF	Steuerregister
$0400	**$0FFF**	3KB (von 4KB) EEPROM (die unteren 1024 Bytes sind durch die Steuerregister verdeckt, die obersten 16 Bytes sind stets reserviert!)
$1000	$3FFF	12KB RAM TwinPEEKs verwendet die oberen 512 Bytes
$4000	$7FFF	16KB Flash (identisch mit Page $3E)
$8000	$BFFF	16KB Flash Page $30 (Page **$30**..$3F mittels PPAGE frei wählbar)
$C000	$FFFF	16KB Flash (identisch mit Page $3F) TwinPEEKs verwendet die oberen 4KB

CardS12.DP512

Start	Ende	Belegung
$0000	$03FF	Steuerregister
$0400	**$07FF**	1KB (von 4KB) EEPROM (die unteren 1024 Bytes sind durch die Steuerregister verdeckt, die oberen 2048 Bytes durch das RAM!)
$0800	$3FFF	14KB RAM TwinPEEKs verwendet die oberen 512 Bytes
$4000	$7FFF	16KB Flash (identisch mit Page $3E)
$8000	$BFFF	16KB Flash Page $20 (Page **$20**..$3F mittels PPAGE frei wählbar)
$C000	$FFFF	16KB Flash (identisch mit Page $3F) TwinPEEKs verwendet die oberen 4KB

Hinweis:

Bedingt durch ein Erratum des MC9S12DP512 Mask Set 4L00M (und frühere) ist nicht nur der Monitorcode in der Page $3F, sondern zusätzlich auch der Adreßbereich $B000 bis $BFFF in der Page $3B schreibgeschützt. An diese Stelle kann der Monitor demzufolge keinen Usercode laden.

Der gesamte Flash (auch der schreibgeschützte Bereich) kann jedoch jederzeit mit Hilfe eines BDM-Tools programmiert werden.

Anhang

Literatur

[1] Kreidl, Kupris, Thamm: Mikrocontroller-Design
Hardware- und Software-Entwicklung mit dem 68HC12/HCS12;
Carl Hanser Verlag; 2003

S-Record Format

Das von Freescale 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", "S2", "S8", "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 Hexziffernpaar 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 bei S1-Records 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 bei S1-Records, (LEN-4) bei S2-Records.

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.

Records vom Typ S2 enthalten ebenfalls Daten. Im Gegensatz zu S1-Records kommen bei S2-Records 24-Bit Adressen zum Einsatz. Demzufolge umfaßt das Adressfeld 6 statt 4 Stellen. Werden S2-Records verwendet, um Pagingdaten für den HC(S)12 zu definieren, errechnet sich die lineare 24-Bit Adresse wie folgt:

```
ADDR24 = PAGE * 0x4000 + OFFSET
```

Neben den S1- und S2-Records, welche die eigentlichen Daten enthalten, werden S9- bzw. S8-Records als End-of-File Markierung verwendet. Abgesehen von dieser Terminierungs-Funktion kann in diesen Records die Startadresse des Programms vermerkt werden. Der Aufbau des S9-Records entspricht dem S1 Typ (S8 analog S9), wobei jedoch das Feld DAT leer bleibt. Das Feld ADDR spezifiziert die Startadresse des Programms. 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-Zertifizierungszeichen 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.