LVCS12

Hardware-
Version 1.10

User Manual

October 13 2004

MCT Elektronikladen GbR
Hohe Str. 9-13, D-04107 Leipzig
Phone: +49-(0)341-2118354
Fax: +49-(0)341-2118355
Email: support@elmicro.com
Web: http://elmicro.com

Contents

1. Overview

LVCS12 is an easy applicable, credit card-sized Controller Module, based on the 16-bit microcontroller family HCS12 by Motorola. The LVCS12 module provides an easy way to evaluate the Microcontroller. It is a versatile tool for rapid prototyping and a very cost-effective, off-the-shelf solution for low- and mid-volume series applications.

The LVCS12 is equipped with a powerful MC9S12E128 microcontroller unit (MCU). It contains a 16-bit HCS12 CPU, 128KB of Flash memory, 8KB RAM and a large amount of peripheral function blocks, such as SCI (3x), SPI, IIC, Timer, PWM, ADC, DAC and General-Purpose-I/Os. The MC9S12E128 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.

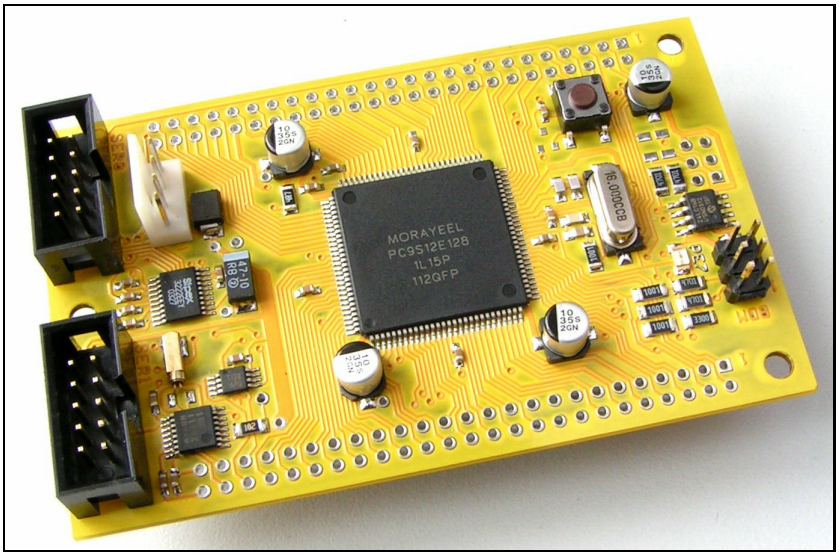
Technical Data

- MCU MC9S12E128 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
- 128KB Flash
- 8KB RAM
- 3x SCI - asynch. serial interface (e.g. RS232, LIN)
- 1x SPI - synch. serial interface
- 1x IIC - Inter-IC-Bus
- 12x 16-Bit Timer (Input Capture/Output Compare)
- 12x PWM (Pulse Width Modulator)
- 16-channel 10-bit A/D-Converter
- 2-channel 8-bit D/A-Converter

- Integrated LVI-circuit (reset controller)
- BDM - Background Debug Mode Interface with standard 6-pin connector available for download & debugging
- two serial interfaces equipped with RS232 transceiver (e.g. for PC connection)
- 2nd serial port can directly drive a serial LC display unit
- 3rd serial port available with CMOS level
- Real Time Clock (RTC) provides time, calendar and alarm functions; accuracy can be further increased by software calibration, 3V LiMn battery-buffered
- 16 kbit Serial EEPROM
- DAC channels equipped with output amplifier (rail-to-rail opamp)
- Indicator-LED
- Reset Button
- up to 87 free general-purpose I/Os
- all I/O-signals signals brought out on header connectors
- 3V..5V operating voltage, current consumption 50 mA typ.
- Mech. Dimensions: 2.1" x 3.4"

Package Contents

- Controller Module with MC9S12E128
- TwinPEEKs Monitor (in the MCU's Flash Memory)
- RS232 cable (Sub-D9)
- two header connectors (2x25 pins each), power connector
- User Manual (this document)
- Schematic Diagrams
- CD-ROM: contains assembler software, data sheets, CPU12 Reference Manual, code examples, C-compiler (evaluation version), etc.



Controller Module LVCS12

2. Quick Start

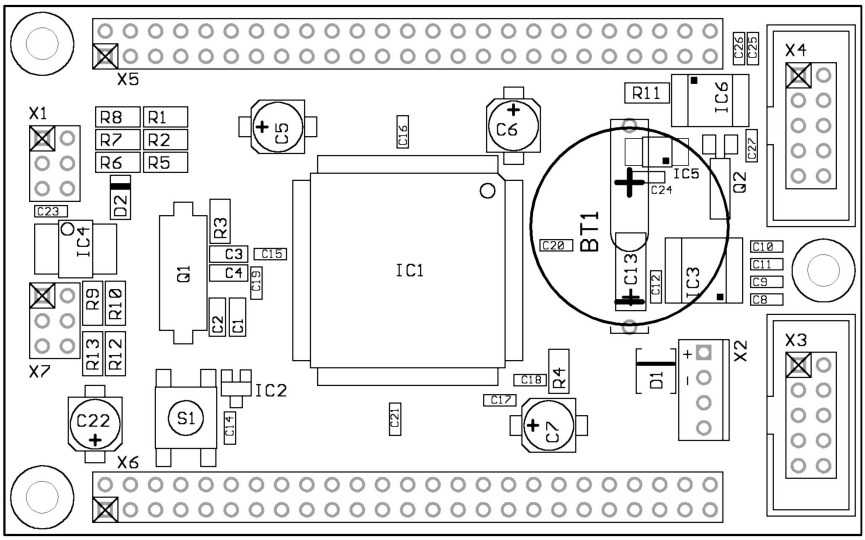
Nobody likes to read big manuals. For that reason we will summarize the most important things in the following section. If you need additional information, please refer to the more detailed sections of this manual.

Here is how you can start:

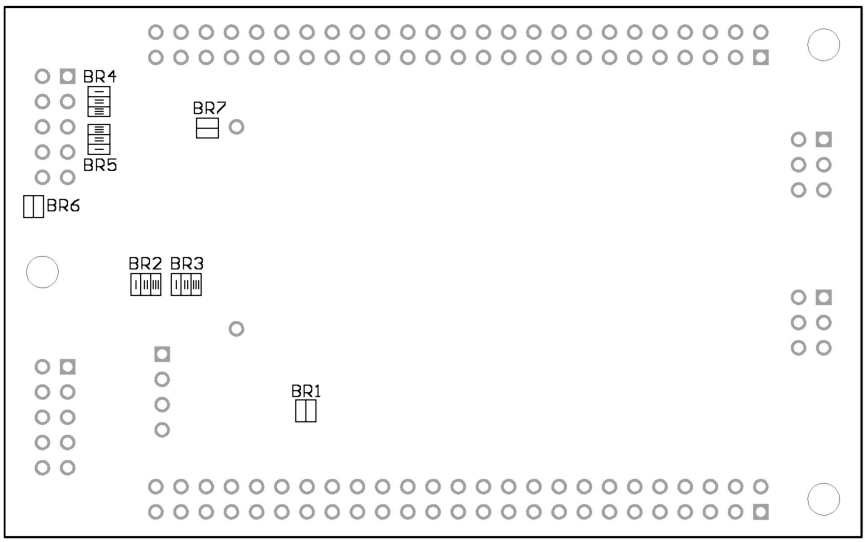
- Please check the board for any damages due to transportation
- Connect the Controller Module via RS232 to a PC. The connection between LVCS12 (interface SER0, connector X3) and PC can be established using the flat ribbon cable which is in the box.
- On the PC, start a terminal program. An easy to use 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 (!) DC power supply, e.g. here:
- GND to X2 pin 2
- +5V to X2 pin 1
- 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 LVCS12!

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 present on this board.

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: RxOUT

1-2*	R1OUT/R2OUT drives PS0/PS2
2-3	R1OUT/R2OUT disabled (Tristate)

BR3: SHDN

1-2*	IC3 always active (/SHDN = H)
2-3	use PP5 to activate/deactivate IC3

BR4, BR5: RS232 TxD/RxD Select (SER1/X4)

1-2*	RS232 configured as "device" (connection to a PC, etc.)
2-3	RS232 configured as "host" (connection to a serial LCD, etc.)

* = Factory Default Setting

BR6: LCD Power Supply (SER1/X4)

open*	VCC not available on RS232 port SER1 (standard Sub-D connector layout)
closed	VCC available on RS232 port SER1 (at Pin 9 of the Sub-D connector)

BR7: RRTC

open*	RTC can not cause system reset
closed	RTC can cause system reset

* = Factory Default Setting

5. Mechanical Dimensions

The following table summarizes the mechanical dimensions of the LVCS12. 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 horizontally, 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 in inch	**Y in inch**
X1	0,150	1,575
X2	2,775	0,725
X3	3,150	0,675
X4	3,150	1,825
X5	0,400	1,900
X6	0,400	0,100
X7	0,150	0,950
B1	0,150	0,150
B2	0,150	1,950
B3	3,250	1,050
PCB	3,400	2,100

6. Circuit Description

In this section, a number of details will be presented on how to work with the HCS12 in general and the LVCS12 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 LVCS12 is provided as a separate document.

Controller Core, Power Supply

VDDR/VSSR, VDDX/VSSX and VDDA/VSSA are the three supply pin pairs of the MC9S12E128. The nominal operating voltage (designated as VCC in the schematic diagram) of this microcontroller unit (MCU) ranges from 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 3..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 VCC levels.

The three terminal pairs mentioned above must be decoupled carefully. A ceramic capacitor of 100nF is connected directly at each pair (C15, C16, C17). A 10µF (electrolytic or tantalum) capacitor per

node is added, 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 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 connected to VDDA via solder bridge BR1. C18 is used for decoupling. VRH can be supplied externally when 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 VDDA. 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 MC9S12E128 contains 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, IC2 can be added as an external LVI circuit. IC2 is equipped with an open-drain output in order to prevent collisions with the MCU's bidirectional reset pin. The /RESET signal is high while in inactive state because IC2 contains an integrated pull-up resistor (approx. 5kOhm). Consequently, R6 is not needed if IC2 is equipped.

The reset pulse issued by IC2 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. 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 MC9S12Exx 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 LVCS12, the MC9S12E128 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 LVCS12 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 16MHz, the internal bus speed (ECLK) becomes 8MHz by default. To realize higher bus clock rates, the PLL has to be engaged. The MC9S12Exx 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 <hcs12dp256.h>
#include "s12_crg.h"

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

void initPLL(void) {

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

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

R5 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 MC9S12Exx.

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 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 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 MC9S12Exx contains a 10-bit Analog-to-Digital Converter modules. The module (ATD) provides 16 multiplexed input channels.

VRH is the upper reference voltage for all A/D-channels. On the LVCS12, VRH is connected to VDDA (VCC) through solder bridge BR1. After opening BR1, it is possible to use an external reference voltage.

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

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

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

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

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

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

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

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

Integrated D/A-Converter

The MC9S12E128 provides two analog output signals at port pins PM0 and PM1. These signals are generated by two D/A-converter modules (DAC0, DAC1), providing 8 bit resolution each. The DAC module outputs can only drive very light loads. Therefore, each channel is equipped with an external operational amplifier in voltage follower configuration (IC5A, IC5B). The output signals of these amplifiers can be accessed at X5/45+46.

The software needed to operate the DAC is quite simple, as illustrated in the following source listing:

```
//============================================================================
// File: S12_DAC.C - V1.00
//============================================================================

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

#include <mc9s12e128.h>
#include "datatypes.h"
#include "s12_dac.h"

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

// Func: Initialize DAC0 module
// Args: -
// Retn: -
//
void initDAC0(void) {

    // enable DAC module
    // use right-justified unsigned data
    // output enable
    DAC0D  = 0;
    DAC0C0 = BM_DACE | BM_DJM | BM_DACOE;
    }

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

// Func: set DAC0 output
// Args: 8 bit value
// Retn: -
//
void setDAC0(UINT8 value) {

    DAC0DL = value;
    }

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

When the DAC access rate is very high, it could be better to replace the setDAC0() function by a macro:

```
#define setDAC0(b)   DAC0DL = b
```

RS232 Interfaces

The MC9S12Exx provides three asynchronous serial interfaces (SCI0, SCI1, SCI2). Each interface has one receive line and one transmit line (RXDx, TXDx). 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 LVCS12, the signals of two SCIs are connected to the RS232 line transceiver circuit IC3. If the RS232 interface is not needed in an application, the outputs R1OUT and R2OUTof IC3 can be tri-stated by connecting contacts 2-3 of solder bridge BR2. As a consequence, the MCU signals PS0...PS3 can be used as additional general-purpose I/Os.

To reduce current consumption, IC3 can be brought into suspend mode by setting solder bridge BR3 to position 2-3. Now, the MCU's signal PP5 can be used to control the /SHDN input of the RS232 transceiver chip. Low level activates the power-saving suspend mode of IC3.

X3 (SCI0) is used as the primary RS232 interface. To connect the LVCS12 to a PC, a 10-wire flat ribbon cable can be used. The cable must have a 10-pin female header connector at the LVCS12 side (X3) and a female Sub-D9 connector at the PC side.

The above is valid for X4 (SCI1) as well, provided that BR4 and BR5 are in position 1-2 (default state). In this case, the PC serves as the host and LVCS12 is configured as device.

The reverse configuration can be used to connect a serial LC display to X4. In this case, the LVCS12 is the host and the LCD is the device. The required signal crossing is done by changing BR4 and BR5 to position 2-3. Additionally, it might be useful to close BR6 in order to supply the LCD module via pin 9 of the Sub-D9 connector (Caution: this is not conform with RS232 standard!).

Serial, alphanumeric LC-Displays are offered by a number of manufacturers, such as the Canadian company Matrix Orbital (http://www.matrixorbital.com).

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

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

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

#include <mc9s12dp256.h>
#include "datatypes.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 MC9S12D64 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 PS4 to PS7). These signals are not used on-bord the LVCS12, though they can be accessed through the header connectors at the edges of the board.

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 - V1.00
//============================================================================

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

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

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

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

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

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

UINT8 xferSPI0(UINT8 abyte) {

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

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

IIC-Bus

The MC9S12Exx offers an Inter-IC-Bus (IIC/I2C/I^{2}C) connection on port pins PM6 and PM7. This function is supported by an integrated hardware module, not a software emulation.

The bus lines (SDA, SCL) are equipped with pull-up resistors (R9, R10).

On the LVCS12 module, the Real Time Clock (IC6) and the serial EEPROM (IC4) are controlled by the IIC bus. The bus signals can also be used externally (X5/47+48).

The file S12_IIC.C contains a demo implementation for the IIC module in master mode using polling. Motorola's Application Note AN2318 provides further reading, including suggestions for the implementation of an interrupt-driven IIC handler.

Serial EEPROM

The MC9S12Exx MCUs do not contain any EEPROM, so the serial memory device IC4 was added to provide 16 kbit non-volatile memory space (up to 256 kbit, optionally). It is connected to the IIC bus interface.

The source file LVCS12_SEEP.C demonstrates how to handle this device:

```
//=============================================================================
// File: LVCSS12_SEEP.C - V1.00
//=============================================================================

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

#include "datatypes.h"
#include "s12_iic.h"
#include "lvcs12_seep.h"

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

// device address of 24LC16B
#define SEEP_DEVICE_ID  0x50

//-- 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;
    b = (addr >> 7) & 0x0e;
    b += SEEP_DEVICE_ID << 1;
    startIIC();
    if(sendIIC(b + IIC_WRITE) != IIC_ACK)
        SEEP_ErrorCode = SEEP_EC_NOTRDY;
    else {
        if(sendIIC((UINT8)addr) != IIC_ACK)
            SEEP_ErrorCode = SEEP_EC_ADDRERR;
        else {
            restartIIC();
            if(sendIIC(b + 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) {

    UINT8 b;

    SEEP_ErrorCode = SEEP_EC_OK;
    b = (addr >> 7) & 0x0e;
    b += SEEP_DEVICE_ID << 1;

    startIIC();
    if(sendIIC(b + IIC_WRITE) != IIC_ACK)
        SEEP_ErrorCode = SEEP_EC_NOTRDY;
    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;
    }

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

Indicator LED

Port pin PE7 drives an indicator LED (D2). To control this LED, some simple macros can be used, as shown in the following C header file:

```
//===========================================================================
// File: LVCS12_LED.H - V1.00
//===========================================================================

#ifndef __LVCS12_LED_H
#define __LVCS12_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 //__LVCS12_LED_H ==================================================
```

Real Time Clock

The LVCS12 module contains 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 open-drain output pin /INTR of the RTC is brought out to X6/8 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 (BT1) provides a backup supply in case the main power (VCC) fails. BT1 is a 3V LiMn primary battery. The switchover to backup power is done automatically by the RTC whenever VCC falls below 2.4V. Under this condition, the /VDCC output of the RTC is driven low. By closing BR7, this signal can be used as an additional system reset source.

LVCS12_RTC.C contains a set of functions to control the RTC on the LVCS12.

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 LVCS12, the MCU's Flash boot block ($F000-$FFFF) contains the TwinPEEKs Monitor Program. The reset vector points to the start of this Monitor firmware. As a result, the monitor will start immediately after reset (for details refer to the Monitor description below).

Startup Code

Every microcontroller firmware starts with a number of hardware initialization commands. For the LVCS12, 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

Any additional information about the LVCS12 Controller Module will be published on our website, as it becomes available:

http://elmicro.com/lvcs12.html

8. TwinPEEKs Monitor

Software Version 2.2

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 detects if port pin PT4 is connected to port pin PT5. 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, some rules have to be followed: 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.

A problem occurs, if the byte stream is continued in the subsequent 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).

Please note, that it is not possible to program or erase the part of Flash memory that contains the monitor code.

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         ; PWM Emergency Shutdown
FF8A : 3F52             dc.w  TP_RAMTOP-174         ; VREG LVI
FF8C : 3F55             dc.w  TP_RAMTOP-171         ; PMF Fault 3
FF8E : 3F58             dc.w  TP_RAMTOP-168         ; PMF Fault 2
FF90 : 3F5B             dc.w  TP_RAMTOP-165         ; PMF Fault 1
FF92 : 3F5E             dc.w  TP_RAMTOP-162         ; PMF Fault 0
FF94 : 3F61             dc.w  TP_RAMTOP-159         ; PMF Gen C reload
FF96 : 3F64             dc.w  TP_RAMTOP-156         ; PMF Gen B reload
FF98 : 3F67             dc.w  TP_RAMTOP-153         ; PMF Gen A reload
FF9A : 3F6A             dc.w  TP_RAMTOP-150         ; T2 Pulse Accu Input Edge
FF9C : 3F6D             dc.w  TP_RAMTOP-147         ; T2 Pulse Accu Overflow
FF9E : 3F70             dc.w  TP_RAMTOP-144         ; Timer 2 Overflow
FFA0 : 3F73             dc.w  TP_RAMTOP-141         ; Timer 2 channel 7
FFA2 : 3F76             dc.w  TP_RAMTOP-138         ; Timer 2 channel 6
FFA4 : 3F79             dc.w  TP_RAMTOP-135         ; Timer 2 channel 5
FFA6 : 3F7C             dc.w  TP_RAMTOP-132         ; Timer 2 channel 4
FFA8 : 3F7F             dc.w  TP_RAMTOP-129         ; reserved
FFAA : 3F82             dc.w  TP_RAMTOP-126         ; T1 Pulse Accu Input Edge
FFAC : 3F85             dc.w  TP_RAMTOP-123         ; T1 Pulse Accu Overflow
FFAE : 3F88             dc.w  TP_RAMTOP-120         ; Timer 1 Overflow
FFB0 : 3F8B             dc.w  TP_RAMTOP-117         ; Timer 1 channel 7
FFB2 : 3F8E             dc.w  TP_RAMTOP-114         ; Timer 1 channel 6
FFB4 : 3F91             dc.w  TP_RAMTOP-111         ; Timer 1 channel 5
FFB6 : 3F94             dc.w  TP_RAMTOP-108         ; Timer 1 channel 4
FFB8 : 3F97             dc.w  TP_RAMTOP-105         ; FLASH
FFBA : 3F9A             dc.w  TP_RAMTOP-102         ; reserved
FFBC : 3F9D             dc.w  TP_RAMTOP-99          ; reserved
FFBE : 3FA0             dc.w  TP_RAMTOP-96          ; reserved
FFC0 : 3FA3             dc.w  TP_RAMTOP-93          ; IIC
FFC2 : 3FA6             dc.w  TP_RAMTOP-90          ; reserved
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          ; reserved
FFCA : 3FB2             dc.w  TP_RAMTOP-78          ; reserved
FFCC : 3FB5             dc.w  TP_RAMTOP-75          ; reserved
FFCE : 3FB8             dc.w  TP_RAMTOP-72          ; Port AD
FFD0 : 3FBB             dc.w  TP_RAMTOP-69          ; ATD
FFD2 : 3FBE             dc.w  TP_RAMTOP-66          ; SCI2
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          ; T0 Pulse Accu Input Edge
FFDC : 3FCD             dc.w  TP_RAMTOP-51          ; T0 Pulse Accu Overflow
FFDE : 3FD0             dc.w  TP_RAMTOP-48          ; Timer 0 Overflow
FFE0 : 3FD3             dc.w  TP_RAMTOP-45          ; Timer 0 channel 7
FFE2 : 3FD6             dc.w  TP_RAMTOP-42          ; Timer 0 channel 6
FFE4 : 3FD9             dc.w  TP_RAMTOP-39          ; Timer 0 channel 5
FFE6 : 3FDC             dc.w  TP_RAMTOP-36          ; Timer 0 channel 4
FFE8 : 3FDF             dc.w  TP_RAMTOP-33          ; reserved
FFEA : 3FE2             dc.w  TP_RAMTOP-30          ; reserved
FFEC : 3FE5             dc.w  TP_RAMTOP-27          ; reserved
FFEE : 3FE8             dc.w  TP_RAMTOP-24          ; reserved
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

All TwinPEEKs commands start with 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 MC9S12E128 starts at 0xE0000 (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.

9. Memory Map

The memory map of the microcontroller is initialized by the TwinPEEKs monitor as follows (please note: some settings are different from reset default values!):

LVCS12.E128

Begin	End	Ressource
$0000	$03FF	Control Registers
$2000	**$3FFF**	8KB RAM (reset default: $0000-$1FFF) TwinPEEKs uses the top 512 bytes
$4000	$7FFF	16KB Flash (equals Page $3E)
$8000	$BFFF	16KB Flash page $38 (any Page $38..$3F, selectable by PPAGE)
$C000	$FFFF	16KB Flash (equals Page $3F) TwinPEEKs uses the top 4KB