ChipS12

Hardware Version 1.11

User Manual

May 30 2008

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

Contents

1. Overview

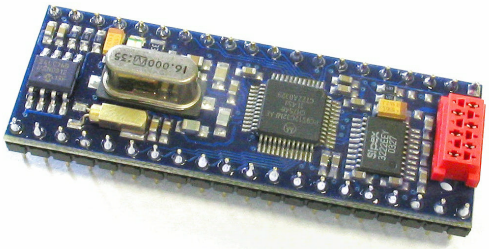
ChipS12 is a miniaturized controller module based on a powerful 16-bit HCS12 microcontroller. It can be easily plugged into a DIP40 socket on the user's application PCB.

The module can be operated with either 3.3V or 5V which makes it suited for a wide range of industrial applications.

A complete development package is available to kick start your development work. It contains a ChipS12 controller module, a carrier board including a large number of useful peripherals (such as LEDs, buttons, buzzer and LC-display) and a set of cables. Tool software, documentation and example programs are provided on a CD-ROM.

The ChipS12 is equipped with a MC9S12C128 microcontroller unit (MCU). It contains a 16-bit HCS12 CPU, 128KB of Flash memory, 4KB RAM and a large amount of peripheral function blocks, such as SCI, SPI, CAN, Timer, PWM, ADC and General-Purpose-I/Os. The MC9S12C128 has full 16-bit data paths throughout. An integrated PLL-circuit allows adjusting performance vs. current consumption according to the needs of the user application.

For HCS12 microcontrollers, a wide range of software tools (monitors, C-compilers, BDM-debuggers) is available to accelerate the development process.

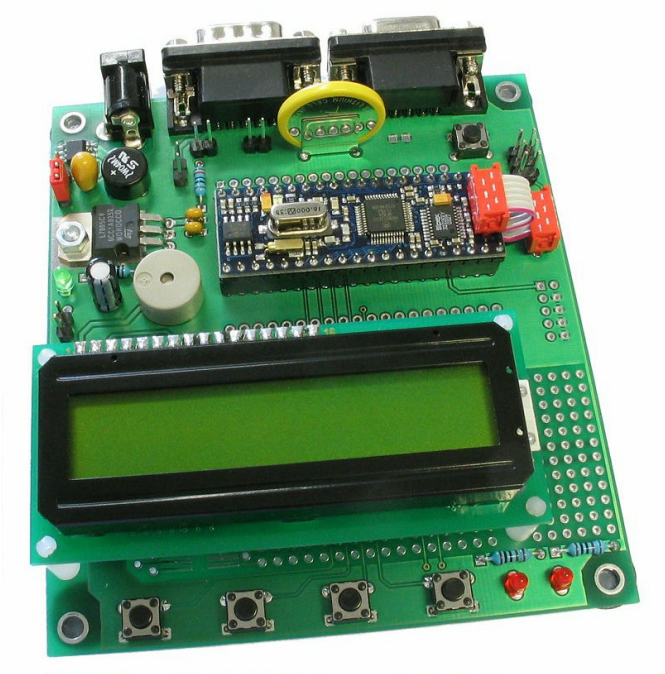


Technical Data

- MCU MC9S12C128 with LQFP48 package (SMD)
- HCS12 16-bit CPU, uses same programming model and command set as the HC12
- 16 MHz crystal clock, up to 25 MHz bus clock using PLL
- 128 KB Flash
- 4 KB RAM
- 256 KBit serial EEPROM
- 1x SCI - asynch. serial interface (incl. RS232 drivers)
- 1x SPI - synch. serial interface
- 1x msCAN module (CAN 2.0A/B-compatible)
- High-Speed CAN bus driver (optional for 5V or 3.3V)
- 8x 16-bit Timer (Input Capture/Output Compare)
- 5x PWM (Pulse Width Modulator)
- 8-channel 10-bit A/D-Converter
- Integrated LVI-circuit (Reset Controller)
- BDM - Background Debug Mode Interface, 6-pin connector
- Indicator LED
- up to 26 general-purpose I/Os available (depends on usage of other on-board functions)
- Option: Real Time Clock providing time of day, calendar, alarm function and automatic switch-over to ext. backup battery
- Operating voltage either 3.3V or 5V (depending on installed CAN driver type), current consumption typ. 25mA
- 2.0" x 0.7" x 0.5" module size
- DIP40 footprint

Development Package Contents

- ChipS12 Controller Module with MC9S12C128, incl. Real Time Clock and 5V CAN bus driver
- TwinPEEKs Monitor (in the MCU's Flash memory)
- Carrier Board with DIP40 socket for the ChipS12 module, LC-display and a large number of peripheral functions
- RS232 cable (Sub-D9)
- BDM cable between ChipS12 and Carrier Board
- User manual (this document)
- Schematic Diagrams
- CD-ROM: contains assembler software, data sheets, CPU12 Reference Manual, code examples, C-compiler (evaluation version), etc.



2. Quick Start

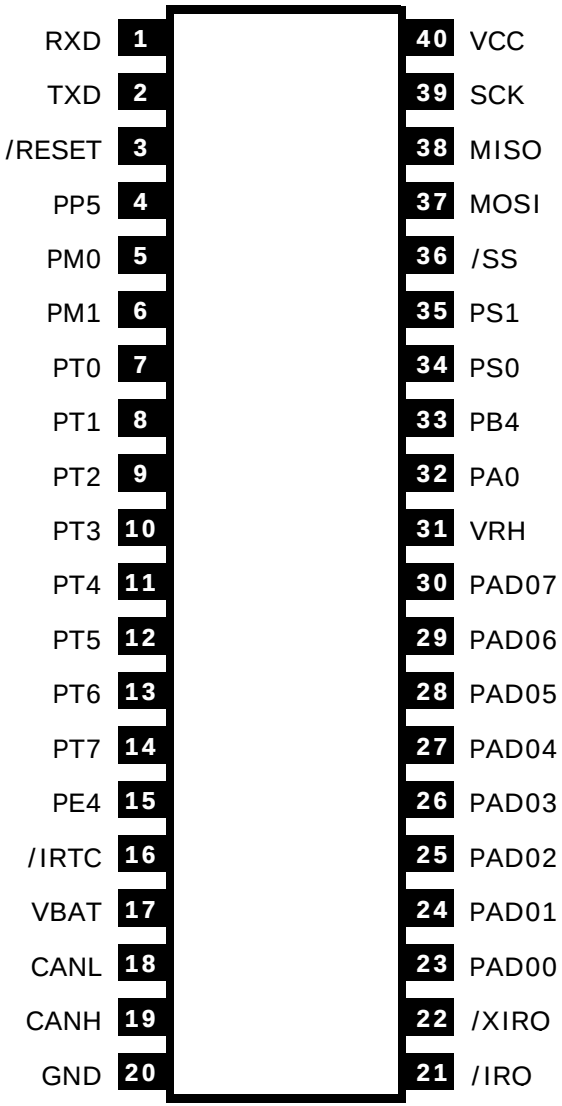
As no one likes to read lengthy manuals, we will summarize the most important things in the following section. If you need any additional information, please refer to the more detailed sections of this manual.

Here is how you can start with the development package:

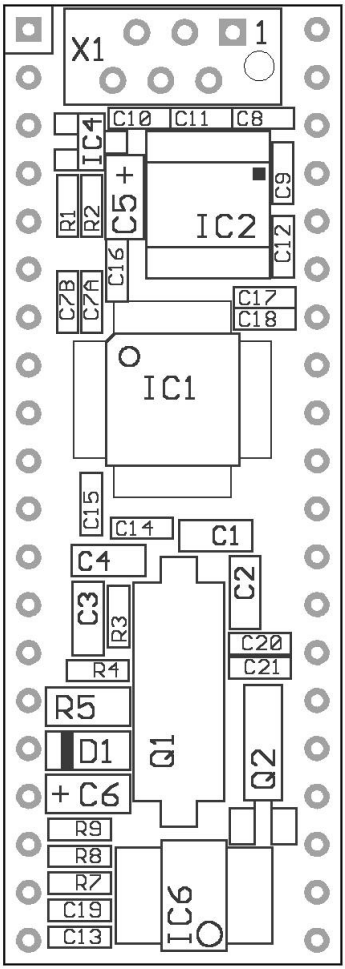
- Please check the board for any damages due to transportation
- Check if the ChipS12 module is mounted correctly on the carrier board (red BDM connectors adjacent to each other)
- Connect the device via RS232 (connector K1 on the carrier board) to your PC. Use the serial cable (Sub-D9, 1:1) which comes in the box.
- On the PC, start a Terminal Program. An easy to use Terminal Program is OC-Console, which is available at no charge from our Website!
- Select a baudrate of 19200 Bd. Disable all hardware or software protocols.
- Connect a power supply to K4, delivering approx. 9V (8..12V, polarity does not matter)
- **Please note**: wall plug power supplies are usually not stabilized and they provide a voltage that is higher than the nominal (full load) voltage. Therefore, in order to get "real" 9V, a "nominal" settinmg of 6V or 7.5V is usually sufficient. The higher the input voltage, the more heat will be produced by VR1.
- Once powered up, LD1 on the carrier board and D1 on the ChipS12 module will turn on and the Monitor program will start, displaying a message and awaiting your commands.

We hope you will enjoy working with ChipS12!

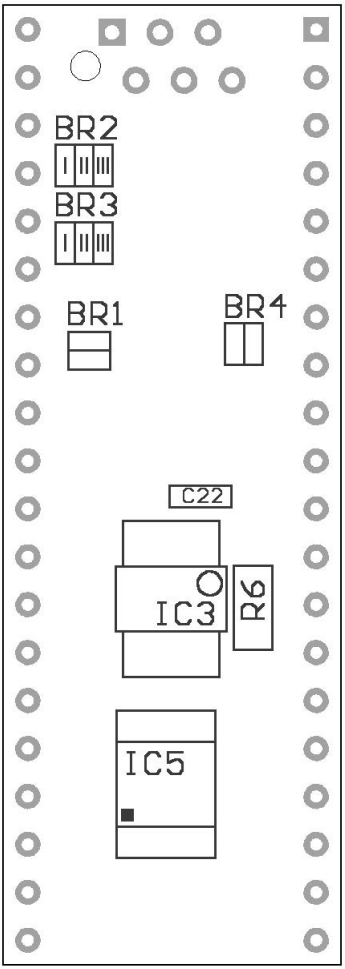
3. Module Pinout



4. Components Location Diagram



Top View



Bottom View

5. Jumpers and Solder Bridges

Jumpers

There are no jumpers on the ChipS12 module.

Solder Bridges

The following solder bridges are located on the bottom side of the PCB (see components location diagram on previous page):

BR1: VRH

open	external supply of VRH required
closed*	VRH connected to VDDA (VCC) on-board

BR2: R1OUT

1-2*	enable RS232 receiver output R1OUT (drives PS0 of MCU)
2-3	disable (tristate) RS232 receiver output R1OUT (Port pin PS0 freely available)

BR3: SHDN

1-2*	enable RS232 transceiver IC2 permanently
2-3	PE4 of MCU controls suspend mode of IC2

BR4: RRTC

open*	disable reset by RTC, /VDCC output of RTC (IC6) not in use
closed	/VDCC output of RTC connected to /RESET; provides additional LVI-function: RTC causes reset if battery-switchover is activated (see RTC data sheet)

* = factory default

6. Mechanical Dimensions

The ChipS12 module fits on a standard DIP40-socket. The pin spacing is 0.1" (2.54 mm) and the distance between pin rows is 0.6" (15.24 mm).

The outline dimensions of the module are 2.0" (50.8 mm) x 0.7" (17.78 mm).

7. Circuit Description

In this section, a number of details will be presented on how to work with the HCS12 in general and the ChipS12 Controller Module in particular.

Please be aware that, even if this manual can provide some specific hints, it is impossible to cover all kinds of knowledge and techniques required to design a microcontroller application. Please refer to the data sheets of the silicon vendors and to the manuals of your software tools to get additional information.

The software demos included in this paragraph are for demonstration puposes only. Please note, that we cannot guarantee for the correctness and fitness for a particular purpose.

Schematic Diagram

To ensure best visibility of all details, the schematic diagram of the ChipS12 is provided as a separate document.

Controller Core, Power Supply

The MCU (IC1) has three supply pin pairs: VDDR/VSSR, VDDX/VSSX and VDDA/VSSA. The nominal operating voltage (designated as VCC in the schematic diagram) of the MC9S12C128 is in the range of 3V to 5V. Internally, the MCU uses a core voltage of only 2.5V. The necessary voltage regulator is already included in the chip, as well as I/O-buffers for all general-purpose input/output pins. Therefore, the MCU behaves like a 5V or 3.3V device from an external point of view. There is just one exception: the signals for oscillator and PLL are based on the core voltage and must not be driven by an external voltage.

The three terminal pairs mentioned above must be decoupled carefully. A ceramic capacitor of 100nF is connected directly at each pair (C15, C16, C17), plus an additional 10µF tantalum capacitor (C5). Special care must be taken with VDDA, since this is the reference point for the internal voltage regulator.

The internal core voltage appears at the pin pairs VDD1/VSS1 and VDDPLL/VSSPLL in order to allow adding decoupling capacitors here as well (C7A, C7B, C14). A static current draw from these terminals is not allowed. This is especially true for VDDPLL, which serves as the reference point for the external PLL loop filter combination (R3, C3, C4).

There are two MCU pins (VRH/VRL) to define the upper and lower voltage limits for the internal analog to digital (ATD) converter. While VRL is grounded, VRH is connected to VDDA via solder bridge BR1. C18 is used for decoupling. VRH can be supplied externally after opening solder bridge BR1. This can be useful if the main supply is not in the desired tolerance band or if the ATD should work with a reference value lower than VCC. VRH must not exceed VDDA, regardless of the selected supply mode.

The TEST pin is used for factory testing only, in an application circuit this pin always has to be grounded.

Reset Generation

/RESET is the MCU's active low bidirectional reset pin. As an input it initializes the MCU asynchronously to a known start-up state. As an open-drain output it indicates that a system reset (internal to MCU) has been triggered. The HCS12 MCUs already contain on-chip reset generation circuitry including power-on reset, COP watchdog timer and clock monitor. Additionally, the MC9S12C128 is equipped with a Low Voltage Inhibit (LVI) circuit. The task of this LVI circuit is to issue a stable reset condition if the power supply falls below the level required for proper MCU operation.

To furthermore increase system reliability, IC4 can be added as an external LVI circuit. IC4 has an open-drain output in order to prevent collisions with the MCU's bidirectional reset pin. The /RESET signal is high in inactive state because IC4 contains an integrated pull-up resistor (approx. 5kOhm). Therefore, R1 is not needed if the optional IC4 is equipped.

The reset pulse issued by IC4 has a typical duration of 250ms (minimum is 140ms). It is important to note, that this pulse will only be applied during a power cycle event. IC4 will not stretch pulses coming from the MCU's internal reset sources. This is essentially important, since otherwise the MCU would not be able to detect the source of a reset. This would finally lead to a wrong reset vector fetch and could result in a system software crash. Please be aware, that also a capacitor on the reset line would cause the same fatal effect, therefore external circuitry connected to the /RESET pin of a HC12/HCS12 MCU should never include a large capacitance!

Clock Generation and PLL

The on-chip oscillator of the MC9S12Cxx can generate the primary clock (OSCCLK) using a quartz crystal (Q1) connected between the EXTAL and XTAL pins. The allowed frequency range is 0.5 to 16MHz. As usual, two load capacitors are part of the oscillator circuit (C1, C2). However, this circuit is modified compared to the standard Pierce oscillator that was widely used for the HC11 and HC12.

On the ChipS12, the MC9S12Cxx uses a Colpitts oscillator with translated ground scheme. The main advantage is a very low current consumption, though the component selection is more critical. The ChipS12 circuit uses a high-quality quartz crystal together with two load capacitors of only a few picofarad. Furthermore, special care was taken for the PCB design to introduce as little stray capacitance as possible in respect to XTAL and EXTAL.

With an OSCCLK of 16 MHz, the internal bus speed (ECLK) becomes 8 MHz by default. To realize higher bus clock rates, the PLL has to be engaged. The MC9S12Cxx can be operated with a bus speed of up to 25MHz, though most designs use 24MHz because this value is a better basis to generate a wide range of SCI baud rates.

A passive external loop filter must be placed on the XFC pin. The filter (R3, C3, C4) is a second-order, low-pass filter to eliminate the VCO input ripple. The value of the external filter network and the reference frequency determines the speed of the corrections and the

stability of the PLL. If PLL usage is not required, the XFC pin should be pulled-up to VDDPLL level.

The choice of filter component values is always a compromise over lock time and stability of the loop. 5 to 10kHz loop bandwidth and a damping factor of 0.9 are a good starting point for the calculations. With a quartz frequency of 16MHz and a desired bus clock of 24MHz, a possible choice is R3 = 4.7k and C3 = 22nF. C4 should be approximately (1/20..1/10) x C3, e.g. 2.2nF in our case. These values are suitable for a reference frequency of 1MHz (Note: to be defined in example file S12_CRG.H). The according reference divider register value is REFDV=15 and the synthesizer register setting becomes SYNR=23. Please refer to the chapter "XFC Component Selection" in the MC9S12DP256B Device User Guide for detailed description of how to calculate values for other system configurations.

The following source listing shows the steps required to initialize the 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 is used to pull /XCLKS high during reset which will select Colpitts configuration of the oscillator. If /XCLKS were low during reset, the oscillator would assume Pierce mode, which would require an alternate circuitry. However, this mode could be used to apply an external clock signal to the EXTAL pin of the MC9S12Cxx.

Please note, that different derivatives of the HCS12 have different funtionality regarding the /XCLKS pin.

Operating Modes, BDM Support

Three pins of the HCS12 are used to select the MCU operating mode: MODA, MODB and BKGD (=MODC). While MODA and MODB are internally pulled low to select Single Chip Mode, BKGD is pulled high (R2) by default. As a consequence, the MCU will start in Normal Single Chip Mode, which is the most common operating mode for application code running on the HCS12.

The HCS12 operating mode used for download and debugging is called Background Debug Mode (BDM). BDM is active immediately out of reset if the mode pins MODA/MODB/BKGD are configured for Special Single Chip Mode. This is done by pulling the BKGD pin low during reset, while MODA and MODB are pulled-down as well.

Because only the BKGD level is different for the two modes, it is quite easy to change over. However, there is no need to switch the BKGD line manually via a jumper or solder bridge because this can be done by a BDM-Pod (such as ComPOD12) attached to connector X1. A BDM-Pod is required for BDM-based download and/or debugging anyway, so it can handle this task automatically, usually controlled by a PC-based debugging program.

Integrated A/D-Converter

The MC9S12C128 contains a 10-bit Analog-to-Digital Converter module. This module (ATD) provides eight multiplexed input channels.

VRH is the upper reference voltage for all A/D-channels. On the ChipS12, VRH is connected to VDDA (VCC) through solder bridge BR1. After opening BR1, it is possible to use an external reference voltage, connected to X0/31.

The following example program shows the initialization sequence for the A/D-converter module ATD and a single-channel conversion routine. The source file S12_ATD.C also contains some additional functions for the integrated ATD module.

```
//=============================================================================
// 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;
    }

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

Indicator LED

Port pin PE7 drives a single indicator LED (D1). To control this LED, some simple macros can be used, as shown in the following C header file:

```
//===========================================================================
// 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 Interface

The MC9S12Cxx contains an asynchronous serial interface (SCI0) including one receive line and one transmit line (RXD0, TXD0). Handshake lines are not provided by the SCI module; they can be added by using general purpose I/O port lines if required.

On the ChipS12, the SCI signal lines are connected to an RS232 transceiver circuit (IC2). If the RS232 interface is not needed in an application, the output R1OUT of IC2 can be tri-stated by connecting contacts 2-3 of solder bridge BR2. As a consequence, the MCU signals PS0 and PS1 can be used as additional general-purpose I/Os.

To reduce current consumption, IC2 can be brought into suspend mode by setting the solder bridge BR3 to position 2-3. Now, MCU signal PE4 can be used to control the /SHDN input of the RS232 transceiver chip. Low level activates power-saving suspend mode.

Please note: PE4 can be configured as clock output (ECLK) by software. Avoid doing so while PE4 is used for suspend control!

The following code example shows how to use SCI0 in polling mode:

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

The MC9S12C128 contains one SPI module (SPI0), which can be used for synchronous serial communication with external SPI chips.

SPI0 consists of four individual signals: MISO, MOSI, SCK and /SS (MCU port pins PM2 .. PM5). These signals are not used on-board the ChipS12. They can be accessed at connector X0.

The following listing demonstrates some basic functions (initialization, 8-bit data transfer) for the SPI-Port SPI0 (chip select signal handling not included):

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

The MC9S12C128 does not contain a IIC hardware module. To control the on-board peripherals IC5 (RTC) and IC6 (serial EEPROM), a simplified software implementation of the IIC bus protocol can be used (for an example, please refer to file S12_SIIC.S)

The MCU signal PA0 is used as bidirectional data line (SDA), while PB4 provides the clock (SCL). Both signals can also be used to access external IIC slaves.

Serial EEPROM

On the ChipS12, extra non-volatile storage space can be provided by IC6. This serial EEPROM device has a capacity of 16 Kbit. Optionally, larger devices can be used (up to 256Kbit).

IC6 communicates over an IIC interface. The file CHIPS12_SEEP.C shows how to control the device using the software IIC module described above:

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

The ChipS12 can be optionally equipped with a R2051 Real Time Clock (RTC) from Ricoh. This chip has an IIC interface and provides time reference and calendar information.

Interrupts can be generated by the R2051 in different ways. The periodic interrupt system is configured to generate interrupt signals with a user-selectable rate. Furthermore, two alarm interrupts can be generated at preset times. The /INTR pin of the RTC is brought out to X0/16 as signal /IRTC. It can be connected externally to one of the MCU's interrupt inputs (/IRQ, /XIRQ or some general-purpose I/O-pin).

A backup battery can be connected to the module's VBAT pin (X0/17) in order to provide a backup supply in case the main power (VCC) fails. For this purpose, the use of a 3V LiMn primary battery is recommended. The switchover to backup power is accomplished when VCC falls below 2.4V. In this state, also the /VDCC output of the RTC is driven low. By closing BR4, this signal can be used as an additional system reset source.

Example routines showing how to drive the RTC of ChipS12 are contained in the file CHIPS12_RTC.C.

CAN Interface

The MC9S12C128 contains one CAN-Module, designated as CAN0. It utilizes port pins PM0 and PM1. IC3 (option) serves as physical CAN-bus interface. The CAN-bus signals CANH and CANL are available at connector X0.

If the ChipS12 is the last node in a CAN bus chain, an external termination resistor is required. Use a resistor of 120 Ohm between CANH and CANL.

R6 determines the slope control setting. The standard value (10k) must be modified for high-speed communication according to the datasheet of the manufacturer.

The device type used for IC3 must match the supply voltage of the module. Please check the part numbers mentioned in the schematic diagram.

CAN communication software can be quite complex. There are many and diverse ways to establish some CAN protocol, particularly when looking at the higher protocol layers.

However, to establish a simple connection between two CAN bus nodes can easily be done, as the following example may show:

```
//============================================================================
// 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. Application Hints

Behaviour after Reset

As soon as the reset input of the microcontroller is released, the MCU reads the Interrupt Vector at memory address $FFFE/F and then jumps to the address found there.

In the default delivery condition of the ChipS12, the Flash module of the MCU contains the TwinPEEKs Monitor Program. The reset vector points to the start of this Monitor Software. As a result, the monitor will start immediately after reset.

Startup Code

Every Microcontroller firmware starts with a number of hardware initialization commands. For the ChipS12, only setting up the stack pointer is crucial. While it was important for HC12 derivatives to disable the Watchdog, the COP Watchdog of HCS12 devices is already disabled out of reset.

Additional Information on the Web

Additional information about the ChipS12 Controller Module will be published on our Website, as it becomes available:

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

9. TwinPEEKs Monitor

Software Version 2.3

Serial Communication

TwinPEEKs communicates over the RS232 interface using a line speed of **19200 Baud**. Settings are: 8N1, no hardware or software handshake, no protocol.

Autostart Function

After reset, the TwinPEEKs monitor checks, whether port pins PT2 and PT3 (X0/9+10) are connected. If this is the case, the monitor immediately jumps to address $8000.

This feature allows to start an application program automatically without modifying the reset vector, which is located in the protected Flash Boot Block.

Write Access to Flash EEPROM

The CPU can read every single byte of the microcontroller's resources - the type of memory does not matter. However, for write accesses, two rules are important: Flash EEPROM has to be erased before any write attempt. Programming is done by writing words (two bytes at a time) to aligned addresses.

To form such aligned words, two subsequent bytes have to be combined. TwinPEEKs is aware of this, but the following problem can not be avoided by the monitor:

The monitor is processing each S-Record line seperately. If the last address of such an S-Record is even, the 2nd byte to form a complete word is missing. TwinPEEKs will append an $FF byte in this case, so it is able to perform the word write.

The problem occurs, if the byte stream continues with the following S-Record line. The byte, that was missing in the first attempt,

would require a second write access to the same (word) address - which is not allowed. As a consequence, a write error ("not erased") will be issued.

To avoid this problem, it is necessary to align all S-Record data before programming. This can be done using the freely available Motorola Tool SRECCVT:

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

A detailed description of this tool is contained in the SRECCVT Reference Guide (PDF).

Redirected Interrupt Vectors

The interrupt vectors of the HCS12 are located at the end of the 64KB memory address range, which falls within the protected monitor code space. Therefore, the application program can not modify the interrupt vectors directly. To provide an alternative way, the monitor redirects all vectors (except the reset vector) to RAM. The procedure is similar to how the HC11 behaved in Special Bootstrap Mode.

The application program can set the required interrupt vectors during runtime (before global interrupt enable!) by placing a jump instruction into the RAM pseudo vector. The following example shows the steps to utilizy the IRQ interrupt:

```
ldaa #$06           ; JMP opcode to
staa $0FEE          ; IRQ pseudo vector
ldd  #isrFunc       ; ISR address to
std  $0FEF          ; IRQ pseudo vector + 1
```

For a C program, the following sequence could be used:

```
// install IRQ pseudo vector in RAM
// (if running with TwinPEEKs monitor)
*((unsigned char *)0x0fee) = 0x06;    // JMP opcode
*((void (**)(void))0x0fef) = isrFunc;
```

The following assembly listing is part of the monitor program. It shows the original vector addresses (1st column from the left) as well as the redirected addresses in RAM (2nd column).

Please note: the actual vector usage depends on the particular HCS12 derivative (see 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
```

Usage

A TwinPEEKs command is comprised by a single character, followed by a number of arguments (as required). All numbers are hexadecimal numbers without prefix or suffix. Both, upper and lower case letters are allowed.

The CPU's visible address range is 64KB, therefore address arguments are not longer than 4 digits. An end address always refers to the following (not included) address. For example, the command "D 1000 1200" will display the address range from $1000 to (including) $11FF.

User input is handled by a line buffer. Valid ASCII codes are in the range from $20 to $7E. Backspace ($08) will delete the character left of the cursor. The <ENTER> key ($0A) is used to conclude the input.

The monitor prompt always displays the current program page (i.e., the contents of the PPAGE register).

Monitor Commands

Blank Check

`Syntax: B`

Blank check whole Flash Memory (ex. monitor code space). If Flash memory is not blank, then display number of first page containing a byte not equal to $FF.

Dump Memory

`Syntax: D [adr1 [adr2]]`

Display memory contents from address adr1 until address adr2. If end address adr2 is not given, display the following $40 bytes. Memory location adr1 will be highlighted in the listing.

Edit Memory

Syntax: E [addr {byte}]

Edit memory contents. In the command line, the start address addr can be followed by up to four data bytes {byte}, thus allowing byte, word and doubleword writes. The write access will be performed immediately and then the function will return to the input prompt.

If the command line did not contain any data {byte}, the interactive mode will be started. The monitor is able to identify memory areas which can only be changed on a word-by-word basis (Flash EEPROM). In such cases, the monitor always awaits and uses 16-bit data.

To exit the interactive mode, simply type "Q" . Additional commands are:

```
<ENTER>  next address
   -     previous address
   =     same address
   .     exit (like Q)
```

Fill Memory

Syntax: F adr1 adr2 byte

Fill memory area starting at address adr1 and ending before adr2 with the value byte.

Goto Address

Syntax: G [addr]

Call the application program at address addr. Note: there is no regular way for the application program to return to the monitor.

Help

Syntax: H

Display a brief command overview.

System Info

Syntax: I

Display system information. This includes address range of register block, RAM, EEPROM and Flash, and the MCU identifier (PARTID).

Load

Syntax: L

Load an S-Record file into memory. Data records of type S1 (16-bit MCU addresses) and S2 (linear 24-bit addresses) can be processed. S0-Records (comment lines) will be skipped. S8- and S9-Records are recognized as end-of-file mark.

S2-Records use linear adresses according to Motorola guidelines. The valid address range for the MC9S12C128 starts at 0x0E0000 (0x38 * 16KB) and ends at 0xFFFFF (0x40 * 16 KB - 1).

Before loading into non-volatile memory (Flash EEPROM), this kind of memory must always be erased. Also, only word writes can be used in this case. It may be required to prepare S-Record data accordingly, before it can be downloaded (see instructions above).

The sending terminal program (such as OC-Console) must wait for the acknowledge byte (*), before starting the transmission of another line. This way, the transmission speed of both sides (PC and MCU) are synchronized.

Move Memory

Syntax: M adr1 adr2 adr3

Copy a memory block starting at address adr1 and ending at adr2 (not included) to the area starting at address adr3.

Select PPAGE

Syntax: P [page]

Select a program page (PPAGE). This page will become visible in the 16KB page window from $8000 to $BFFF.

Erase Flash

Syntax: X [page]

Erase one page (16KB) of Flash memory.

If page is not specified, the whole Flash memory (ex. monitor code space) will be erased after user confirmation. To remove (erase) the monitor code, a BDM tool such as ComPOD12/StarProg is required.

10. Memory Map

The memory map of the MC9S12C128 is initialized by the TwinPEEKs monitor as follows:

ChipS12.C32*

Start	Ende	Belegung
$0000	$03FF	Control Registers
$0800	$0FFF	2KB RAM TwinPEEKs uses the top 512 bytes
$4000	$7FFF	16KB Flash (equals Page $3E)
$8000	$BFFF	16KB Flash Page $3E (Page **$3E** or $3F selectable using PPAGE)
$C000	$FFFF	16KB Flash (equals Page $3F) TwinPEEKs uses the top 4KB

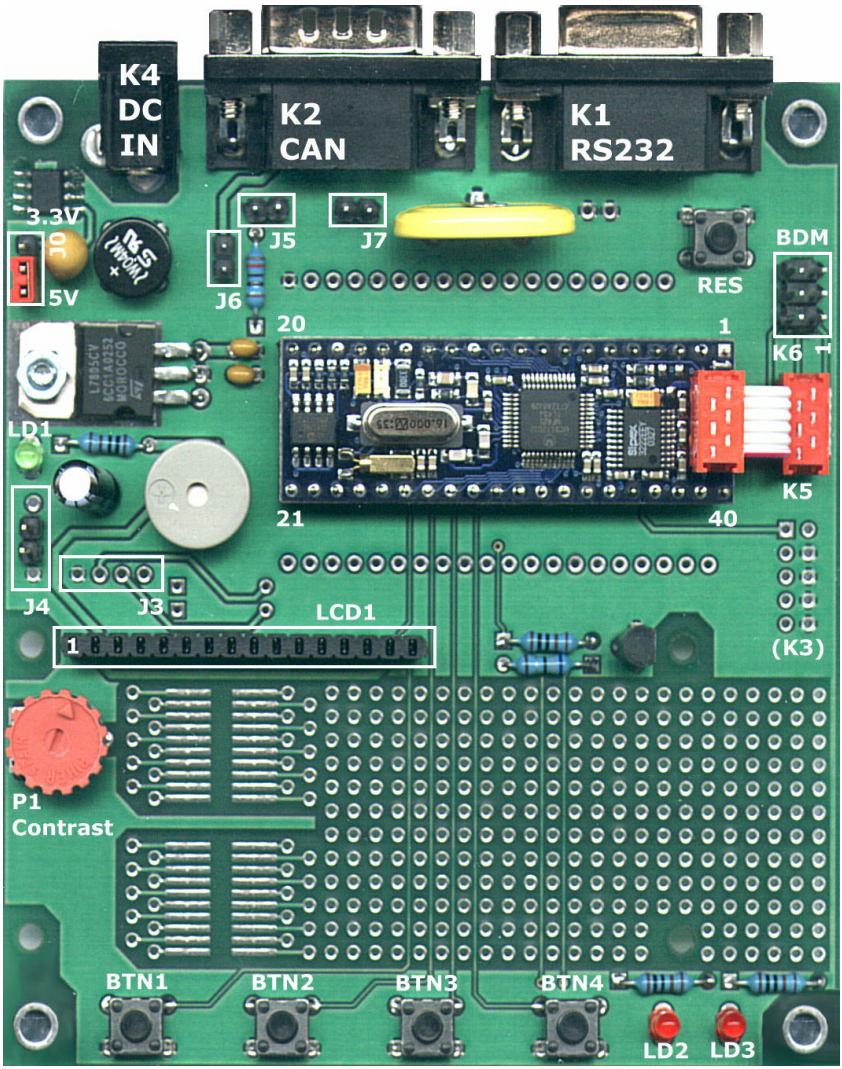
ChipS12.C128

Start	Ende	Belegung
$0000	$03FF	Control Registers
$0400	$0FFF	4KB RAM, with 3KB of it visible (the lower 1024 bytes are hidden by Control Registers) TwinPEEKs uses the top 512 Bytes
$1000	$3FFF	16KB Flash (equals Page $3D), with 12KB of it visible (the lower 4KB are hidden by RAM and Control Registers)
$4000	$7FFF	16KB Flash (equals Page $3E)
$8000	$BFFF	16KB Flash Page $38 (Page **$38**..$3F selectable using PPAGE)
$C000	$FFFF	16KB Flash (equals Page $3F) TwinPEEKs uses the top 4KB

* old board version (no longer available)

11. Carrier Board

Parts Location Plan



Jumpers and Connectors

J0	1-2	ChipS12 supply voltage VCC=3.3V (I_{CC}<0.1A!)
	2-3*	ChipS12 supply voltage VCC=5V
J1	not equip'd	Connection (solder pads) to ChipS12 pin 21..40
J2	not equip'd	Connection (solder pads) to ChipS12 pin 3..20
J3	not equip'd	optional; same pin-out as J4; may be used as I2C-bus connection (requires software driver)
J4	open*	1=VCC, 2=PT2, 3=PT3, 4=GND if pin 2 and pin 3 are connected during reset, monitor autostart becomes active
J5	open*	close to activate CAN-bus termination by R5 (required at both bus end points)
J6	open*	close to connect supply voltage VIN to K2/9, can be used to share power supply with another CAN-bus node
J7	open*	close to connect backup battery BAT1 to VBAT input of ChipS12 (RTC backup supply)
K1		RS232 connector (use Sub-D9 1:1 cable to PC)
K2		CAN connector Sub-D9
K3	not equ.	optional; may be used as SPI-port
K4		Connector for (wall) power supply, polarity does not matter, input ca. 8..15Volt DC
K5		BDM connection to ChipS12 module
K6		BDM connector for debugger (BDM-pod)
LCD1		Connector for alphanum. LC-display

* = factory default

Schematic Diagram

To ensure best visibility of all details, the schematic diagram of the ChipS12 Carrier Board is provided as a separate document.

Notes on Power Supply

On the carrier board, the voltage regulator VR2 can deliver up to 100mA (see data sheet of the LE33). This value is more than sufficient for the ChipS12 Module alone. However, if additional external componets are connected, the limit could easily be exceeded. In this case the total current consumption should be monitored (e.g. on jumper J8, pins 1 and 2)!

The LC-display LCD1 is always 5V-powered, even if the controller module is operated with 3.3V. The R/W-input of the LCD is permanently connected to L-level, so display control is done by write accesses only. The 3.3V-CMOS-outputs of the microcontroller deliver sufficient voltage levels to drive the 5V-TTL-inputs of the LC-display.