# S12compact

Hardware Version 1.10

## User Manual

July 4 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

The S12compact is a versatile controller module based on the HCS12 16-bit microcontroller family from Motorola. It combines compact size (Half-Euro form factor, 80mm x 100mm) with powerful computing features and a rich set of peripherals. The controller module can be used for a wide range of applications, e.g. industrial controllers, measuring devices and data loggers.

The S12compact is equipped with a MC9S12DP512 microcontroller unit (MCU). It contains a 16-bit HCS12 CPU, 512KB of Flash memory, 14KB RAM, 4KB EEPROM and a large amount of peripheral function blocks, such as SCI, SPI, CAN, IIC, Timer, PWM, ADC and General-Purpuse-I/Os. The MC9S12DP512 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.

In addition to the on-chip controller functions, the S12compact module provides a number of useful peripheral options: 16-bit A/D-Converter and 16-bit D/A-Converter including precision voltage reference, battery-backed Real Time Clock (RTC), USB-Interface and 2MB (16Mbit) extra memory using Serial Data Flash.

For the HCS12 microcontrollers, a wide range of software tools (Monitors, C-Compilers, BDM-Debuggers) is available to accelerate the development process.

## Technical Data

- MCU MC9S12DP512 with LQFP112 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
- 512KB Flash
- 4KB EEPROM
- 14KB RAM

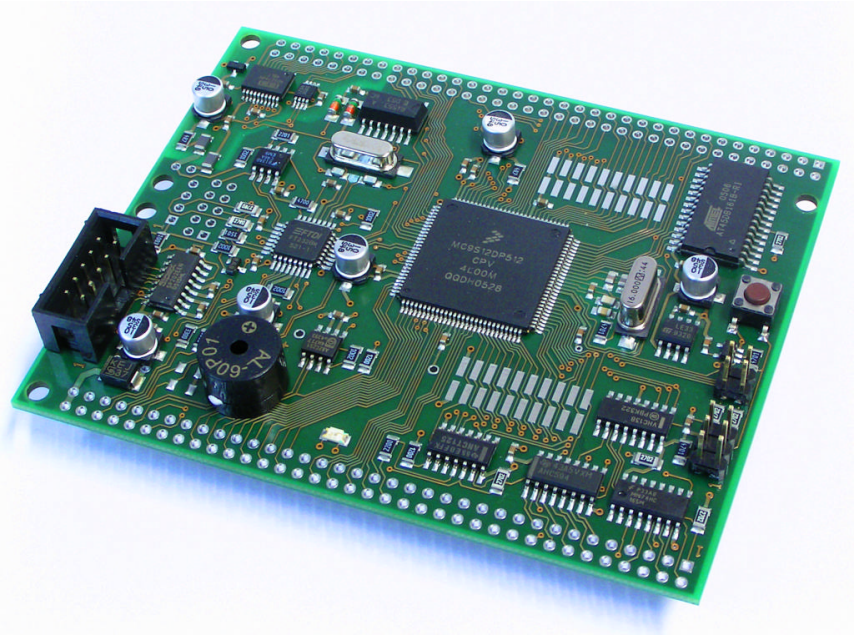
- 2x SCI - asynch. serial Interface (e.g. RS232, LIN)
- 3x SPI - synch. serial Interface
- 1x IIC - Inter-IC-Bus
- 5x msCAN-Module (CAN 2.0A/B-compatible)
- 8x 16-Bit Timer (Input Capture/Output Compare)
- 8x PWM (Pulse Width Modulator)
- 16-channel 10-bit A/D-Converter
- BDM-Interface for Download and Debugging
- Special LVI-Circuit (Reset Controller)
- Serial Interface with RS232-Transceiver (for PC connection)
- Second serial port for IF-Modules (RS232, RS485, LIN...)
- Indicator-LED
- Sound Transducer (Buzzer)
- High-Speed phys. CAN-Interface
- Reset Button
- up to 70 free General-Purpose-I/Os
- eight additional digital inputs
- eight additional digital outputs
- Operating voltage 5V, current consumption 70 mA typ.
- Mech. Dimensions: 80mm x 100mm

## Optional Components

- Option RTC: battery-backed Real Time Clock
- Option ADC: 8-channel 16-bit A/D-Converter (4096mV)
- Option DAC: 2-channel 16-bit D/A-Converter (4096mV)
- Option USB: Full-speed USB2.0 Interface (uses second SCI)
- Option SDF: 2MB (16Mbit) Serial Data Flash

## Package Contents

- Controller Module (with options as ordered)
- TwinPEEKs Monitor program (in the MCU's Flash Memory)
- RS232 Cable (Sub-D9)
- USB Cable (only for option /USB)
- set of header connectors (two 72-pin male headers)
- 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

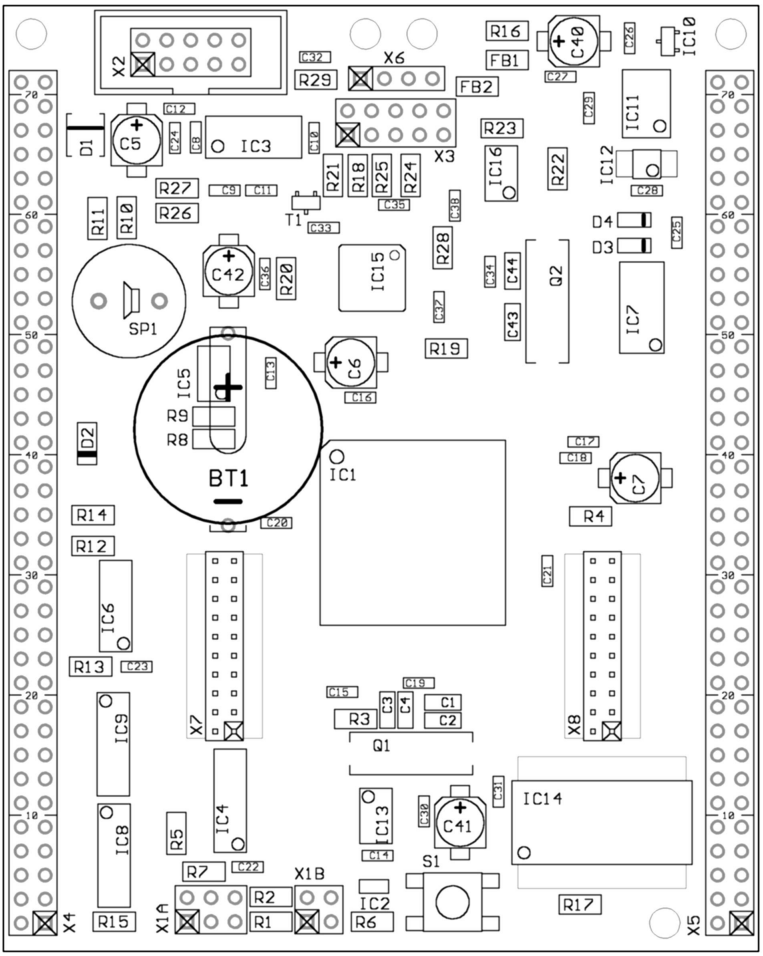
As no one likes to read lengthy manuals, we will summarize the most important things in the following section. If you need any additional information, please refer to the more detailed sections of this manual.

Here is how you can start:

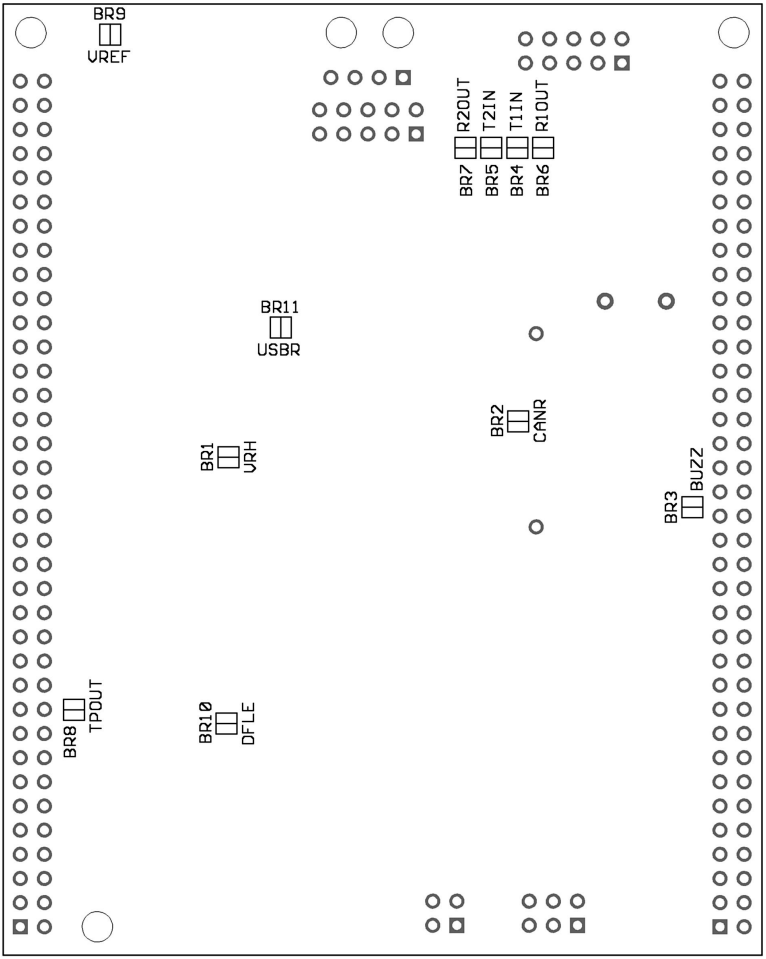
- Please check the board for any damages due to transportation
- Connect the Controller Module via RS232 to a PC. The connection between S12compact (interface SCI0, connector X2) and PC is simply made using the flat ribbon cable which is in the box.
- On the PC, start a Terminal Program. An easy to use 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 (stabilized!) power supply, e.g. here:
- Ground on X4 pin 70
- +5V on X4 pin 72
- Check voltage and polarity **before** making the connection!
- Once powered up, the Monitor program will start, display a message and await your commands.

We hope you will enjoy working with the S12compact Controller Module!

# 3. Parts Location Diagram




*Place Plan - Component Side*




*Solder Bridges on the solder side of the PCB*

# 4. Jumpers and Solder Bridges

## Jumpers

There are no jumpers on the S12compact.

## Solder Bridges

On the solder side of the module, the following solder bridges can be found:

**BR1: VRH**

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

**BR2: CANR**

| | |
|---|---|
| open | no CAN-Bus termination on-board |
| closed* | on-board CAN-Bus termination active |

**BR3: BUZZ**

| | |
|---|---|
| open | Port pin PT2 freely available |
| closed* | PT2 controls buzzer |

**BR4: T1IN**

| | |
|---|---|
| open | Port pin TXD0 (PS1) freely available |
| closed* | TXD0 connected to RS232 Transceiver IC3 |

**BR5: T2IN**

| | |
|---|---|
| open | Port pin PM3 freely available |
| closed* | PM3 connected to RS232 Transceiver IC3 |

**BR6: R1OUT**

| | |
|---|---|
| open | Port pin RXD0 (PS0) freely available |
| closed* | RXD0 connected to RS232 Transceiver IC3 |

* = Factory Default Setting

### BR7: R2OUT

| | |
|---|---|
| open | Port pin PM2 freely available |
| closed* | PM2 connected to RS232 Transceiver IC3 |

### BR8: TPOUT

| | |
|---|---|
| open* | TPOUT and /IRQ not connected |
| closed | TPOUT connected to /IRQ, Real Time Clock IC7 can issue an interrupt |

### BR9: VREF

| | |
|---|---|
| open | external supply of VREF required |
| closed* | IC10 delivers reference voltage VREF for the 16-bit A/D-Converter IC11 and the 16-Bit D/A-Converter IC12 |

### BR10: DFLE

| | |
|---|---|
| open | Port pin MISO1 (PH0) freely available |
| closed* | MISO1 connected to buffer IC6B (Serial Data Flash) |

### BR11: USBR

| | |
|---|---|
| open* | Reset input of USB Transceiver IC15 and system reset signal /RESET not connected |
| closed | system /RESET also resets USB Transceiver |

* = Factory Default Setting

# 5. Mechanical Dimensions

The following table summarizes the mechanical dimensions of the S12compact. The values provide a basis for the design of carrier boards etc. Please note: Always check all mechanical dimensions using the real hardware module!

The reference point (0,0) is located at the "south/west" corner of the PCB. The PCB is orientated vertically, as shown in the Parts Location Diagram (see above).

All data for holes/drills (B) refer to the center of the hole/drill, connectors (X) are referenced by pin 1.

| | X<br>in mm | Y<br>in mm |
|---|---|---|
| X1A | 19,5 | 3,2 |
| X1B | 32,2 | 3,2 |
| X2 | 14,8 | 93,8 |
| X3 | 36,4 | 86,4 |
| X4 | 4,5 | 3,0 |
| X5 | 78,1 | 3,0 |
| X6 | 37,8 | 92,3 |
| X7 | 24,4 | 23,3 |
| X8 | 64,4 | 23,3 |
| B1 | 3,0 | 97,0 |
| B2 | 77,0 | 97,0 |
| B3 | 70,0 | 3,0 |
| PCB | 80,0 | 100,0 |

# 6. Circuit Description

## Schematic Diagram

To ensure best visibility of all details, the schematic diagram of the S12compact controller module is provided as a separate document.

## Controller Core, Power Supply

The nominal operating voltage of the MC9S12DP512 is 5V. This MCU (IC1) has three supply pin pairs: VDDR/VSSR, VDDX/VSSX and VDDA/VSSA. Internally, the MCU uses a core voltage of only 2.5V. The necessary voltage regulator is already included in the chip, as well as 5V I/O-buffers for all general-purpose input/output pins. Therefore, the MCU behaves like a 5V device from an external point of view. There is just one exception: the signals for oscillator and PLL are based on the core voltage und must not be driven by 5V levels. High level on the pin VREGEN is needed to enable the internal voltage regulator.

The three terminal pairs mentioned above must be decoupled carefully. A ceramic capacitor of at least 100nF should be connected directly at each pair (C15, C16, C17). It is recommended to add a 10µF (electrolytic or tantalum) capacitor per node, especially if some MCU port pins are loaded heavily (C5, C6, C7). Special care must be taken with VDDA, since this is the reference point (VDDA/2) for the internal voltage regulator.

The internal core voltage appears at the pin pairs VDD1/VSS1, VDD2/VSS2 and VDDPLL/VSSPLL, which have to be decoupled as well (C19, C20, C21). 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 usually tied to VDDA. C18 is used for decoupling. VRH can be supplied externally after opening the 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 5V. 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. It is, however, necessary to add an external Low Voltage Inhibit (LVI) circuit, also referred to as "reset controller". The task of this reset controller is to issue a stable reset condition if the power supply falls below the level required for proper MCU operation.

To prevent collisions with the bidirectional /RESET pin of the MCU, the LVI circuit IC2 has an open-drain output. In the inactive state it is pulled-up high by the resistor R6. The detector treshold of IC2 is typically 4.6V, which is slightly higher than the required minimum MCU operating voltage of 4.5V.

Furthermore, IC2 is capable of stretching the reset output to filter out short pulses on the power supply effectively. The duration of that delay can be selected using the capacitor C14. A value of 100nF results in a delay of approx. 50..80ms.

It is important to note, that this delay will only be applied during a power cycle event. IC2 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 MC9S12DP512 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 used for the HC11 or most HC12 derivatives.

The MC9S12DP512 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 S12compact circuit uses a 16MHz automotive quartz from NDK together with two load capacitors of only 3.9pF. 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 16MHz, the internal bus speed (ECLK) becomes 8MHz by default. To realize higher bus clock rates, the PLL has to be engaged. The MC9S12DP512 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 must be tied to VDDPLL.

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
    }

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

An alternative, external clock source can be used for the MC9S12DP512 if the internal oscillator and PLL are disabled by applying a low level to the /XCLKS pin during reset. Since this option is not used on the S12compact Controller Module, R5 is used to pull /XCLKS high. Please note, that other HCS12 derivatives will have different features associated with 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 pulled low (R1, R2) to select Single Chip Mode, BKGD is pulled high (R7) 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 X1A. 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.

The 6-pin header X1A uses the suggested standard BDM12 connector layout. Connector X1B carries additional MCU signals, which are normally not needed for BDM12 debugging. Some debuggers, however, provide additional features, which rely on the presence of these supplemental signals.

## Integrated A/D-Converter

The MC9S12DP512 contains two 10-bit Analog-to-Digital Converter modules. Each module (ATD0, ATD1) provides eight multiplexed input channels.

VRH is the upper reference voltage for all A/D-channels. On the S12compact, VRH is connected to VDDA (5V) through solder bridge BR1.

After opening BR1, it is possible to use an external reference voltage, which has to be applied to X5/46. For this purpose, the precision voltage reference IC10 could be used (if present). IC10 delivers 4.096V at X5/71.

The following example program shows the initialization sequence for the A/D-converter module ATD0 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;
    }

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

## Integrated EEPROM

The internal EEPROM module of the MC9S12DP512 contains 4KB of memory. It consists of 1024 sectors with 4 bytes (32 bits) per sector. For erasure, any single sector can be selected. Programming is done by words (2 bytes). Read accesses can be made to any word or byte.

After reset, the EEPROM module of the MC9S12DP512 is mapped to address 0x0000. In the lower 1KB area (0x0000..0x03FF), control registers take precedence over EEPROM. In order to use the full EEPROM space, the EEPROM module can be relocated (see INITEE control register).

In the following example, the EEPROM module is left at it's default position. The initialization sequence just takes care for setting up the EEPROM Clock Divider according to the quartz crystal frequency. The write function wrSectEETS() copies two words (4 bytes) from source address src to EEPROM address dest. dest must be identical to an EEPROM sector border (aligned 32 bit value). If the sector is not erased (erased state = 0xFFFFFFFF), the routine will perform a sector erase before writing to the sector.

The access functions readItemEETS() and writeItemEETS() provide a more abstract way to deal with EEPROM contents. Instead of using certain addresses, which must be part of the EEPROM address range, these routines use abstract "item numbers", with each item consisting of a variable amount of data (1 to 4 bytes).

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

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

#include "datatypes.h"
#include <mc9s12dp512.h>
#include "s12_eets.h"

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

void initEETS(void) {

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

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

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

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

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

INT8 writeItemEETS(UINT16 item_no, void *item) {

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

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

INT8 readItemEETS(UINT16 item_no, void *item) {

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

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

## Indicator-LED

The logic level of port pin PE7 is latched at the rising edge of reset and provides the control signal /XCLKS. This signal is used to select one of two possible clock configurations for the MC9S12DP512. High level activates the integrated Colpitts oscillator. After reset, PE7 can be used as a general-purpose I/O-pin. On the S12compact, this signal is used to control the indicator LED D2, driven by buffer IC6C.

To turn the LED on or off, some simple macros can be used, as shown in the following C header file:

```
//============================================================================
// File: S12CO_LED.H - V1.00
//============================================================================

#ifndef __S12CO_LED_H
#define __S12CO_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 //__S12CO_LED_H ==================================================
```

## Buzzer

The sound transducer (buzzer) SP1 is driven by buffer IC6D and controlled by the MCU's port pin PT2, provided the solder bridge BR3 is closed.

PT2 is connected with one of the eight timer channels of the MCU. Frequency generation is realized using the Output-Compare function of the timer system.

The following example demonstrates, how Output-Compare interrupts can be used to generate oscillations in the audible range:

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

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

#include "datatypes.h"
#include <mc9s12dp512.h>
#include "s12_ect.h"
#include "s12_crg.h"                    // contains S12_ECLK value
#include "acprd_freqout.h"

//-- Static Vars -------------------------------------------------------------

UINT16 freqout_tticks;

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

void initFreqOut(void) {

    // make sure timer is enabled
    TSCR1  |= BM_TEN;
    // prescaler = 2**4 = 16
    TSCR2 = 0x04;
    // select Output Compare function for channel 2
    TIOS |= BM_2;
    DDRT |= BM_2;
    // enable Interrupt for channel 2
    TIE |= BM_2;
    // timer disconnected from PT2 pin
    TCTL2 &= ~(BM_OM2 | BM_OL2);
    }

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

// period is in µs
//
void setFreqOut(UINT16 period) {

    UINT16 tticks;

    tticks = period * (S12_ECLK / 2000000L);
    tticks /= TIMER_TCNT_PRE;

    if(period == 0) {
        // disconnect PT2 pin
        TCTL2 &= ~(BM_OM2 | BM_OL2);
        }
    else {
        // connect PT2 pin
        TCTL2 |= BM_OL2;
        }
    freqout_tticks = tticks;
    }

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

// OC2 toggles buzzer
//
#ifdef METROWERKS_C
interrupt
#endif
#ifdef IMAGECRAFT_C
#pragma interrupt_handler isrOC2
#endif
void isrOC2(void) {

    TC2 += freqout_tticks;
    TFLG1 = BM_2;                       // clear Intr flag
    }

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

## RS232 Interface

The MC9S12DP512 provides two asynchronous serial interfaces (SCI0, SCI1). Each interface has one receive line and one transmit line (RXDx, TXDx). Handshake lines are not provided by the SCI module, though they can be added by using general purpose I/O port lines if required.

IC3 is an industry standard RS232 line transceiver circuit. It is connected to SCI0, which is the MCU's first asynchronous serial communications channel. In addition to the receive and transmit lines (RXD0, TXD0), two general purpose I/Os (PM2, PM3) can be used as hardware handshake lines. If the RS232 transceiver IC3 is not needed, the dedicated MCU pins can be used freely after opening the four solder bridges BR4..BR7.

To connect the S12compact to a PC, a 10-wire flat ribbon cable can be used. The cable must have a 10-pin female header connector at the S12compact side (X2) and a female Sub-D9 connector at the PC side.

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

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

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

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

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

void initSCI0(UINT16 bauddiv) {

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

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

UINT8 getSCI0(void) {

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

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

void putSCI0(UINT8 c) {

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

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

## IF-Module Connection

SCI1 can be used in two different ways on the S12compact. If the USB option is equipped, SCI1 is used to control the USB-Transceiver IC15 (refer to section USB Interface for details). Otherwise, SCI1 can be used as an universal TTL-RS232 (without level shifter) to control an external IF-Module.

IF-Modules are serial interface modules, having a standardized connector definition. They are available for different physical interface types, such as RS232, RS485, current-loop or LIN. IF-Modules can be connected to X3 using a 10-wire flat ribbon cable.

The I/O signals PM4..7 are associated to SCI1 as handshake lines on the S12compact. If there is no IF-Module on X3 and also the USB option is not present, these signals (including RXD1 and TXD1) can be used as general-purpose I/Os. They are accessible at connector X4 and X5, respectively.

## USB Interface

IC15 (FT232BM) is an USB-UART. This chip provides a transparent conversion from RS232 to USB and back. The communication protocol is implemented according to the current USB specification 2.0. The FT232BM belongs to the USB "Full-Speed" device class.

On the S12compact, the microcontroller can send data over SCI1 to the USB-UART. Thereupon, all data is transferred to he USB-bus and can be accessed using a virtual COM-Port driver on the Host-PC. The required driver software is provided by FTDI, the manufacturer of the FT232BM (see http://www.ftdichip.com). The drivers are free of charge. Currently, there are driver versions for Windows-PCs (98 to XP), Apple computers and Linux systems. On the S12compact supplement CD, drivers (for Windows platform) are provided as well, though there may exist newer versions on the website.

To transfer data between USB-UART and microcontroller, only the signals TXD and RXD (MCU pins TXD1 and RXD1) are required. Optionally, a CTS/RTS hardware handshake protocol can be

implemented. The MCU can read the /RTS output using port pin PM6, while PM7 controls the /CTS input of the USB-UART.

Transmit- and receive-activities can be displayed using two LEDs. Their anodes must be wired to VCC (5V), while the cathodes have to be connected to /RXLED and /TXLED (X4/67+68), respectively.

/PWREN is set by IC15 according to the enumeration status of the USB device. This signal can be detected by MCU port pin PM4. The control signal /SLEEP is low if IC15 is in Suspend-Mode, PM5 is used to read this signal.

BR11 must be closed if a system reset should also reset the USB-UART, which is normally not required. The USB-UART has an independent power-on reset circuit, so, normally it does not need an external reset source. If BR11 is closed, any system reset will temorarily remove the USB device from the bus "logically". When the reset condition has ended, the device needs to be re-enumerated by the USB-Host.

The serial EEPROM IC16 can hold configuration data for the USB-UART. If IC16 is erased (default delivery state), the USB-UART uses standard descriptors to answer descriptor requests from the host. User descriptors (VID, PID, strings, serial numbers etc.) can be downloaded to EEPROM using a PC-based utility program (provided by FTDI). The programming can be done in-circuit via USB.

Please note: If the USB-Option is equipped, the IF-Module connection X3 is not available.

## SPI Subsystem

The MC9S12DP512 provides three independent SPI-Ports. On the S12compact, port SPI0 is used to control the peripheral components RTC, ADC, DAC, PARIN and PAROUT. Port SPI1 is used to communicate with the Serial Data Flash; SPI2 is not in use.

SPI0 consists of four individual signals: MISO, MOSI, SCK and /SS (MCU port pins PS4 to PS7). The slave select signal /SS is not in use on the S12compact, though it can be accessed on the side header connectors.

The SPI chip select signals are derived from MCU port pins PH[4..6]. The decoder chip IC4 activates one of it's eight low-active outputs depending on the Port H pattern. This provides an economical way to provide up to eight chip selects with a small number of MCU ressources. The table below shows the usage of SPI chip selects on the S12compact:

| Chip Select | Port H | Usage |
|---|---|---|
| /SPICS0 | x000xxxx | Real Time Clock |
| /SPICS1 | x001xxxx | D/A-Converter |
| /SPICS2 | x010xxxx | A/D-Converter |
| /SPICS3 | x011xxxx | Parallel In Shift |
| /SPICS4 | x100xxxx | Parallel In Latch |
| /SPICS5 | x101xxxx | Parallel Out |
| /SPICS6 | x110xxxx | freely available (X4/20) |
| /SPICS7 | x111xxxx | no chip select active |

The following listing demonstrates some basic functions (initialization, 8-bit data transfer) for the SPI-Port SPI0:

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

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

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

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

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

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

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

UINT8 xferSPI0(UINT8 abyte) {

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

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

## Extra Ports PARIN and PAROUT

While the MC9S12DP512 provides a large number of general-purpose I/O-pins, it may be desireable for some applications to keep a number of ports unused. This is especially the case if the external bus interface is intended to be used.

A preferred way to provide additional input or output ports consists of using one of the Serial Peripheral Interface (SPI) modules of the MC9S12DP512. It is quite easy to create eight additional binary outputs by adding a shift register like IC9. In addition to the SPI signals MOSI and SCK only a chip select signal (/SPICS5) is needed to operate IC9. This shift register has an asynchronous reset input for both the shift register and the output latch, so all output lines PO0..PO7 will drive L-level out of reset automatically.

The following listing shows an example for parallel output:

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

void putSPIPO(UINT8 abyte) {

    // set up SPI mode
    SPI0CR1 = BM_SPE | BM_MSTR;         // CPOL=0 CPHA=0
    // send data
    PTH = S12CO_SPICS5;                 // enable /CS5
    xferSPI0(abyte);
    PTH = S12CO_SPICSH;                 // disable /CS5
    }

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

Using the same SPI port, IC8 provides eight additional input pins. Two chip selects (/SPICS3 and /SPICS4) are needed to latch the input information coming from PI0..PI7 and to shift out data via MISO respectively. IC6A is needed to decouple the push/pull-output QH of IC8 from MISO, in order to allow multiple slaves on the same SPI port.

The following listing shows an example for parallel input:

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

UINT8 getSPIPI(void) {

    UINT8 abyte;

    // set up SPI mode
    SPI0CR1 = BM_SPE | BM_MSTR;         // CPOL=0 CPHA=0
    // latch input data
    PTH = S12CO_SPICS4;
    PTH = S12CO_SPICSH;
    // serialize latched data
    PTH = S12CO_SPICS3;
    abyte = xferSPI0(0);
    PTH = S12CO_SPICSH;
    return abyte;
    }

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

## Real Time Clock (RTC)

The Real Time Clock (RTC) IC7 provides an independent timing reference including calendar information. Under normal operating conditions, it is supplied by VCC (via D3). In case of a power-loss, the lithium cell BT1 delivers a backup supply in order to avoid data corruption. The installed battery type (CR2032) is designed for a life time of at least five years.

The RTC is controlled via SPI-port SPI0. This includes the signals MISO, MOSI and SCK (port pins PS4/5/6). The chip select signal /SPICS0 is provided by IC4 (see above). The read/write control is done by PH7.

The following listing shows some basic input/output routines for the RTC. They are based upon the SPI driver functions which are shown above. For more complex functions, such as set date/time, please refer to the software examples contained on the product CD.

For further instructions how to program the RTC, please refer to the RTC4553 Application Manual.

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

// Func: getRTC()
// Args: regno[0..3] is RTC register number
// Retn: [0..3] RTC data (register contents)
// Note: there is no need to range check the regno argument since the
//       RTC will ignore any additional bits anyway
//
UINT8 getRTC(UINT8 regno) {

    UINT8 result;

    // set up SPI mode: enable, master, CPOL=1, CPHA=1, LSB first
    SPI0CR1 = BM_SPE | BM_MSTR | BM_CPOL | BM_CPHA | BM_LSBFE;
    // transfer data
    PTH = S12CO_SPICS0_RD;              // enable /CS0 (reading)
    xferSPI0(regno);                    // send register number to RTC
    result = xferSPI0(0) >> 4;          // receive data from RTC
    PTH = S12CO_SPICSH;                 // disable all /CSx
    return result;
    }

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

// Func: putRTC()
// Args: regno[0..3] is RTC register number
//       data[0..3] is RTC data (for that register)
// Retn: -
//
void putRTC(UINT8 regno, UINT8 data) {

    // set up SPI mode: enable, master, CPOL=1, CPHA=1, LSB first
    SPI0CR1 = BM_SPE | BM_MSTR | BM_CPOL | BM_CPHA | BM_LSBFE;
    // transfer data
    PTH = S12CO_SPICS0_WR;              // enable /CS0 (writing)
    xferSPI0((data << 4) | (regno & 0x0f));
    PTH = S12CO_SPICSH;                 // disable all /CSx
    }

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

On the S12compact, the RTC (including BT1) is an optional component (not included in the standard version).

## A/D-Converter (ADC)

The S12compact can be equipped with a high-resolution A/D-Converter (ADC). While the on-chip ATD modules of the HCS12 provide sufficient resolution (10-bit) for a wide range of industrial applications, the external ADC system on the S12compact provides enhanced accuracy, which is suitable for measuring devices and data loggers.

The ADS8344 (IC11) from Burr Brown is an 8-channel 16-bit ADC with serial interface and a conversion rate of up to 100ksps. It is connected to the SPI-port SPI0 of the MCU. /SPICS2 is the chip select signal for the ADC (delivered by IC4).

IC10 is a precision voltage reference, delivering 4.096V for the ADC and the DAC on the S12compact. This reference voltage determines the maximum limit for the analog inputs AIN0..AIN7. The lower limit is ground (0V). After opening solder bridge BR9, an external reference voltage can be used instead of IC10 (note: influence on DAC IC12). The analog inputs AIN0 to AIN7 as well as the analog supply voltages (VCCA, VREF, GNDA) are available at connector X4.

The following software example for the ADS8344 uses the basic SPI functions (shown above).

```
//-------------------------------------------------------------------------
// Note: CPHA=0, CPOL=0 required
//       clock rate max. 2MHz
//       conversion takes max. 8µs
//
UINT16 getADS8344(UINT8 channel) {

    volatile UINT8 n;
    UINT16 adcresult;

    // set up SPI mode
    SPI0CR1 = BM_SPE | BM_MSTR;         // CPOL=0 CPHA=0
    // send conversion command
    PTH = S12CO_SPICS2;                 // enable /CS2
    xferSPI0((channel << 4) | 0x86);    // single ended, internal clock mode
    PTH = S12CO_SPICSH;                 // disable /CS2
    // wait 8µs
    for(n=0; n<100; n++) ;
    // get conversion result
    PTH = S12CO_SPICS2;                 // enable /CS2
    adcresult = xferSPI0(0) << 8;       // get bits 15..9
    adcresult += xferSPI0(0);           // get bits 8..1
    adcresult <<= 1;
    if(xferSPI0(0) & 0x80) adcresult++; // get bit 0
    PTH = S12CO_SPICSH;                 // disable /CS2
    return adcresult;
    }

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

On the S12compact, the ADC is an optional component (not included in the standard version).

## D/A-Converter (DAC)

The 16-bit D/A-Converter (IC12) is a DAC8532 from Burr Brown. This device contains two output channels (VOUTA, VOUTB), which can be set either simultaneously or individually.

The reference voltage (4.096V) is deliverd by IC10 (see ADC description).

The load resistance at the output(s) should be at least 2kOhm. At power-up, the output registers of the DAC are reset, thus delivering 0V.

The DAC is controlled via SPI0. /SPICS1 serves as chip select signal for the DAC.

The analog outputs VOUTA and VOUTB are available at connector X4.

The following listing shows a software example for the DAC:

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

// Note: CPHA=0, CPOL=0 required
//
void putDAC8532(UINT8 channel, UINT16 value) {

    // set up SPI mode
    SPI0CR1 = BM_SPE | BM_MSTR | BM_CPHA; // CPOL=0 CPHA=1
    // send command
    PTH = S12CO_SPICS1;                 // enable /CS1
    if((channel & 1) == 0)
        xferSPI0(0x10);                 // load & set DACA
    else
        xferSPI0(0x24);                 // load & set DACB
    // send data
    xferSPI0(value >> 8);               // send bits 15..8
    xferSPI0(value);                    // send bits 7..0
    PTH = S12CO_SPICSH;                 // disable /CS1
    }

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

On the S12compact, the DAC is an optional component (not included in the standard version).

## Serial Data Flash (SDF)

The Serial Data Flash (SDF) on the S12compact allows to store a large amount (up to 2MB / 16Mbit) of data, arriving sequentially at high rates. This is often the case for data logger applications.

Normally, a disadvantage of Flash memory is the lack of a randomly writeable buffer space. As a consequence, an additional RAM area is required for temporary storage. To avoid the need for extra RAM resources, Atmel provides two 528 byte buffers in their Serial Data Flash devices. This allows to fill one buffer via SPI, while the contents of the other buffer is programmed into a Flash sector. Using this method, a continous throughput of up to 35KB/s (writing) can be reached.

The Serial Data Flash (IC14) is connected to SPI1, thus, being independent from RTC, ADC, DAC etc., which are using SPI0. The SPI1 signals MISO1, MOSI1, SCK1 and the chip select signal /SS1 are provided by the MCU's port pins PH0..PH3.

Note: To use this assignment, bit 5 in the Module Routing Register (MODRR) of the MC9S12DP512 must be set. Otherwise (reset default) SPI1 is associated with port pins PP0..PP3.

Programming voltage and timing is generated on-chip by IC14. The supply voltage of 3.3V is provided by a low-dropout voltage regulator (IC13). The inputs of the Serial Data Flash device are 5V-tolerant, the output SO is connected to the 5V-based MISO1 signal through a level shifter (IC6B).

On the S12compact, the SDF is an optional component (not included in the standard version). If IC14 is not equipped, R17 will provide a defined input level for IC6B. By opening solder bridge BR10, the output of IC6B can be decoupled from MISO1.

## IIC-Bus

The port pins PJ6 and PJ7 grant access to the Inter-IC-Bus module (IIC/I2C/I²C) of the MC9S12DP512. Since the IIC-Bus is implemented as a hardware module, an IIC software emulation is obsolete.

For the two IIC-Bus signals (SDA, SCL), pull-up resistors are required. They can be soldered in directly on the S12compact PCB (R10, R11) or they can be added externally.

The following listing shows a simplified Master Mode implementation without using interrupts:

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

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

#include "datatypes.h"
#include <mc9s12dp512.h>
#include "s12_iic.h"

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

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

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

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

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

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

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

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

    IBCR |= BM_RSTA;                    // issue RESTART condition
    }

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

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

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

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

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

    UINT8 stat;

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

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

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

    UINT8 bval;

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

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

The IIC-Bus signals are acessible at X4/65+66.

## CAN Interface

The MC9S12DP512 contains five independent CAN-Modules, designated as CAN0 to CAN4.

CAN0 communicates over MCU pins PM0 and PM1 with IC5, which serves as the CAN bus physical interface. CAN bus signals CANH und CANL can be accessed at X4/63 and X4/64.

R9 determines the slope control setting for the CAN bus signals. To operate IC5 in High Speed mode, R9 must be shorted (refer to the PCA82C251 data sheet for details).

R8 is a termination resistor, required if the S12compact is the last node in a CAN bus chain. BR2 should be closed in this case, otherwise opened.

For CAN1 to CAN4, no physical drivers are provided on the S12compact. They can be added externally, if required.

TTL signals für CAN1 to CAN3 are available at Port M. Please refer to the schematic diagram to see if other functions (which share the same pins) are needed in your application. TTL signals for CAN4 are available through port pins PJ6 and PJ7. However, in this case a conflict with the IIC-Bus will occur, since both functions share the same two pins. If IIC and CAN4 have to be used at the same time, CAN4 can be re-routed to port pins PM4/5 or PM6/7 by setting the re-routing control register MODRR accordingly (which, in return, may have an influence on using an IF-Module or the USB interface option).

The following listing demonstrates a number of basic functions for CAN communication:

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

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

## Bus Interface

The MCU ports A, B, K and (partly) E are related to the Multiplexed External Bus Interface (MEBI). On the S12compact, all bus signals are accessible via two header connectors (X7, X8; not mounted by default).

A small memory expansion PCB can be plugged onto these two connectors, which is especially useful for debugging purposes (Flash emulation).

In Single Chip Mode, which is the default MCU operating mode of the S12compact, the Multiplexed External Bus Interface is not used and the ports A, B, K can be used as general-purpose I/O-ports.

# 7. 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 S12compact, the Flash module of the MCU contains Motorola's Monitor Program D-Bug12. The reset vector points to the start of this Monitor Software. As a result, the monitor will start immediately after reset.

The monitor functions are described in a seperate document (D-Bug12 Reference Guide, see product CD).

## Startup Code

Every Microcontroller firmware starts with a number of hardware initialization commands. For the S12compact, 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 S12compact Controller Module will be published on our Website, as it becomes available:

http://elmicro.com/en/s12compact.html

# 8. TwinPEEKs Monitor

Software Version 2.3

## Serial Communication

TwinPEEKs communicates over the first RS232 interface ("SER0", X3) at **19200 Baud**. Settings are: 8N1, no hardware or software handshake, no protocol.

## Autostart Function

After reset, the TwinPEEKs monitor checks, whether port pins PE5 (MODA) and PE6 (MODB) are connected (X1B pins 1+2). 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 and EEPROM

The CPU can read every single byte of the microcontroller's resources - the type of memory does not matter. However, for write accesses, some rules have to be followed: Flash and EEPROM have 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 Freescale Tool SRECCVT:

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

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

Please note, that it is not possible to program or erase the part of Flash memory that contains the monitor code. Also, the last 16 bytes of the EEPROM block are reserved for system use.

## 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 $3FEE         ; IRQ pseudo vector
ldd  #isrFunc      ; ISR address to
std  $3FEF         ; 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 *)0x3fee) = 0x06;    // JMP opcode
*((void (**)(void))0x3fef) = 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):

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

## 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 Freescale guidelines. The valid address range for the MC9S12DP512 starts at 0x080000 (0x20 * 16KB) and ends at 0x0FFFFF (0x40 * 16 KB - 1).

Before loading into non-volatile memory (EEPROM, Flash), 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.

### Erase EEPROM

**Syntax: Y [sadr]**

Erase one sector (double word = 4 byte) of EEPROM memory. The sector is specified by it's starting address sadr (bits 0 and 1 of sadr are "don't care").

If sadr is not specified, the whole EEPROM will be erased after user confirmation.

# 9. Memory Map

The memory map of the microcontroller is initialized by the TwinPEEKs monitor as follows (Note: partly different from reset default values!):

### S12compact.DP512

| Begin | End | Ressource |
|---|---|---|
| $0000 | $03FF | Control Registers |
| $0400 | **$07FF** | 1KB (of total 4KB) EEPROM<br>(the area below $0400 is hidden by control registers, the top 2048 bytes by the RAM!) |
| **$0800** | $3FFF | 14KB RAM<br>TwinPEEKs uses the top 512 bytes |
| $4000 | $7FFF | 16KB Flash (equals Page $3E) |
| $8000 | $BFFF | 16KB Flash page $20<br>(any Page **$20**..$3F, selectable by PPAGE) |
| $C000 | $FFFF | 16KB Flash (equals Page $3F)<br>TwinPEEKs uses the top 4KB |

### Note:

Due to a mask set erratum of the MC9S12DP512 Mask Set 4L00M (and earlier) not only the monitor code in page $3F is write protected, but also an additional area starting at $B000 up to $BFFF in page $3B. Consequently, the monitor can not download user code to this region.

However, the whole Flash memory (including the write protected areas) can be programmed using a BDM tool at any desired time.