ChipS12

Hardware Version 1.11

Benutzerhandbuch

30. Mai 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

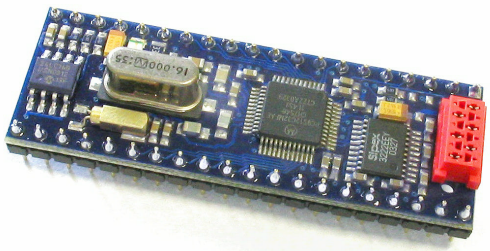
ChipS12 ist ein leistungsstarkes HCS12 Controller Modul mit kleinsten Abmessungen. Es kann einfach auf einen 40-poligen Sockel in der Anwenderschaltung gesteckt werden.

Das Modul kann wahlweise mit 3,3V oder 5V betrieben werden und ist somit geeignet für eine Vielzahl industrieller Anwendungen.

Zur schnellen Realisierung von Prototypen steht ein komplettes Entwicklungspaket zur Verfügung. Es enthält neben dem Controller Modul eine Trägerplatine (Carrier Board) mit LC-Display, die erforderlichen Anschlusskabel sowie Tools und Beispielsoftware auf CD-ROM.

Auf dem Modul ChipS12 kommt eine leistungsstarke MCU vom Typ MC9S12C128 zum Einsatz. Dieser Mikrocontroller enthält die 16-Bit HCS12 CPU, 128KB Flash, 4KB RAM und eine große Anzahl integrierter Peripheriefunktionen, wie SCI, SPI, CAN, Timer, PWM, ADC und Input-/Output-Kanäle. Der MC9S12C128 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.

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

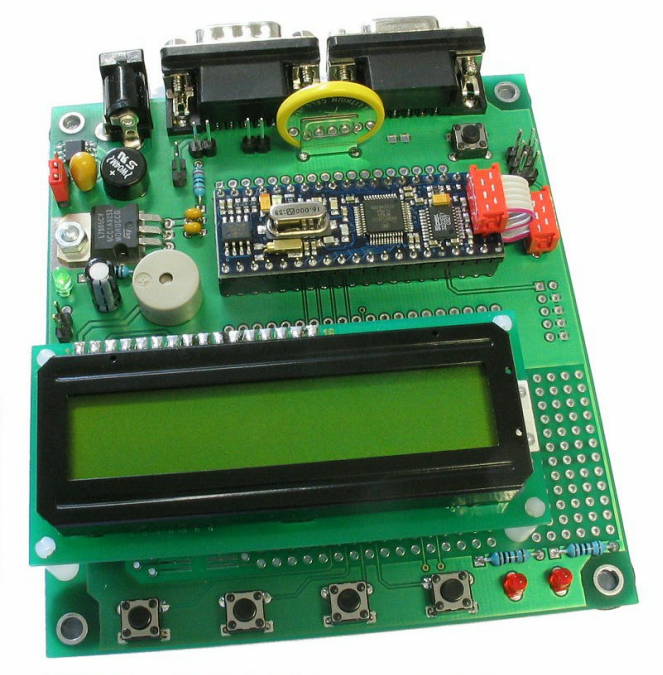


Technische Daten

- MCU MC9S12C128 im LQFP48 Gehäuse (SMD)
- HCS12 16-Bit CPU, Programmiermodell und Befehlssatz wie beim HC12
- 16 MHz Quarztakt, bis zu 25 MHz Bustakt über PLL
- 128 KB Flash
- 4 KB RAM
- 256 KBit serielles EEPROM
- 1x SCI - asynchrones serielles Interface (inkl. RS232-Treiber)
- 1x SPI - synchrones serielles Interface
- 1x msCAN-Modul (CAN 2.0A/B-kompatibel)
- High-Speed CAN Bustreiber (optional für 5V oder 3,3V)
- 8x 16-Bit Timer (Input Capture/Output Compare)
- 5x PWM (Pulse Width Modulator)
- 8-Kanal 10-Bit A/D-Wandler
- Integrierte LVI-Schaltung (Reset Controller)
- BDM-Anschluß für Download und Debugging
- Indikator-LED
- bis zu 26 freie Ein-/Ausgabeleitungen (je nach Nutzung anderer integrierter Peripheriefunktionen)
- Real Time Clock mit Datum, Zeit und Alarmfunktionen (optional)
- Betriebsspannung wahlweise 5V oder 3,3V (abhängig vom bestücktem CAN-Treiber), typ. Stromaufnahme 25 mA
- Abmessungen 2,0" x 0,7" x 0,5" (50,8mm x 17,8mm x 12,7mm)
- passt auf einen DIP40-Sockel

Lieferumfang des Entwicklungspakets

- Controller Modul mit MC9S12C128, inkl. Real Time Clock und 5V-CAN-Bustreiber
- TwinPEEKs Monitorprogramm (im Flash Speicher der MCU)
- Carrier Board mit DIP40-Sockel zum Aufstecken des ChipS12 Moduls, LC-Display und zahlreichen Peripheriefunktionen
- RS232 Anschlußkabel (Sub-D9)
- BDM-Verbindungskabel vom ChipS12 zum Carrier Board
- Hardwarehandbuch (dieses Dokument)
- Schaltplan
- CD-ROM: enthält Assemblersoftware, verschiedene Datenblätter, CPU12 Reference Manual, Softwarebeispiele, C-Compiler Demoversion u.v.m.



2. Schnellstart

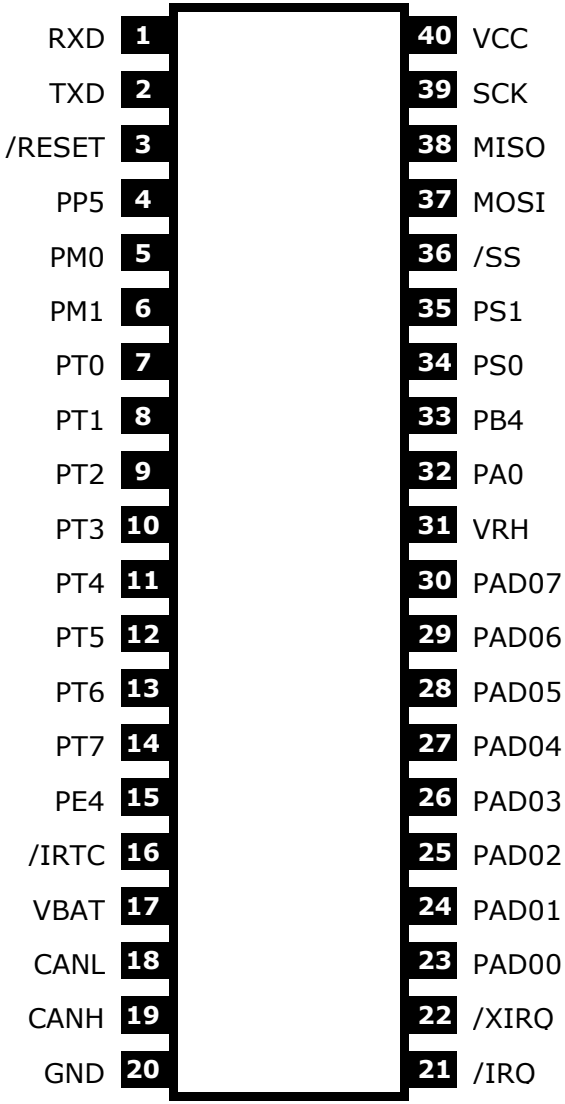
Kein Mensch liest gern dicke Handbücher. Daher hier die wichtigsten Hinweise in Kürze. Wenn Sie sich jedoch über ein Detail einmal nicht sicher sind, dann informieren Sie sich am Besten in den nachfolgenden Kapiteln.

Und so starten Sie mit dem ChipS12 Entwicklungspaket:

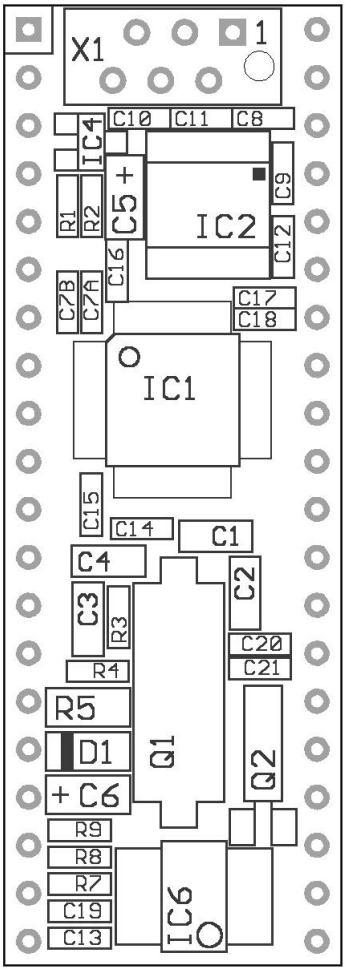
- Überprüfen Sie die Baugruppe (Carrier Board mit ChipS12 Modul) zunächst auf sichtbare Transportschäden.
- Prüfen Sie, ob das ChipS12 Modul korrekt auf dem Carrier Board steckt (die beiden roten BDM-Verbinder befinden sich unmittelbar nebeneinander).
- Verbinden Sie die Baugruppe via RS232 (Anschluß K1 auf dem Motherboard) mit Ihrem PC. Verwenden Sie das mitgelieferte Seriellkabel (Sub-D9, 1:1).
- 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 an Buchse K4 eine Gleichspannung von ca. 9V (8..15V) an, die Polarität ist beliebig.
- **Hinweis**: Unstabilisierte Steckernetzteile geben meist eine deutlich höhere Spannung ab, als (für volle Last) spezifiziert. Oft genügen daher bereits 6V oder 7,5V "nominal", um 9V "real" zu erhalten. Je höher die Eingangsspannung, desto mehr Wärme verbreitet der Spannungsregler VR1.
- Daraufhin leuchten LD1 auf dem Carrier Board und D1 auf dem ChipS12 Modul auf. Das Monitorprogramm startet und zeigt eine kurze Systemmeldung an. Mit Ausgabe des Promptzeichens erwartet es Ihre Anweisungen.

Wir wünschen Ihnen viel Erfolg bei der Arbeit mit ChipS12!

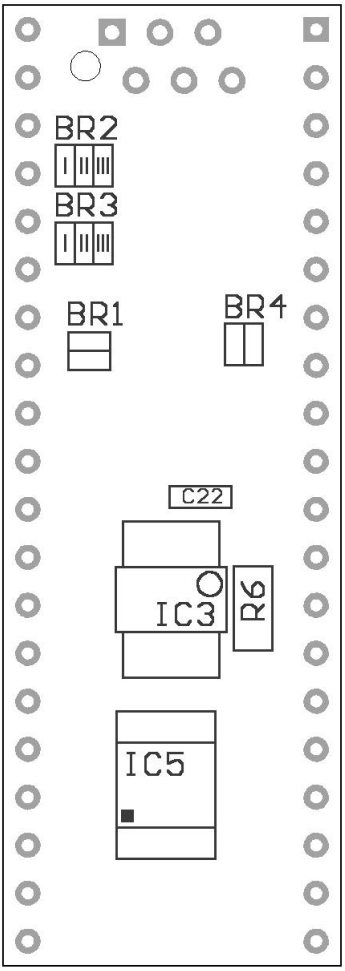
3. Anschlußbelegung



4. Bestückungsplan



Bestückungsseite



Platinenunterseite

5. Jumper und Lötbrücken

Jumper

Auf dem ChipS12 Modul 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: R1OUT

1-2*	RS232-Empfängerausgang R1OUT ist aktiv (treibt PS0 der MCU)
2-3	RS232-Empfängerausgang R1OUT ist hochohmig (PS0 anderweitig verwendbar)

BR3: SHDN

1-2*	RS232-Transceiver IC2 permanent in Betrieb
2-3	IC2 Suspend-Mode über PE4 der MCU steuerbar

BR4: RRTC

offen*	RTC-Reset disabled, /VDCC Ausgang der RTC (IC6) ungenutzt
geschl.	/VDCC-Ausgang der RTC verbunden mit /RESET; zusätzliche LVI-Funktion: RTC löst Reset aus bei Unterschreiten der Batterieschaltschwelle (siehe Datenblatt der RTC)

* = Standardeinstellung

6. Mechanische Abmessungen

Das ChipS12 Modul kann auf einen gewöhnlichen DIP40-Sockel aufgesteckt werden. Der Pinabstand beträgt 0,1" (2,54 mm) und der Reihenabstand 0,6" (15,24 mm).

Die Außenabmessungen des Moduls betragen 2,0" (50,8 mm) x 0,7" (17,78 mm).

7. 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 (Bezeichnung im Schaltplan: VCC) beträgt nominal 3 bis 5 Volt, intern arbeitet der Prozessor jedoch mit 2,5 Volt. Der hierzu erforderliche Spannungsregler ist bereits in der MCU integriert.

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 entsprechend VCC, 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 Keramikkondensator mit mindestens 100nF (C15, C16, C17) sowie

zusätzlich ein 10µF Elektrolytkondensator (C5). 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 sowie VDDPLL/VSSPLL weitere Keramikkondensatoren vorgesehen (C7A, C7B, C14). 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 Motorola 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. Zusätzlich ist im MC9S12C128 eine LVI-Schaltung enthalten, welche die Aufgabe hat, zuverlässig Reset auszulösen, sobald die Versorgungsspannung der MCU unter den zulässigen Mindestwert gefallen ist.

Optional kann IC4 als zusätzlicher, externer LVI-Schaltkreis eingesetzt werden. Der Ausgang des IC4 ist als Open-Drain Ausgang ausgeführt, um Kollisionen mit dem bidirektionalen Resetpin der MCU zu vermeiden. Im inaktiven Zustand stellt sich an /RESET H-Pegel ein. IC4 enthält dazu einen integrierten Pull-Up Widerstand (ca. 5kOhm). R1 hingegen entfällt bei Bestückung der Option IC4.

Der von IC4 generierte Resetimpuls hat eine Dauer von ca. 250ms (mindestens jedoch 140ms). Es ist wichtig zu bemerken, dass dieser Impuls nur bei einem Power-On-Ereignis wirksam wird. Die MCU-internen Resetimpulse werden von IC4 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 MC9S12Cxx 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 MC9S12Cxx verwendet einen Colpitts-Oszillator mit translated Ground. Der Hauptvorteil dieser Oszillatorschaltung ist eine sehr geringe Leistungsaufnahme, dafür ist die Komponentenwahl um einiges kritischer. Auf dem ChipS12 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 MC9S12Cxx 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. 5 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 (1/20–1/10) x C3 betragen, hier also 2,2nF. Das Kapitel "XFC Component Selection” im MC9S12DP256B Device User Guide illustriert die erforderlichen Rechenschritte.

Das folgende Listing zeigt die erforderlichen Initialisierungsschritte für die PLL:

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

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

#include <mc9s12dp512.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
    }

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

R4 dient dazu, /XCLKS während Reset auf H-Pegel zu halten, um die gewünschte Colpitts-Oszillatorkonfiguration auszuwählen. Hat /XCLKS während Reset L-Pegel, dann befindet sich der Oszillator des MC9S12Cxx im Pierce-Mode (andere Beschaltung erforderlich!). Alternativ zur Takterzeugung mit Q1 könnte dann über den EXTAL-Pin des MC9S12Cxx ein externer Takt eingespeist werden.

Achtung: die verschiedenen HCS12-Typen haben z.T. unterschiedliche Funktionalität hinsichtlich des /XCLKS-Pins.

Betriebsarten, BDM-Unterstützung

Drei Signale des HCS12 dienen der Auswahl der MCU-Betriebsart: MODA, MODB und BKGD (=MODC). MODA und MODB werden durch interne Widerstände auf L-Pegel gebracht, um Single Chip Mode auszuwählen. BKGD ist über R2 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 MC9S12Cxx verfügt über ein integriertes Analog/Digital-Wandler Modul mit einer Auflösung von max. 10 Bit. Das Modul (ATD) hat acht gemultiplexte Eingänge.

Das folgende Beispielprogramm zeigt die Initialisierungssequenz für das A/D-Wandler Modul ATD und eine Routine zum Erfassen des Spannungswertes eines einzelnen Eingangskanals. Weitere Beispielroutinen für das integrierte ATD-Modul sind in der Quelltextdatei S12_ATD.C enthalten.

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

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

#include "datatypes.h"
#include <mc9s12dp512.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;
    }

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

Die Referenzspannung VRH legt die obere Grenze der Eingangsspannung aller A/D-Kanäle fest, sie ist auf dem ChipS12 ab Werk über BR1 mit VDDA (VCC) verbunden. Durch Öffnen der Lötbrücke BR1 ist es möglich, über X0/31 eine externe Referenzspannung einzuspeisen.

Indikator-LED

An Portpin PE7 ist eine Indikator-LED (D1)angeschlossen. Die Steuerung der Indikator-LED kann durch einige einfache Makros erfolgen, wie das folgende Headerfile zeigt.

```
//===========================================================================
// File: CHIPS12_LED.H - V1.00
//===========================================================================

#ifndef __CHIPS12_LED_H
#define __CHIPS12_LED_H

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

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

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

/* module contains no code */

#endif //__CHIPS12_LED_H ==================================================
```

RS232-Schnittstelle

Der MC9S12Cxx verfügt über eine asynchrone Schnittstelle (SCI0) mit den beiden Signalleitungen RXD0 und TXD0. Hardware-Handshake-Leitungen sind nicht Bestandteil des SCI-Moduls des Controllers.

Die Signalleitungen der SCI-Schnittstelle sind mit dem RS232-Pegelwandler IC2 verbunden. Wird die RS232-Schnittstelle nicht benötigt, kann der Ausgang R1OUT des IC2 in den hochohmigen Zustand gebracht werden. Hierzu sind die Kontakte 2-3 der Lötbrücke BR2 zu verbinden. Die MCU-Signale PS0 und PS1 können dann bei Bedarf als zusätzliche I/O-Leitungen verwendet werden.

Zur Verringerung der Stromaufnahme kann IC2 in den Suspend-Mode versetzt werden. Hierzu ist Lötbrücke BR3 in Position 2-3 zu

versetzen. Nun kann das MCU-Signal PE4 zur Steuerung des /SHDN-Eingangs des RS232-Pegelwandlers verwendet werden. Ein L-Signal schaltet IC2 in den stromsparenden Suspend-Mode.

Achtung: PE4 kann mittels Software als Taktausgang (ECLK) konfiguriert werden. Dies muss unterbleiben, falls PE4 als Suspend-Steuersignal genutzt werden soll!

Das folgende Codebeispiel zeigt die Ansteuerung von SCI0 mittels Polling:

```
//============================================================================
// File: S12_SCI.C - V1.10
//============================================================================

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

#include "datatypes.h"
#include <mc9s12dp512.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
    }

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

BOOL testSCI0(void) {

    if((SCI0SR1 & BM_RDRF) == 0) return FALSE;
    return TRUE;
    }

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

UINT8 getSCI0(void) {

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

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

void putSCI0(UINT8 c) {

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

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

SPI-Bus

Der MC9S12C128 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 PM2 bis PM5. Diese Signale werden in der Schaltung des ChipS12 selbst nicht benutzt, sind aber am Steckverbinder X0 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 - V1.02
//============================================================================

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

#include "datatypes.h"
#include <mc9s12dp512.h>
#include "s12_spi.h"

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

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

    // set SS,SCK,MOSI lines to Output
    DDRM |= 0x38;                       // for HCS12C-Series
//  DDRS |= 0xe0;                       // for HCS12D-Series
    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 until transmitter available
    SPI0DR = abyte;                     // start transfer
    while((SPI0SR & BM_SPIF) == 0) ;    // wait until transfer finished
    return(SPI0DR);                     // read back data received
    }

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

IIC-Bus

Der MC9S12C128 verfügt nicht über ein Hardware-IIC-Modul. Zur Ansteuerung der boardeigenen Peripheriebausteine IC5 (RTC) und IC6 (serielles EEPROM) wird eine Software-Implementierung des IIC-Busprotokolls verwendet (siehe Datei S12_SIIC.S).

Das MCU-Signal PA0 dient als Datenleitung (SDA), PB4 wird als Clocksignal (SCL) genutzt. Diese Signale stehen auch zur externen Nutzung zusätzlicher IIC-Slavebausteine zur Verfügung.

Serielles EEPROM

Die MCU selbst enthält kein EEPROM, daher ist auf dem ChipS12 ein separates IC vorgesehen. Der Speicherbaustein IC6 hat eine Kapazität von 256 kBit und wird über ein IIC-Bus-Interface angesteuert.

Die Datei CHIPS12_SEEP.C enthält die grundlegenden Ansteuerroutinen und setzt auf dem oben beschriebenen Software-IIC-Modul auf:

```
//=============================================================================
// File: CHIPS12_SEEP.C - V1.01
//       for ChipS12 using 256kBit EEPROM 24LC256
//=============================================================================

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

#include "datatypes.h"
#include "s12_siic.h"
#include "chips12_seep.h"

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

// device signature of 24LC256 (8 bit left-justified value)
#define SEEP_DEVICE_ID  0xA0

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

static INT16 SEEP_ErrorCode;

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

void initSEEP(void) {

    SEEP_ErrorCode = SEEP_EC_OK;
    }

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

INT16 peekSEEP(UINT16 addr) {

    UINT8 b;
```

```
    SEEP_ErrorCode = SEEP_EC_OK;
    startIIC();
    if(sendIIC(SEEP_DEVICE_ID + IIC_WRITE) != IIC_ACK)
        SEEP_ErrorCode = SEEP_EC_NOTRDY;
    else {
        if(sendIIC((UINT8)((addr >> 8) & 0x7f)) != IIC_ACK)
            SEEP_ErrorCode = SEEP_EC_ADDRERR;
        else {
            if(sendIIC((UINT8)addr) != IIC_ACK)
                SEEP_ErrorCode = SEEP_EC_ADDRERR;
            else {
                restartIIC();
                if(sendIIC(SEEP_DEVICE_ID + IIC_READ) != IIC_ACK)
                    SEEP_ErrorCode = SEEP_EC_RDERR;
                else {
                    b = receiveIIC(IIC_NOACK);
                    }
                }
            }
        }
    stopIIC();
    if(SEEP_ErrorCode != SEEP_EC_OK)
        return SEEP_ErrorCode;
    return b;
    }

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

INT16 pokeSEEP(UINT16 addr, UINT8 bval) {

    SEEP_ErrorCode = SEEP_EC_OK;
    startIIC();
    if(sendIIC(SEEP_DEVICE_ID + IIC_WRITE) != IIC_ACK)
        SEEP_ErrorCode = SEEP_EC_NOTRDY;
    else {
        if(sendIIC((UINT8)((addr >> 8) & 0x7f)) != IIC_ACK)
            SEEP_ErrorCode = SEEP_EC_ADDRERR;
        else {
            if(sendIIC((UINT8)addr) != IIC_ACK)
                SEEP_ErrorCode = SEEP_EC_ADDRERR;
            else {
                if(sendIIC(bval) != IIC_ACK)
                    SEEP_ErrorCode = SEEP_EC_WRERR;
                }
            }
        }
    stopIIC();
    return SEEP_ErrorCode;
    }

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

INT16 getLastErrSEEP(void) {

    return SEEP_ErrorCode;
    }

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

Real Time Clock

Auf dem ChipS12 befindet sich als Bestückungsoption eine Real Time Clock (Echtzeituhr, RTC) vom Typ R2051. Diese wird über den IIC-Bus angesprochen und stellt eine Zeitreferenz inkl. Kalenderinformation bereit.

Die RTC ist in der Lage, zyklisch bzw. zu vorgegebenen Zeitpunkten einen Interrupt auszulösen. Der Open-Drain Ausgang /INTR steht zu diesem Zweck als Signal /IRTC am Anschluß X0/16 des Moduls zur Verfügung und kann extern mit einem der Interrupteingänge (/IRQ, /XIRQ oder einer der I/O-Pins) verbunden werden.

Um die Funktion der RTC auch bei Ausfall der Hauptversorgungsspannung VCC aufrecht zu erhalten, kann über den Modulanschluss VBAT (X0/17) eine Stützbatterie angeschlossen werden. Bevorzugt kommt hierfür eine LiMn-Primärzelle mit 3V Zellenspannung zum Einsatz. Die Umschaltung auf die Backup-Batterie wird von der RTC automatisch durchgeführt, sobald VCC unter einen Schwellwert von 2,4V sinkt. Zugleich wird unterhalb dieses Schwellwertes der /VDCC Ausgang der RTC betätigt. Schließt man die Lötbrücke BR4, kann dieses Signal als zusätzliche Resetquelle für die MCU verwendet werden.

Beispielroutinen zur Ansteuerung der RTC des ChipS12 sind in der Datei CHIPS12_RTC.C enthalten.

CAN-Interface

Der MC9S12C128 verfügt über ein integriertes CAN-Modul (CAN0). Es kommuniziert über die Portpins PM0 und PM1 mit einem on-board CAN-Interface Chip (IC3, Option), welcher das physische Businterface bildet. Die CAN-Bussignale CANH und CANL sind dann an X0 abzugreifen.

Wenn das ChipS12 Modul der letzte Knoten am CAN-Bus ist, wird eine Terminierung erforderlich. Hierzu ist extern ein Widerstand von 120 Ohm zwischen CANH und CANL vorzusehen.

R6 dient der Einstellung der Flankensteilheit (Slope Control) des Bussignals. Der installierte Wert (10k) muss für High-Speed Anwendungen ggf. verringert werden. Die Berechnungsgrundlagen hierzu sind im Herstellerdatenblatt enthalten.

Der CAN Bustreiber IC3 muss entsprechend der Betriebsspannung des Moduls ausgewählt werden. Bitte beachten Sie die Typangaben im Schaltplan.

Software zur Kommunikation über den CAN-Bus kann sehr umfangreich sein. Es existieren viele Varianten, insbesondere der höheren Protokollschichten. Dennoch läßt sich eine Verbindung zwischen CAN-Knoten bereits mit einfachen Mitteln relisieren, wie das folgende Beispiel zeigt:

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

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

#include "datatypes.h"
#include <mc9s12dp512.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
    }

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

8. 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 ChipS12 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 des ChipS12 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 MC9S12Cxx hingegen ist der Watchdog nach Reset zunächst stets disabled.

Zusatzinformationen im Web

Wenn zusätzliche Informationen zu Hard- und Software des ChipS12 vorliegen, veröffentlichen wir diese auf unserer Website:

http://elmicro.com/de/chips12.html

9. Monitorprogramm TwinPEEKs

Software Version 2.3

Serielle Kommunikation

TwinPEEKs kommuniziert über die RS232-Schnittstelle mit einer Schnittstellengeschwindigkeit von **19200 Baud**. Weitere Einstellungen: 8N1, kein Hardware- oder Softwarehandshake, kein Protokoll.

Autostart Funktion

Der TwinPEEKs Monitor überprüft nach Reset, ob die Port Pins PT2 und PT3 (X0/9+10) 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 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 EEPROM muß 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 Motorola 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 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 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 $0FEE         ; IRQ pseudo vector
ldd  #isrFunc      ; ISR address to
std  $0FEF         ; 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 *)0x0fee) = 0x06;    // JMP opcode
*((void (**)(void))0x0fef) = isrFunc;
```

Der folgende Ausschnitt aus dem Assemblerlisting des Monitorprogramms dokumentiert die Adressen der umgeleiteten Interruptvektoren. Die erste Spalte von links zeigt die ursprüngliche Vektoradresse, die zweite Spalte die Adresse des Pseudovektors im RAM (Anmerkung: welche Interruptvektoren tatsächlich belegt sind, richtet sich nach dem verwendeten HCS12-Derivat - Einzelheiten siehe Device Guide!):

```
FF80 : 0F43             dc.w  TP_RAMTOP-189         ; reserved
FF82 : 0F46             dc.w  TP_RAMTOP-186         ; reserved
FF84 : 0F49             dc.w  TP_RAMTOP-183         ; reserved
FF86 : 0F4C             dc.w  TP_RAMTOP-180         ; reserved
FF88 : 0F4F             dc.w  TP_RAMTOP-177         ; reserved
FF8A : 0F52             dc.w  TP_RAMTOP-174         ; reserved
FF8C : 0F55             dc.w  TP_RAMTOP-171         ; PWM Emergency Shutdown
FF8E : 0F58             dc.w  TP_RAMTOP-168         ; Port P
FF90 : 0F5B             dc.w  TP_RAMTOP-165         ; CAN4 transmit
FF92 : 0F5E             dc.w  TP_RAMTOP-162         ; CAN4 receive
FF94 : 0F61             dc.w  TP_RAMTOP-159         ; CAN4 errors
FF96 : 0F64             dc.w  TP_RAMTOP-156         ; CAN4 wake-up
FF98 : 0F67             dc.w  TP_RAMTOP-153         ; CAN3 transmit
FF9A : 0F6A             dc.w  TP_RAMTOP-150         ; CAN3 receive
FF9C : 0F6D             dc.w  TP_RAMTOP-147         ; CAN3 errors
FF9E : 0F70             dc.w  TP_RAMTOP-144         ; CAN3 wake-up
FFA0 : 0F73             dc.w  TP_RAMTOP-141         ; CAN2 transmit
FFA2 : 0F76             dc.w  TP_RAMTOP-138         ; CAN2 receive
FFA4 : 0F79             dc.w  TP_RAMTOP-135         ; CAN2 errors
FFA6 : 0F7C             dc.w  TP_RAMTOP-132         ; CAN2 wake-up
FFA8 : 0F7F             dc.w  TP_RAMTOP-129         ; CAN1 transmit
FFAA : 0F82             dc.w  TP_RAMTOP-126         ; CAN1 receive
FFAC : 0F85             dc.w  TP_RAMTOP-123         ; CAN1 errors
FFAE : 0F88             dc.w  TP_RAMTOP-120         ; CAN1 wake-up
FFB0 : 0F8B             dc.w  TP_RAMTOP-117         ; CAN0 transmit
FFB2 : 0F8E             dc.w  TP_RAMTOP-114         ; CAN0 receive
FFB4 : 0F91             dc.w  TP_RAMTOP-111         ; CAN0 errors
FFB6 : 0F94             dc.w  TP_RAMTOP-108         ; CAN0 wake-up
FFB8 : 0F97             dc.w  TP_RAMTOP-105         ; FLASH
FFBA : 0F9A             dc.w  TP_RAMTOP-102         ; EEPROM
FFBC : 0F9D             dc.w  TP_RAMTOP-99          ; SPI2
FFBE : 0FA0             dc.w  TP_RAMTOP-96          ; SPI1
FFC0 : 0FA3             dc.w  TP_RAMTOP-93          ; IIC
FFC2 : 0FA6             dc.w  TP_RAMTOP-90          ; BDLC
FFC4 : 0FA9             dc.w  TP_RAMTOP-87          ; Self Clock Mode
FFC6 : 0FAC             dc.w  TP_RAMTOP-84          ; PLL Lock
FFC8 : 0FAF             dc.w  TP_RAMTOP-81          ; Pulse Accu B Overflow
FFCA : 0FB2             dc.w  TP_RAMTOP-78          ; MDCU
FFCC : 0FB5             dc.w  TP_RAMTOP-75          ; Port H
FFCE : 0FB8             dc.w  TP_RAMTOP-72          ; Port J
FFD0 : 0FBB             dc.w  TP_RAMTOP-69          ; ATD1
FFD2 : 0FBE             dc.w  TP_RAMTOP-66          ; ATD0
FFD4 : 0FC1             dc.w  TP_RAMTOP-63          ; SCI1
FFD6 : 0FC4             dc.w  TP_RAMTOP-60          ; SCI0
FFD8 : 0FC7             dc.w  TP_RAMTOP-57          ; SPI0
FFDA : 0FCA             dc.w  TP_RAMTOP-54          ; Pulse Accu A Input Edge
FFDC : 0FCD             dc.w  TP_RAMTOP-51          ; Pulse Accu A Overflow
FFDE : 0FD0             dc.w  TP_RAMTOP-48          ; Timer Overflow
FFE0 : 0FD3             dc.w  TP_RAMTOP-45          ; TC7
FFE2 : 0FD6             dc.w  TP_RAMTOP-42          ; TC6
FFE4 : 0FD9             dc.w  TP_RAMTOP-39          ; TC5
FFE6 : 0FDC             dc.w  TP_RAMTOP-36          ; TC4
FFE8 : 0FDF             dc.w  TP_RAMTOP-33          ; TC3
FFEA : 0FE2             dc.w  TP_RAMTOP-30          ; TC2
FFEC : 0FE5             dc.w  TP_RAMTOP-27          ; TC1
FFEE : 0FE8             dc.w  TP_RAMTOP-24          ; TC0
FFF0 : 0FEB             dc.w  TP_RAMTOP-21          ; RTI
FFF2 : 0FEE             dc.w  TP_RAMTOP-18          ; IRQ
FFF4 : 0FF1             dc.w  TP_RAMTOP-15          ; XIRQ
FFF6 : 0FF4             dc.w  TP_RAMTOP-12          ; SWI
FFF8 : 0FF7             dc.w  TP_RAMTOP-9           ; Illegal Opcode
FFFA : 0FFA             dc.w  TP_RAMTOP-6           ; COP Fail
FFFC : 0FFD             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 abgewickelt. 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 Motorola-Empfehlung. Der gültige Adressbereich startet für den MC9S12C128 bei 0x0E0000 (0x38 * 16KB) und endet bei 0x0FFFFF (0x40 * 16 KB - 1).

Beim Laden in nichtflüchtige Speicher (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).

10. Memory Map

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

ChipS12.C32*

Start	Ende	Belegung
$0000	$03FF	Steuerregister
$0800	$0FFF	2KB RAM TwinPEEKs verwendet die oberen 512 Bytes
$4000	$7FFF	16KB Flash (identisch mit Page $3E)
$8000	$BFFF	16KB Flash Page $3E (Page **$3E** oder $3F mittels PPAGE frei wählbar)
$C000	$FFFF	16KB Flash (identisch mit Page $3F) TwinPEEKs verwendet die oberen 4KB

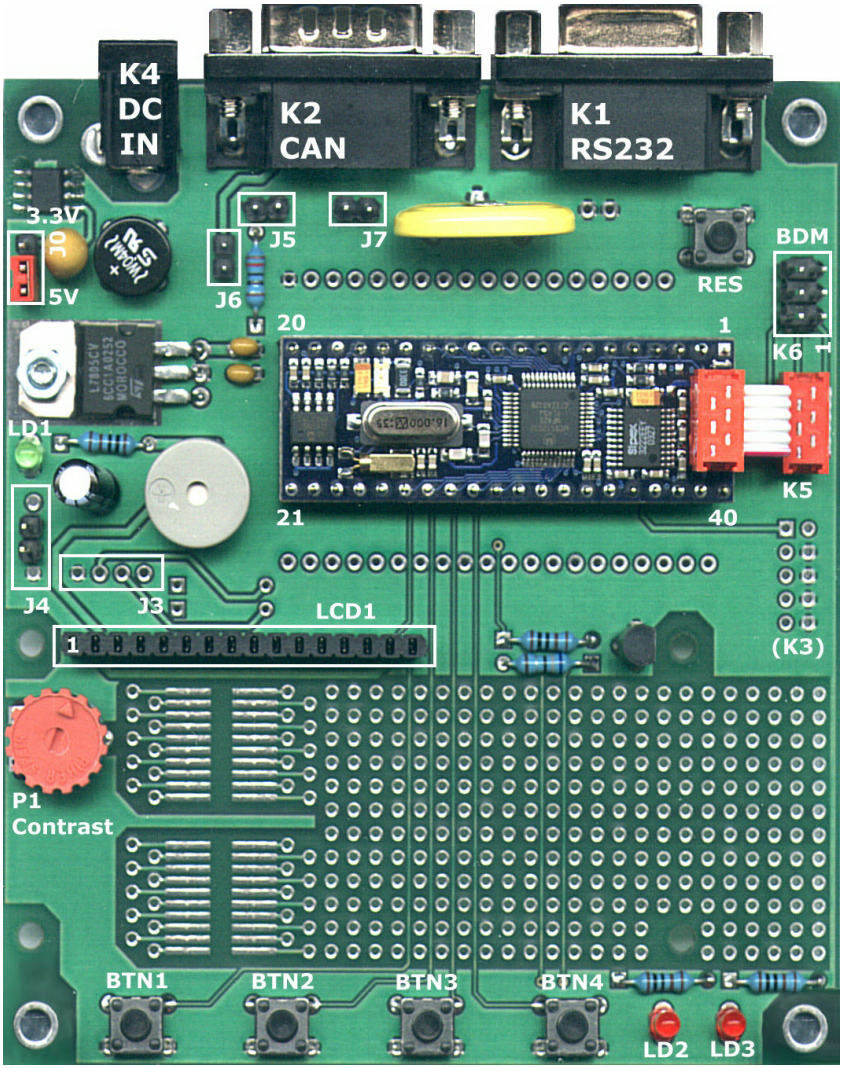
ChipS12.C128

Start	Ende	Belegung
$0000	$03FF	Steuerregister
$0400	$0FFF	4KB RAM, davon 3KB sichtbar (die unteren 1024 Bytes sind durch die Steuerregister verdeckt) TwinPEEKs verwendet die oberen 512 Bytes
$1000	$3FFF	16KB Flash (identisch mit Page $3D), davon 12KB sichtbar (die unteren 4KB sind durch RAM und Steuerregister verdeckt)
$4000	$7FFF	16KB Flash (identisch mit Page $3E)
$8000	$BFFF	16KB Flash Page $38 (Page **$38**..$3F mittels PPAGE frei wählbar)
$C000	$FFFF	16KB Flash (identisch mit Page $3F) TwinPEEKs verwendet die oberen 4KB

* ältere Boardversion (nicht mehr im Angebot)

11. Carrier Board

Lageplan



Jumper und Anschlüsse

J0	1-2	Betriebsspannung VCC = 3,3V (I_{CC} < 0,1A!)
	2-3*	Betriebsspannung VCC = 5V
J1	nicht bestückt	Anschluß (Lötpads) ChipS12 Pin 21..40
J2	nicht bestückt	Anschluß (Lötpads) ChipS12 Pin 3..20
J3	nicht bestückt	optional; Belegung wie J4; ggf. als I2C-Bus Anschluß nutzbar (erfordert Softwaretreiber)
J4	offen*	1=VCC, 2=PT2, 3=PT3, 4=GND wenn Pin 2 und Pin 3 während Reset verbunden sind, wird die Autostart-Funktion des Monitors aktiviert
J5	offen* geschl.	aktiviert die CAN-Bus Terminierung mit R5 (erforderlich an beiden Endpunkten des CAN-Busses)
J6	offen* geschl.	verbindet Versorgungsspannung VIN mit K2/9, zwecks Weitergabe der Versorgungsspannung an weitere CAN-Bus Teilnehmer
J7	offen* geschl.	verbindet Stützbatterie BAT1 mit VBAT Eingang des ChipS12 (RTC Backup Versorgung)
K1		RS232-Anschluß für Sub-D9 Kabel 1:1 zum PC
K2		CAN-Anschluß Sub-D9
K3	n. best.	optional; ggf. Verwendung als SPI-Port
K4		Anschluß für Steckernetzteil, Polarität beliebig, Arbeitsbereich ca. 8..15V, Gleichspannung
K5		BDM-Verbindung zum ChipS12 Modul
K6		BDM-Anschluß für Debugger (BDM-Pod)
LCD1		Steckverbinder für alphanum. LC-Display

* = Standardeinstellung

Schaltplan

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

Hinweise zur Stromversorgung

Der auf dem Carrier Board befindliche 3,3V-Spannungsregler VR2 kann maximal 100mA bereitstellen (siehe Datenblatt des LE33). Dieser Wert ist reichlich bemessen für das ChipS12 Modul an sich. Bei Anschluß zusätzlicher externer Komponenten könnte der Grenzwert jedoch leicht überschritten werden. In diesem Fall sollte unbedingt die Gesamtstromaufnahme überprüft werden (Pins 1 und 2 des Jumpers J8)!

Das LC-Display LCD1 wird stets mit 5V betrieben, auch wenn das Controllermodul mit 3,3V arbeitet. Die Ansteuerung des Moduls ist so gestaltet, daß ausschließlich Schreibzugriffe stattfinden. Der R/W-Eingang des LCDs ist zu diesem Zweck permanent mit L-Pegel verbunden. Die 3,3V-CMOS-Ausgänge des Mikrocontrollers liefern ausreichende Spannungspegel für die 5V-TTL-Eingänge des LC-Displays.

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 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", "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.