

---

# HC12compact

Hardware Version 1.0

## User Manual

July 7 2008

---

Copyright (C)1997-2008 by  
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>

This manual and the product described herein were designed carefully by the manufacturer. We have made every effort to avoid mistakes but we cannot guarantee that it is 100% free of errors.

The manufacturer's entire liability and your exclusive remedy shall be, at the manufacturer's option, return of the price paid or repair or replacement of the product. The manufacturer disclaims all other warranties, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the product including accompanying written material, hardware, and firmware.

In no event shall the manufacturer or its supplier be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the product, even if the manufacturer has been advised of the possibility of such damages. The product is not designed, intended or authorized for use in applications in which the failure of the product could create a situation where personal injury or death may occur. Should you use the product for any such unintended or unauthorized application, you shall indemnify and hold the manufacturer and its suppliers harmless against all claims, even if such claim alleges that the manufacturer was negligent regarding the design or implementation of the product.

Product features and prices may change without notice.

All trademarks are property of their respective holders.

---

---

# Contents

---

<b>1. Overview</b> .....	<b>3</b>
Package Contents .....	4
<b>2. Quick Start</b> .....	<b>5</b>
<b>3. Parts Location Diagram</b> .....	<b>6</b>
<b>4. Jumpers and Solder Bridges</b> .....	<b>7</b>
Jumpers .....	7
Solder Bridges .....	8
<b>5. Connectors</b> .....	<b>10</b>
<b>6. Memory</b> .....	<b>11</b>
Memory Interface .....	11
Addressing, Bank Switching .....	12
Flash-Programming .....	14
<b>7. RS232 Interface</b> .....	<b>15</b>
<b>8. Peripherals on the SPI-Bus</b> .....	<b>17</b>
Real Time Clock .....	18
A/D-Converter .....	19
D/A-Converter .....	20
<b>9. Peripherals on the System Bus</b> .....	<b>22</b>
CAN-Controller .....	22
LCD-Interface .....	24
<b>10. Power Management</b> .....	<b>27</b>
<b>11. Application Hints</b> .....	<b>28</b>
About Operating Modes .....	28
Initialization Example .....	29
Schematic Diagram .....	31
Additional Information on the Web .....	31

<b>12. Monitor Program TwinPEEKs</b> .....	<b>32</b>
Introduction .....	32
Notation .....	32
Write Access .....	33
Autostart Function .....	34
Interrupt Vectors .....	35
Memory Map .....	36
Monitor Commands .....	37

# 1. Overview

---

HC12compact is an universal microcontroller module on the basis of a Motorola MC68HC812A4 MCU.

In addition to the on-chip features of the MCU itself, the following peripherals are available on the HC12compact:

- 512 KB Flash-Memory and 256 KB (optional: 1024 KB) RAM
- Real Time Clock (battery backed)
- Analog/Digital-Converter (12 bit, 11 channels)
- Digital/Analog-Converter (12 bit, 2 channels)
- CAN Controller
- RS232 interface driver
- beeper
- indicator LED

Key features of the HC12compact unit are:

- compact design
- low power consumption
- easy handling
- comprehensive software support available (Monitor, C-Compiler, BDM-Debugger etc.)

## Package Contents

---

The base version (stock code HC12CO/1) of HC12compact is equipped as follows:

- Ready-to-use controller board with 256 KB RAM and 512 KB Flash
- None of the options are populated (order RTC, ADC, DAC and CAN seperately, but always together with the board)
- The header connectors at both edges of the board are not mounted (so the user may solder them up- or downward, depending on the application)
- User Manual (this document) and software on CD-ROM
- TwinPEEKs Monitor Program, residing in the internal EEPROM of the HC12
- Serial cable with Sub-D9 connector (PC side)

The fully equipped version (stock code HC12CO/FULL) includes the following additional peripherals:

- Real Time Clock and LiMn battery
- A/D Converter (12 bits, 11 channels)
- D/A Converter (12 bits, 2 channels)
- CAN Controller SJA1000

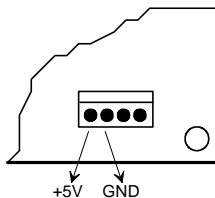
## 2. Quick Start

---

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:

- First check the board for (mechanical) damages!
- Check the default jumper settings (see below). The HC12 must start in Normal Single Chip Mode (JP1/2/3=L/L/H) in order to activate its monitor program.
- Connect the single board computer via RS232 with your PC. Use the flat ribbon cable supplied with the board. Pin 1 of the header connector on the PCB goes to Pin 1 of the Sub-D9 connector on the PC side.
- 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!
- Choose a baud rate of 19200Bd, 8N1, no protocol used.
- Connect the 5V power supply (must be stabilized!). Check voltage and polarity **before!**



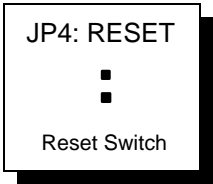
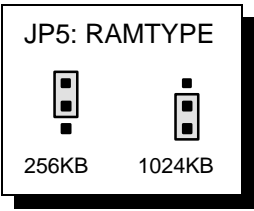
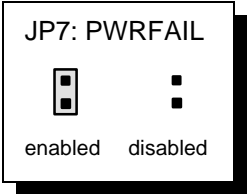
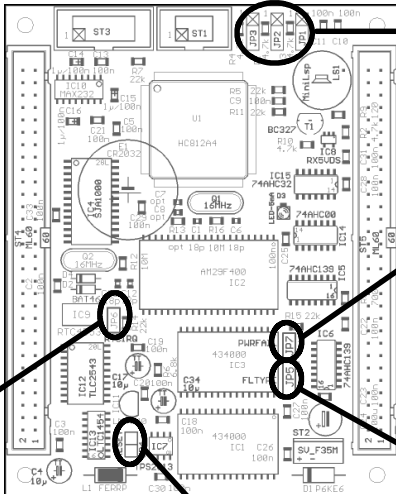
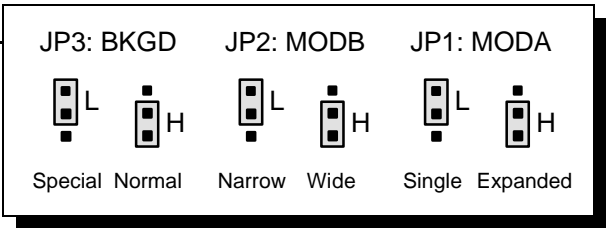
- Now the monitor software starts and sends a message. After the prompt you may type in your commands.





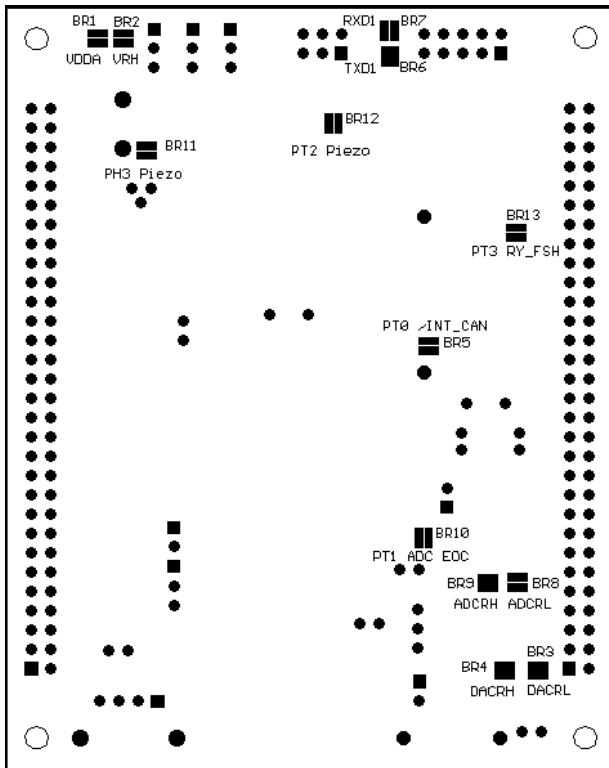
# 4. Jumpers and Solder Bridges

## Jumpers



*Jumper Locations*

## Solder Bridges

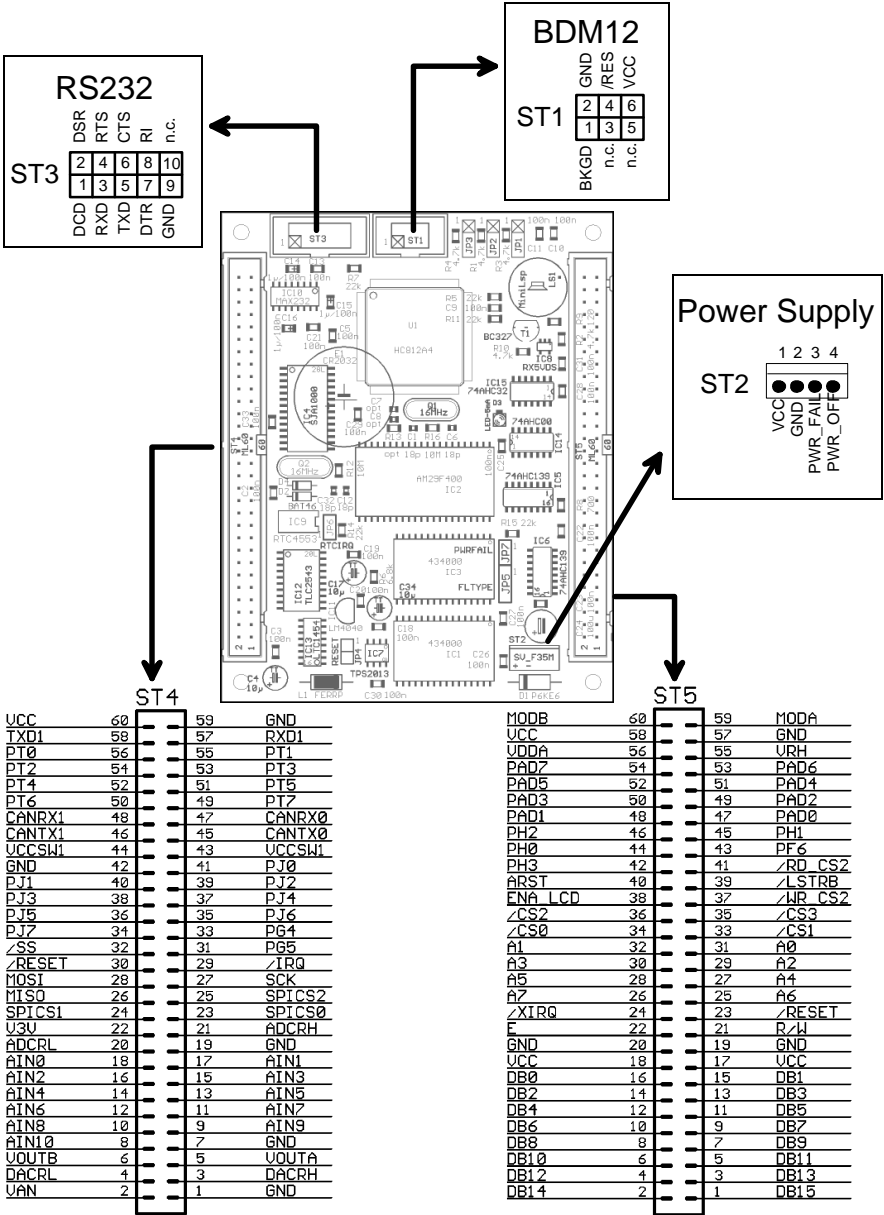


*Locations of Solder Bridges (Solder Side)*

<b>Pad</b>	<b>Function</b>	<b>Description</b>	<b>*</b>
BR1	VDDA	HC12 VDDA = VCC	X
BR2	VRH	HC12 VRH = VCC	X
BR3	DACRL	DAC XA/B = GND	X
BR4	DACRH	DAC REFHA/B = REFOUT	X
BR5	/INT_CAN->PT0	CAN Interrupt Detection	O
BR6	TXD1	TXD1 drives DTR	O
BR7	RXD1	DSR drives RXD1	O
BR8	ADCRL	ADC VREF- = GND	X
BR9	ADCRH	ADC VREF+ from IC11	X
BR10	ADC_EOC->PT1	ADC End Of Conversion Detection	O
BR11	PH3->Piezo	PH3 drives beeper	O
BR12	PT2->Piezo	PT2 drives beeper	X
BR13	RY_FSH->PT3	Flash Ready Detection	O

\* Default settings: O=open X=closed

# 5. Connectors



## 6. Memory

---

### Memory Interface

---

In part 1 of the schematic diagram (provided separately) is shown, how the memory devices are connected to the system bus. The flash memory IC2 (29F400 from AMD) has a capacity of 512 KB. Another option is to mount a smaller device like the 29F100 with 128 KB which is pin compatible to the 29F400.

Address and data lines of the flash memory go 1:1 to the system bus. Since A0 of the HC12 address bus is not used (we have a 16 Bit wide data bus!) the address bus lines used start with A1 but the flash memory addresses still start with A0 (flash device operates in word mode).

The flash memory can be selected via /CE and /OE. The /CE pin is connected to a chip select line of the MCU (/CSP0). This chip select is already active after reset in Expanded Mode. This is, /CSP0 goes low if a memory access occurs between \$8000 and \$FFFF - providing, there is no higher prioritised resource such as the internal EEPROM.

/WE and /OE control the read or write direction. The output enable signal is the negated R/W signal of the controller.

The RY/BY output may be used to detect the completion of a flash write cycle. To utilize this signal the solder pad BR13 (on the solder side of the PCB) must be closed. As a result Input Capture channel 3 of the MCU can generate an interrupt as soon as the write (or erase) cycle of the flash is finished.

There are also software means of detecting the status of the flash memory chip (e.g. Toggle Bit Polling) which simplify the handling of those memories drastically. For details see the data book from AMD.

For the RAM two 8 bit devices are used in this application. IC1 covers the upper half of the data bus, IC3 the lower half.

The chip select signal /CSD must be ORed with A0 and /LSTRB to get two independent selects for the upper and lower half of the data bus.

If the MCU makes a byte access /LSTRB has always the opposite level of A0. If an aligned word access occurs both A0 and /LSTRB are low (=active). A misaligned word access is a special case which can only occur during an access to the internal RAM, which can handle this transparently by a special byte-swapping mechanism. External memory accesses are always executed in an aligned mode.

Two types of RAM chips may be mounted on the HC12compact. To select between the 128 KB type (256 KB total, default) and the 512 KB type (1 MB RAM, special option) the jumper JP5 must be placed in the right position (see section Jumpers above)

## **Addressing, Bank Switching**

---

The HC12 has a 64 KB linear addressing area. So it is possible to use 2 byte addresses, which is very code efficient. In addition, the HC812A4 has a bank switching hardware, which can handle up to 4 MB program memory and 1 MB data memory.

There are also special machine instructions which simplify the usage of the paged memory. The CALL instruction works like a JSR instruction, but it saves also the current program page on the stack. The enhanced counterpart of RTS is RTC.

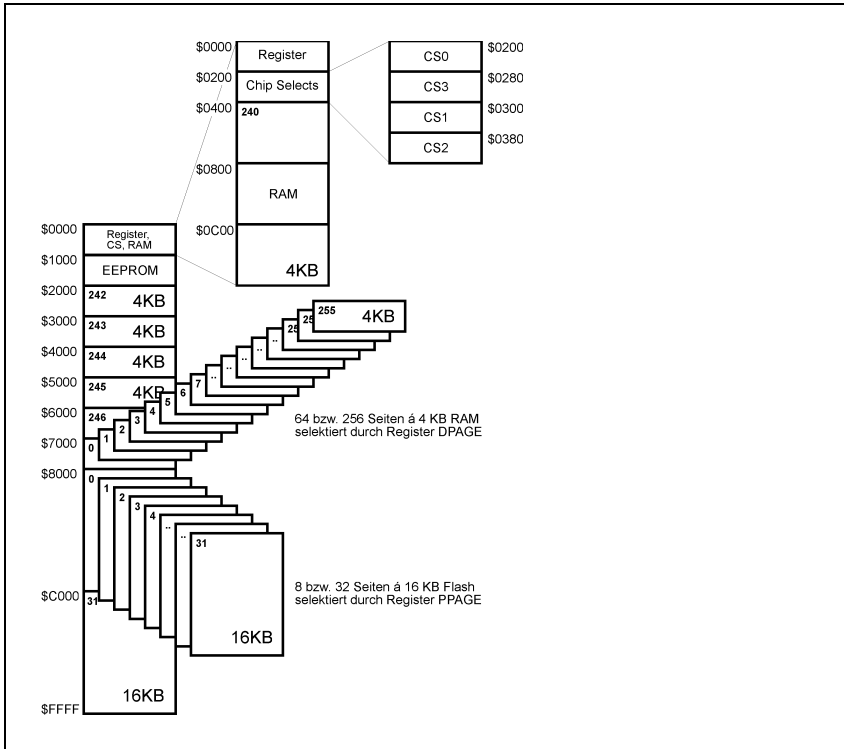
The program memory pages have a size of 16 KB and the program memory window is located at \$8000 - \$CFFF. The selected program page number is in the PPAGE register. This is an 8 bit register, so there are 256 possible program memory pages.

The window for the paged data memory is between \$7000 and \$7FFF (4 KB). Again, there are 256 combinations, selectable via the DPAGE register. There are no specific instructions to deal with the data memory.

There is another window, called Extra Window, with 1 KB page size and (again) up to 256 pages. This window is not explicitly used on the HC12compact.

For details about the whole memory paging mechanism please refer to the data book "Technical Summary" from Motorola.

The following diagram shows the "default" memory map of the :HC12compact:



*Memory Map*

The upper memory area (\$8000 to \$FFFF) can be used as program memory space, it is provided by a Flash memory device of 128-512KB. The microcontroller uses the highest 16 KB of this device to fill the memory area between \$C000 and \$FFFF. The 16 KB below is the Program Window. Any 16 KB memory page of the Flash device can be mapped into this area by selecting a specific PPAGE register value.

The lower half of the memory address space (\$0000 to \$7FFF) is filled with 32 KB of the external RAM (256 KB or 1 MB total size), except there is a higher prioritised internal resource. So the control registers of the controller occupy \$0000 to \$01FF, followed by the chip

select space (to \$03FF). From \$0400 to \$07FF there is a short block of "nomal" external RAM. The internal RAM starts at \$0800 (1 KB size). Starting at \$1000 there is the 4 KB internal EEPROM area (providing the controller is reset in Expanded Mode). \$2000 to \$6FFF is filled with (linear) external RAM again.

The 4 KB area from \$7000 to \$7FFF makes the Data Window. With 1 MB RAM on-board there are 256 possible pages. With 256 KB RAM (standard version) there are still 64 pages. Note: Some of the pages may be visible in the Data Window *and* in the linear address space at the same time, as shown in the figure above.

## Flash Programming

---

The Flash memory on the HC12compact can be programmed using the TwinPEEKs monitor program. Please refer to the AMD data sheets if you want to create your own programming functions. Maybe the following code example can be helpful:

```
-----  
; Func: Write Flash Word  
; Args: Y = Destination Address (must be even!)  
;       D = Word to write  
-----  
;  
wrFlashWord    pshx  
               pshd  
               ldaa PPAGE  
               psha  
               movb #$02,PPAGE  
               movw #$00aa,$aaaa  
               movb #$01,PPAGE  
               movw #$0055,$9554  
               movb #$02,PPAGE  
               movw #$00a0,$aaaa  
               pula  
               staa PPAGE  
               puld  
               std  0,y  
               ; /Data Polling Sequence  
               ldx  #1000                ; 1000 x 8 x 0,125µs  
_wrflcplloop   cpd  0,y                  ; [3]  
               beq  _wrflwdone           ; [1]  
               dex  0,y                  ; [1]  
_wrflwdone     bne  _wrflcplloop         ; [3]  
               pulx  
               rts  
-----
```

### *Flash Write Algorithm*



## 7. RS232 Interface

---

The HC812A4 has two internal asynchronous serial ports (SCI0 and SCI1). Each port provides an input (RxD) and an output (TxD), but no handshake signals.

Both ports work mostly independent from each other. Thus, the second SCI channel can be disabled and the two spare I/O-signals can be used as handshake signals for SCI0 (needs some user software).

In this case, or in the case that both SCI channels will be used, the solder pads BR6 and BR7 must be closed. By doing this, the HC12 port signals PS2 and PS3 are connected to a receiver and transmitter channel of the RS232 level converter IC10.

The firmware for the SCI ports is not difficult, as shown in the following example:

```

;=====
; File: SCI12.A
; Func: Serial-I/O via the Serial Communication Interface (SCI0)
;=====
;
; CPU 68HC12
; include "reghc12.inc"

SC0BD equ SC0BDH
SC0CR equ SC0CR1

; Func: Initialize SCI (9600 Baud, 8N1, Polling Mode)
; Args: -
; Retn: -
; Dest: D
;
initSCI      ldd    #26                ; 19200 Bd
             std    SC0BD
             ldd    #$000c            ; 8N1, TE + RE
             std    SC0CR              ; A:B -> SC0CR1:SC0CR2
             rts

; Func: Test if any character available (received)
; Args: -
; Retn: A = 0 (Z = 1) -> no char
;       A <> 0 (Z = 0) -> char available
; Dest: -
;
testSCI      ldaa   SC0SR1             ; read status
             anda   #$20              ; receive data reg full?
             rts                      ; returns 0, if no data

; Func: Get character from SCI (wait for)
; Args: -
; Retn: A = char
; Dest: -
;

```

# HC12compact

---

```
getSCI          brclr SC0SR1,$20,getSCI ; receive data reg full?
                ldaa SC0DRL           ; read out data
                rts

; Func: Send a character via SCI
; Args: A = char
; Retn: -
; Dest: -
;
putSCI          brclr SC0SR1,$80,putSCI ; transmit data reg empty?
                staa SC0DRL           ; send data
                rts

;=====;
```

## 8. Peripherals on the SPI-Bus

The HC12compact uses the SPI (Serial Peripheral Interface) of the HC12 to communicate with the on-board Real Time Clock, A/D-Converter and D/A-Converter. The HC12 works as a SPI master and needs one select line for each SPI slave.

To select one of the SPI slaves the HC12compact uses the port signals PH4 and PH5 of the controller and the decoder IC6B. With these two signals, the decoder generates four different conditions:

PH4	PH5	Select-Output	selected Slave
0	0	/SPICS0	RTC
1	0	/SPICS1	DAC
0	1	/SPICS2	ADC
1	1	inactive	none

PH4=PH5=H is the default state (no Slave selected).

The following listing shows examples for initializing and sending/receiving. There are no arguments for the init function, the send/receive function expects the character to send in accumulator A and gives back the received value in the same way.

```

;=====
; File: SPI12.A
; Func: HC12 SPI Routines
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 08.02.98
;=====

; SPI Slave Chip Select Decoder:
; SPICOD0 SPICOD1
;  PH4     PH5
; -----
;  0       0     SPICS0  RTC
;  1       0     SPICS1  DAC
;  0       1     SPICS2  ADC
;  1       1     none selected

initSPI      bset PORTH,$30          ; SPICOD0/1 (PH4,5) = H
             bset DDRH,$30          ; PH4,5 Output
             bset DDRS,$e0          ; /SS,SCK,MOSI Output
             movb #$03,SP0BR        ; SPI Rate = ECLK/16 (500kHz)
             movb #$54,SP0CR1       ; SPE+MSTR+/CPOL+CPHA
             movb #$08,SP0CR2       ; as default
             rts

```

```
xferSPI      staa SP0DR
_xfs1        tst  SP0SR
            bpl  _xfs1
            ldaa SP0DR
            rts
```

;=====

## Real Time Clock

---

The Real Time Clock (RTC4553 from Seiko/Epson) not only uses the SPI signals and a select line, but an additional write enable signal. PH6 serves for this purpose. The default state of PH6 is H (Read). Select the RTC with PH4=PH5=L.

The following listing shows basic I/O-functions. Some additional source codes are available on disk, e.g. for reading/setting the time and date. Please check the RTC data sheet of the manufacturer for details!

```
;=====
; File: RTC12.A
; Func: HC12 Driver Routines for RTC4553
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 08.02.98
;=====
```

```
; SPICOD0 = PH4 = 0   selects
; SPICOD1 = PH5 = 0   the RTC
; /WRRTC  = PH6
```

```
initRTC      bset PORTH,$40      ; /WRRTC (PH6) = H
            bset DDRH,$40        ; PH6 Output
            rts
```

```
getRTC       movb #$5d,SP0CR1    ; SPE+MSTR+CPOL+CPHA+LSBF
            bset PORTH,$40      ; /WRRTC = H
            bclr PORTH,$30      ; SPICOD0/1 = L (enable SPICSO)
            bsr  xferSPI        ; send Register Address
            bsr  xferSPI        ; receive Data from RTC
            bset PORTH,$30      ; SPICOD0/1 = H (disable SPI slaves)
            rts
```

```
putRTC       movb #$5d,SP0CR1    ; SPE+MSTR+CPOL+CPHA+LSBF
            bclr PORTH,$40      ; /WRRTC = L
            bclr PORTH,$30      ; SPICOD0/1 = L (enable SPICSO)
            psha
            bsr  xferSPI        ; send Register Address + Data
            pula
            bset PORTH,$30      ; SPICOD0/1 = H (disable SPI slaves)
            rts
```

;=====

The RTC has a build-in alarm timer, which can cause an interrupt. This is useful for waking up the processor after an idle period ("sleep mode"). Close Jumper JP6 in order to use this feature.

The contents of the RTC is buffered by a lithium battery (E1). The type used is specified for a life time of 5+ years.

## A/D-Converter

The Analog-to-Digital Converter TLC2543 (Texas Instruments) has 11 input channels and a resolution of 12 bits.

To select the ADC set PH4=L and PH5=H.

The listing below shows how to initiate a conversion. Accumulator A must contain the channel number before entering the getADC function. It returns the result in double accu D with 12 significant bits.

```

;=====
; File: ADC12.A
; Func: HC12 Driver Routines for TLC2543 11ch 12bit ADC
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 10.02.98
;=====

; SPICOD0 = PH4 = 0   selects
; SPICOD1 = PH5 = 1   the ADC

initADC      ; nothing to do!
             rts

; Func: Get ADC result
; Args: A=Channel ($00..$0A)
;       Special channels:
;       $0B: VREF/2 ($800)
;       $0C: VREF- ($000)
;       $0D: VREF+ ($FFF)
;       $0E: enter power save mode
; Retn: D=Value (12 Bit significant)
;
getADC       movb #$50,SP0CR1   ; SPE+MSTR+/CPOL+/CPHA
             bclr PORTH,$10    ; SPICOD0 = L (enable SPICS2)
             lsla
             lsla              ; move channel no. to
             lsla              ; upper 4 bits of control word
             lsla
             oraa #$0c         ; lower nibble: DL=16Bits,MSB 1st,Unipolar
             psha              ; save control word
             jsr xferSPI       ; send control word
             jsr xferSPI       ; send bits 8..15 (ignored by ADC)
             bset PORTH,$30    ; SPICOD0/1 = H (disable SPI slaves)
             ;
             ldaa #100         ; wait for conversion result
             deca              ; 4 cyc x 125 ns x 100 = 50µs (>10µs)
_gadcwt     bne _gadcwt
             ;
             bclr PORTH,$10    ; SPICOD0 = L (enable SPICS2)
             pula              ; restore control word
             jsr xferSPI       ; get data (high)
             tab
             jsr xferSPI       ; get data (low)
             exg a,b           ; order h/l
             lsr
             lsr
             lsr               ; move result 4 bits to the right
             lsr

```



```
lslb
lslb
lslb
lslb
ldaa 2,x
anda #$0f
aba
jsr xferSPI           ; send DACA lowest 4 Bit, DACB highest 4 Bit
ldaa 3,x
jsr xferSPI           ; send DACB lowest 8 Bit
bset PORTH,$30        ; SPICOD0/1 = H (disable SPI slaves)
puld                  ; releasing /CS will latch new DAC output!
rts
```

=====

The DAC contains a voltage reference which results in an output level between 0 mV and 4095 mV. BR4 closes the reference output and the reference input of both channels. The lower reference voltage is tied to ground via BR3. Other voltage values may be applied after disconnecting these solder pads.

## 9. Peripherals on the System Bus

---

The HC812A4 provides several Chip Select signals. The signals /CS0../CS3 (Port F) are used on the HC12compact to control peripheral devices on the system bus:

Signal	Address	Function
/CS0	\$0200	CAN-Controller (Read-/Write-Access to the selected Register/Address)
/CS1	\$0300	CAN-Controller (select Register/Address)
/CS2	\$0380	LC-Display (alphanum.) control or general purpose select signal (associated with R/W and E-Clock)
/CS3	\$0280	general purpose Chip Select

### *Overview Chip Select Signals*

Normally the initialization code of the microcontroller application should stretch the chip select signals (up to 3 E-Clocks). In this way the MCU can communicate also with slow peripheral devices. The maximum cycle time is 1 $\mu$ s at a 16 MHz xtal clock.

## CAN-Controller

---

The CAN (Controller Area Network) is a multimaster network protocol, which originally started in the the automotive industry. Now it has become an industry standard, supported by a large number of manufacturers.

Motorola supports the CAN standard too, e.g. with the On-Chip CAN module of the HC912BC32. Since the HC812A4 does *not* have a CAN module On-Chip, the HC12compact provides an external CAN controller chip.

The device used is a SJA1000 by Philips (compatible to the former PCA82C200). It contains a complete implementation of the lower level layers of the CAN protocol. This means, the user need not to deal with



low level implementaion issues and can concentrate on the (higher level) application layers.

The whole CAN description is still rather complex. Please refer to the CAN controller documentation available on the web for details!

The following listing shows the low level accesss functions used with the SJA1000 CAN device:

```

;=====
; File: CAN12.A
; Func: HC12 Driver Routines for SJA1000 CAN2.0B Controller
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.0
; Date: 14.02.98
;=====

;-----
; Func: Target dependend hardware access functions
; Rem.: Modify this according to the actual bus interface!
;-----
CAN_ADDR    equ $0301    ; write to this address to select a register
CAN_DATA    equ $0383    ; read/write data from/to selected register
;-----

; Func: Read CAN Controller Register
; Args: X = Register Number ($00..$1f)
; Retn: A = Data
;
readSJA1000    stx    CAN_ADDR-1
               ldaa   CAN_DATA
               rts

; Func: Write CAN Controller Register
; Args: X = Register Number ($00..$1f)
;       A = Data
;
writeSJA1000   stx    CAN_ADDR-1
               staa  CAN_DATA
               rts

;=====

```

The Philips CAN controller does not have an HC12 bus interface. Therefore the decoders IC5B and IC6A are used to link the Philips part with the Motorola bus.

Each access is made in two steps: First the desired control register of the SJA1000 is selected by writing the register address (for a list of register addresses see the SJA1000 data sheet) to CAN\_ADDR. Then the content of this control register can be read or written by accessing CAN\_ADDR.

The CAN controller is able to generate an interrupt via its /INT output. To make use of this feature close solder pad BR6. Then you may detect the CAN interrupt condition via PT0 of the HC12.

Note: You will need a media interface device between the CAN controller and the two-wire CAN bus. The PCA82C250 (also from Philips) is a typical device for this purpose. We also offer ready-to-use interface modules based on this chip.

## LCD-Interface

---

There are -zillions of alphanumeric LC-Diplays (made by Philips, Seiko, Toshiba, Batron, Picvue, Optrex, Sharp...). Most of them use a Hitachi HD44780 (or compatible) controller and a standard 14-pin connector with the following scheme:

LCD Pin	LCD Signal	Function	HC12compact Signal	ST5 Pin
1	VSS	Ground	GND	19/20
2	VDD	+5V	VCC	17/18
3	VEE	Contrast Voltage	./.	./.
4	RS	Register Select	A1	32
5	R/W	Read/Write	R/W	21
6	E	Enable	ENA_LCD	38
7	D0	Data Bit 0	DB0	16
8	D1	Data Bit 1	DB1	15
9	D2	Data Bit 2	DB2	14
10	D3	Data Bit 3	DB3	13
11	D4	Datenbit 4	DB4	12
12	D5	Data Bit 5	DB5	11
13	D6	Data Bit 6	DB6	10
14	D7	Data Bit 7	DB7	9

The table also contains pins on connector ST5 which are used to connect such a LCD device to the HC12compact.

---

The following listing shows a part of the LCD driver routines (see disk for the complete source). The routines need a two byte buffer PosXY which the user program has to provide.

The header must contain a definition of the display size (number of columns and lines, see example). 2x16, 2x20, 2x40 and 4x20 are common types.

```

;-----
; File: ALCD12.A
; Func: HC12 Driver Routines for alphanumeric LC-Displays
;       (HD44780 based, from 1x8 up to 2x40/4x20 characters)
; Copr: (C)1998 by Oliver Thamm
; Vers: 1.2
; Date: 07.02.98
;-----
;-- defines -----
;-----
PosX          equ PosXY+0    ; RAM location PosXY must be
PosY          equ PosXY+1    ; defined in the main program!

;-----
; Func: LCD definitions
; Rem.: Modify this according to LCD type used!
;-----

MAX_X         equ 40         ; number of characters per line (8..40)
MAX_Y         equ 2         ; number of lines (1,2 or 4)

RowAddrTable dc.b $00       ; DDRAM address for 1st line
              dc.b $40       ; DDRAM address for 2nd line
              dc.b $10       ; DDRAM address for 3rd line
              dc.b $50       ; DDRAM address for 4th line

;-----
; Func: Target dependend hardware access functions
; Rem.: Modify this according to the actual bus interface!
;-----

LCD_CMD       equ $0381
LCD_DATA      equ $0383

;-- write a command byte -----
_writeCmd     brset LCD_CMD,$80,_writeCmd
              nop
              staa LCD_CMD
              rts

;-- write a data byte -----
_writeData    brset LCD_CMD,$80,_writeData
              nop
              staa LCD_DATA
              rts

;-----
; Func: Initialize LCD
; Args: -
; Retn: -
;-----
initLCD       psha
              pshy
              ldy #_iniTableStart
_initCmdLoop  ldaa 0,y
              bsr _writeCmd
              iny
              cpy #_iniTableEnd

```

```
        bne _iniCmdLoop
        clr PosX
        clr PosY
        puly
        pula
        rts

_iniTableStart dc.b $38      ; Function Set (8Bit,5x7)
               dc.b $38      ; dito
               dc.b $0C      ; Display on, Cursor off, Blink off
               dc.b $01      ; Clear display
               dc.b $06      ; Entry Set: Increment, No shift
               dc.b $80      ; Reset Address counter

_iniTableEnd
;=====
```

## 10. Power Management

---

Nearly all devices on the HC12compact have some kind of a power-down or suspend mode. Also parts of the controller itself can be powered up and down separately. This makes the HC12compact suitable for low-current/battery-driven applications.

Some examples:

- turn off internal modules of the HC12, e.g. ADC or Timer
- turn off the external bus interface (providing the programs runs in an internal memory location)
- Optional: use of the PLL mode clock generation
- Power Down modes of ADC, DAC and CAN-Controller

However, the on-board RAM can not be buffered by a backup battery. Instead, please reduce overall power consumption of the whole module, as described above.

# 11. Application Hints

---

In this section some special programming issues of the HC12 - and the HC12compact in particular - will be discussed. Please note, that this hardware manual can not cover all topics and techniques necessary to program the HC12 in assembly or a high level language. Please refer to the technical data books, reference and user manuals of the HC12 and the programming language you intend to use.

## About Operating Modes

---

The operating mode of the HC12 MCU is determined by the three signals MODA, MODB and BKGD at the time of reset. The following table shows all eight combinations.

MODA	MODB	BKGD	Startbetriebsart	Erläuterung
0	0	0	Special Single Chip	Background Debugging sofort aktiv, kein externes Businterface
1	0	0	Special Expanded Narrow	Wie Normal Exp. Narrow Mode, einige Schutzmaßnahmen sind jedoch deaktiviert
0	1	0	Special Peripheral	Testbetriebsart - CPU disabled
1	1	0	Special Expanded Wide	Wie Normal Exp. Wide Mode, einige Schutzmaßnahmen sind jedoch deaktiviert
0	0	1	Normal Single Chip	kein externes Businterface, interner EEPROM als Programmspeicher gemappt
1	0	1	Normal Expanded Narrow	Externer 8 Bit Datenbus aktiv, langsamer als 16 Bit Zugriff
0	1	1	reserviert	
1	1	1	Normal Expanded Wide	Externer 16 Bit Datenbus aktiv, volle Geschwindigkeit möglich

### *Operating Modes and Mode Select Pins*

Just two of them are important while working with the HC12compact: Normal Single Chip Mode and Normal Expanded Wide Mode.

During application development the Normal Single Chip Mode is the preferred operating mode. In this mode the 812A4 moves the internal EEPROM block to \$F000-\$FFFF. Doing so, the monitor program, which was programmed into EEPROM during production of the HC12compact, becomes active.

As the name says, in Single Chip Mode the external bus interface is disabled. That is why the monitor immediately switches to Normal Expanded Mode by setting the MODE control register and initializes the bus interface.

It is recommended to use Normal Single Chip Mode settings, even if you want to connect a Background Debug Interface. The BDM hardware can start the target MCU in Special Single Chip Mode, since it can override the high level on the BKGD pin. In this mode BDM is active out of reset and the CPU does not try to start a user program. This makes sense especially at the beginning of the debugging when there is not any user code.

A BDM pod is useful, but not mandatory when working with the HC12compact, since the monitor program handles download into Flash or RAM and helps with several debugging functions.

If debugging is finished, the monitor is not required anymore. To start the application, which resides in the external Flash memory, just set the jumpers to Normal Expanded Mode and reset the board.

## **Initialization Example**

---

In Normal Expanded Mode the data bus is 16 bits wide and the chip select line for the program memory (/CSP0) is active from reset. The user program resides in Flash memory and the internal EEPROM is located at \$1000-\$1FFF.

As soon as the reset line goes high, the HC12 fetches the Reset Vector from \$FFFE,FF and starts execution at the address this vector points to.

Every program starts with a number of initializations. The sequence shown in the listing below is an example for the HC12compact starting in Expanded Wide Mode:

```
main    lds    #StackPtr          ; Init Stack Pointer
        ; === REM: MCU starts in Normal Expanded Wide Mode ===
        clr   COPCTL             ; Disable Watchdog
        movb #$0c,PEAR          ; LSTRE+RDWE, ECLK
        movb #$3f,CSCTL0        ; Enable CSP0+CSD+CS3+CS2+CS1+CS0
        movb #$10,CSCTL1        ; CSD covers $0000-$7fff
        movb #$30,CSSTR0        ; CSP0+CSD not stretched (CSP1:3x)
        movb #$ff,CSSTR1        ; CS0..3 stretched (3x)
        movb #$fe,PPAGE         ; Select Program Page $FE
        movb #$0f,MXAR          ; A16E...A19E
        movb #$c0,WINDEF        ; DWEN+PWEN
        movb #$80,DDRE          ; PE7=Out (LED driver)
        movb #$00,PORTE         ; LED on
```

The sequence starts with the setup of the stack pointer. Users "love" to forget this, so double check this line if your program behaves "strange".

Another important issue is the watchdog (COP). If don't use it in your application - switch it off! If you forget this, you will encounter a reset about every second.

Initialization of the Port E Assignment Register (PEAR) follows. The configuration bits NECLK, LSTRE and RDWE have influence on some important bus signals. NECLK must be cleared to zero in order to output the E-Clock on PE4. The /LSTRB signal is used in conjunction with address line A0 to access the two 8 bits wide RAM devices, therefore LSTRE must be set to one. The Read/Write signal is also necessary, so RDWE must be set too.

The following code lines establish the appropriate chip select settings. It is recommended to use this example setup in a user application too. For details refer to the HC812A4 Technical Summary.

The PPAGE register has a reset value of \$00. Before enabling the paging, it makes sense to re-initialize the PPGAE register with \$FE. By doing this, the same memory page is visible in the \$8000-\$BFFF area as in the linear address mode immediately after reset.

In the following line the Memory Expansion Assignment Register (MXAR) is used to determine, how many lines of Port G are used as additional address lines. The HC12compact uses A16 to A19, so two lines (PG4 and PG5) remain general purpose I/Os.



By setting two bits in the WINDEF register, the paging for both program and data memory is enabled.

Finally, PE7, which drives a LED via a buffer, is set to output and the LED is switched on. This is the end of the example initialization sequence.

## **Schematic Diagram**

---

To ensure best visibility of all details, the schematic diagram of the HC12compact is provided as a separate document (2 pages).

## **Additional Information on the Web**

---

Additional information about the HC12compact Controller Module will be published on our Website, as it becomes available:

<http://elmicro.com/en/hc12compact.html>

## 12. Monitor Program TwinPEEKs

---

Software Version 1.3d

### Introduction

---

The monitor program TwinPEEKs is useful to load and execute user programs and to view and modify memory locations.

TwinPEEKs resides in the internal EEPROM of the HC12. The whole 4 KB EEPROM area plus a region of about 512 bytes of RAM are reserved for TwinPEEKs. The user program will be loaded into an external (RAM or Flash) memory space. For details see "Memory Map" below.

The HC12compact must be jumpered for **Normal Single Chip Mode** in order to start TwinPEEKs out of EEPROM (see section "Jumpers"). During initialization the monitor program automatically switches to **Normal Expanded Wide Mode**.

TwinPEEKs on the HC12compact communicates with a host PC via the first serial interface (SCI0). Communication parameters are: 19200 Baud, 8N1, no protocol. These parameters are based on the standard 16 MHz crystal clock of the HC12compact.

When the monitor program shows its prompt (containing the actual program page setting from the PPAGE register), it is ready to receive your commands.

### Notation

---

All numbers are in hex format (w/o any additional characters). The input is not case-sensitive.

A monitor command consists of a single character, followed by a list of arguments. The argument list may contain up to six items, separated by a space or comma.

The address space is 64 KB, so addresses have a maximum length of 4 digits. This address space of the HC12 contains all ports and control registers.

**Note:**

**An end address in TwinPEEKs is the address *after* the last included address. E.g. the command "D 1000 1200" displays the memory ranging from \$1000 up to (including) \$11FF.**

An editable line buffer receives the RS232 user input. Valid ASCII codes are from \$20 to \$7E. Backspace (\$08) deletes the character left of the cursor. Enter (\$0A) finishes the input.

---

## Write Access

---

**Flash Memory**

TwinPEEKs is able to write Flash memory bytes. Please note, that the memory cell will not be deleted automatically! The X command should be used to delete a Flash memory block before writing to it. It is not possible to delete **single** bytes or words (see Flash memory data sheet for details).

The Flash memory device is organized in words (double bytes), though TwinPEEKs has no problem to write even single bytes. For this purpose the monitor reads the other byte of a Flash word, adds the desired byte and then writes the complete word information to the Flash.

There is an area from \$F000 to \$FFFF where the internal EEPROM hides the external Flash memory area. The EEPROM is protected from write accesses in order to protect the monitor program in it. So TwinPEEKs' strategy is to access the external Flash memory instead of the internal EEPROM if a write access to the area mentioned above takes place. Reads from this area are not affected, a read from \$F000 or above will access the EEPROM (which the CPU actually "sees").

With this behaviour the monitor program helps to load a complete user program - together with all interrupt and reset vectors - into Flash memory. Just load the user program, change the jumpers to Expanded Wide Mode, reset the board and the program will start in real time. After switching back to Single Chip Mode, the monitor will start again.

As mentioned above, you will not be able to display what you have loaded before into the \$F000-\$FFFF Flash memory area. A memory dump command will always show what the CPU sees (the EEPROM!). To have a look at the contents of the Flash memory between \$F000 and \$FFFF you should select program page \$FF (use "P FF") and display the "moved" memory contents of \$B000-\$BFFF.

## Ports and Control Registers

Please note, that **not** all ports and control registers may be read and written in the same way. You may get an error message when writing to a write only register, because TwinPEEKs tries to read back the contents in order to verify the write access. A typical example are interrupt flags, which are *cleared* by writing a *one-bit* to the desired bit position. Reading back the one's written will result in zero's in this case!

## Autostart Function

---

To provide an easy way to start a user program out of reset, the monitor program has an autostart function.

After reset and some basic initializations TwinPEEKs detects if the MCU signals PAD0 and PH1 are connected. In this case, the monitor jumps to location \$8000 instead of starting the normal monitor command loop.

The advantage of this autostart mode is that the users need not to worry about system initialization, because the monitor program will do this job before jumping to the start of the user code. The mode jumpers also may remain in the Single Chip Mode position. The user just has to connect pins 45 (PH1) and 47 (PAD0), which are located side-by-side on connector ST5.

## Interrupt Vectors

The interrupt vectors of the HC12 are located at the end of the 64 KB memory map. On the HC12compact this is a write protected EEPROM area, if the monitor program is active. Normally, this would prevent the user from changing the interrupt and reset vectors.

There was a similar problem with the HC11 running in bootstrap mode. The HC11 used pseudo interrupt vectors in the internal RAM, and so does the TwinPEEKs monitor on the HC12compact.

The user program just has to place a jump instruction into RAM to call a specific interrupt function. Here comes an example for the SPI interrupt:

```

ldaa #06          ; JMP Opcode
staa $0BC7       ; SPI Pseudo Vector
ldd #isrFunc     ; Jump Address
std $0BC8        ; SPI Pseudo Vector + 1

```

The following listing is part of the monitor program. It shows, how (and where) the monitor redirects all interrupts:

```

FFCE : 0B B8 dc.w RAMTOP-72 ; KWUH
FFD0 : 0B BB dc.w RAMTOP-69 ; KWUJ
FFD2 : 0B BE dc.w RAMTOP-66 ; ATD
FFD4 : 0B C1 dc.w RAMTOP-63 ; SCI1
FFD6 : 0B C4 dc.w RAMTOP-60 ; SCI0
FFD8 : 0B C7 dc.w RAMTOP-57 ; SPI
FFDA : 0B CA dc.w RAMTOP-54 ; Pulse Accu Input Edge
FFDC : 0B CD dc.w RAMTOP-51 ; Pulse Accu Overflow
FFDE : 0B D0 dc.w RAMTOP-48 ; Timer Overflow
FFE0 : 0B D3 dc.w RAMTOP-45 ; TC7
FFE2 : 0B D6 dc.w RAMTOP-42 ; TC6
FFE4 : 0B D9 dc.w RAMTOP-39 ; TC5
FFE6 : 0B DC dc.w RAMTOP-36 ; TC4
FFE8 : 0B DF dc.w RAMTOP-33 ; TC3
FFEA : 0B E2 dc.w RAMTOP-30 ; TC2
FFEC : 0B E5 dc.w RAMTOP-27 ; TC1
FFEE : 0B E8 dc.w RAMTOP-24 ; TC0
FFF0 : 0B EB dc.w RAMTOP-21 ; RTI
FFF2 : 0B EE dc.w RAMTOP-18 ; IRQ / KWUD
FFF4 : 0B F1 dc.w RAMTOP-15 ; XIRQ
FFF6 : 0B F4 dc.w RAMTOP-12 ; SWI
FFF8 : 0B F7 dc.w RAMTOP-9 ; Illegal Opcode
FFFA : 0B FA dc.w RAMTOP-6 ; COP Fail
FFFC : 0B FD dc.w RAMTOP-3 ; Clock Monitor Fail
FFFE : F8 00 dc.w main ; Reset

```

## Memory Map

---

The following table contains the memory usage on the HC12compact if the TwinPEEKs monitor is active (external memory regions not shown):

<b>Start</b>	<b>End</b>	<b>Memory</b>	<b>Usage</b>
\$0800	\$0A00	int. RAM	available for user program
\$0A00	\$0Axx	int. RAM	monitor variables
\$0Axx	\$0BB8	int. RAM	monitor stack
\$0BB8	\$0C00	int. RAM	redirected interrupt vectors
\$F000	\$0000	int. EEPROM	monitor code

---

## Monitor Commands

---

### Analog/Digital Converter

---

#### Syntax: A

Displays the results of all 11 ADC channels (#0..#10) and the three internal reference channels #11 (Vref/2), #12 (Vref) and #13 (GND). The results are updated continuously. Press Enter to abort this command.

The values range from \$0000 to \$0FFF (12 bits). \$0000 equals 0V, \$0FFF equals 4.095V.

Command w/o arguments:

A	Display ADC results
---	---------------------

### Dump Memory

---

#### Syntax: D (<SADR> (<EADR>))

Displays the memory contents. Each display line contains 16 bytes in hex and in ASCII format. Non-printable ASCII characters are displayed as a dot.

Command with one, two or three arguments:

D F000 F800	Displays the memory from \$F000 to \$F7FF
D F000	Displays 4 lines (\$40 bytes) starting at \$F000
D	Displays 4 lines (\$40 bytes) starting at the current address

## Edit Memory

---

### Syntax: E (<SADR>)

Edit memory bytes.

Command w/o or with one argument:

E F000	Edit memory starting at \$F000
E	Edit memory starting at the current address

In edit mode, the monitor displays the address and its contents. Then it waits for a new value or one of the following commands:

<ENTER>	Next address
-	Previous address
=	Display current address again
.	Quit edit mode
Q	Quit edit mode

If a new value was given, the monitor writes and verifies it. Flash memory may be written as well as RAM cells. If everything is okay, the next line with the next address is output.

If a Flash write fails, check if the cell was erased before (using monitor command X).



## Fill Memory

---

**Syntax: F <SADR> <EADR> <BY>**

Fills a memory area with a specific value.

Command with three arguments:

F F000 F800 FF	Fills memory from \$F000 to \$F7FF with \$FF
----------------	--

The command will be aborted if a write error occurs.

## Goto Address

---

**Syntax: G (<SADR>)**

Executes a user program (using JSR) at the given address. The address is displayed by the monitor. User program may return to the monitor using a RTS instruction.

Command w/o or with one argument:

G F100	Call user program at \$F100
G	Call user program at the current address

In order to successfully return to the monitor program, the user program must not change the stack position, the serial interface and interrupt masks.

## Help

---

**Syntax: H**

Displays a short help message. Command w/o arguments:

H	Displays a list of available commands
---	---------------------------------------

## Flash-ID

---

### Syntax: I

Displays the device ID of the Flash memory device. Should be 2223 for the 29F400BT (default type used). see AMD data sheet for details)

Command w/o arguments:

I	Info on Flash device ID
---	-------------------------

## Load S-Records

---

### Syntax: L (<OFFS>)

Loads S-Record (code and/or data) into memory. The S-Records already contain the address information needed.

The destination addresses may be relocated by providing an offset argument to this command:

L	Loads S-Records into memory
L E000	Loads S-Records into memory and adds \$E000 to all destination addresses. (example: data in the range from \$1000 bis \$10FF will be loaded at \$F000 to \$F0FF)

To use this command, your terminal software on the PC must be able to send the S-Record file after the message "Loading..." is displayed by TwinPEEKs. This function normally is known as "Text-" or "ASCII-Upload".

The S-Record lines are not echoed by the monitor. The command will be cancelled if a write error occurs. The user may stop the load function by pressing Escape.

The monitor can process S0, S1 and S9 types of S-Record lines. For details on S-Records see the motorola documentation.

---

**Note:**

Writing to Flash memory takes a certain execution time which is longer than for RAM. Therefore S-Record data must not be sent faster than the monitor program can write it to Flash.

To synchronize both data receive and write, there is a simple protocol which uses the monitor: After a S-Record line is finished (processed) the monitor sends a \* (asterisk). This is a signal to the terminal program on the PC side, that another S-Record line may be sent.

Even simple terminal programs (such as TERMINAL.EXE from Windows 3.1) can handle this protocol.

Another way is to send the S-Records with full speed and store them in RAM. The offset argument of the load command may be used to modify the destination address. When the download is finished, the code can be moved using the move command to the final destination in Flash memory.

## Move Memory

---

**Syntax: M <SADR> <EADR> <ADR2>**

Copies the memory area from <SADR> to <EADR> to another area starting at <ADR2>. The size <LEN> of the memory block to move is:

$$\langle \text{LEN} \rangle = \langle \text{EADR} \rangle - \langle \text{AADR} \rangle$$

Command with three arguments:

M 1000 1800 F000	Copies the memory block \$1000 - \$17FF to \$F000 - \$F7FF
------------------	--

The monitor detects whether original and destination block overlap. In the case that <ADR2> is lower than <SADR> the monitor starts copying from the bottom of the block, otherwise from top.

If a write error occurs, the command is aborted.

## Page Select

---

### Syntax: P (<PAGE>)

Selects a memory page using the PPAGE register. The selected page is then visible in the 16 KB area from \$8000.

The 29F400 Flash memory has 512 KB or 32 pages. To select the last page use page number \$1F. Since the upper two bits are not significant, the same page is selected with \$FF.

If the command is used w/o argument, the current program page number is displayed.

P	Displays the currently selected program page number
P 12	Selects PPAGE=\$12

## Real Time Clock

---

### Syntax: T (<HH> <MM> <SS> <DD> <MM> <YY>)

The command displays the RTC time when used w/o argument. Of course, the optional RTC chip must be present.

To set the Real Time Clock, add the following arguments to the command: hour, minute, second, day, month, year. Each argument is two **decimal** digits long (BCD format).

Remark: The RTC will always set the seconds to 0.

T	Displays time and date
T 14 30 00 18 10 98	Sets the RTC to 14:30, October 18 98

---

## Digital/Analog Converter

---

### Syntax: V <CH0> <CH2>

The optional DAC has two channels which may be adjusted with this command. The values range from \$0000 to \$0FFF (12 bits). \$0000 equates 0V, \$0FFF equates 4.095V.

V 800 FFF	Outputs 2.048V on DAC channel 0 and 4.095V on DAC channel 1
-----------	---

---

## Sektor/Bulk Erase Flash

---

### Syntax: X (<SECTOR>)

Deletes a single Flash memory sector. The argument denotes a sector address as described in the 29Fx00 Flash memory data sheet. The sector address represents the address lines A12-A17 of the Flash memory.

If no argument is given, the command erases the whole Flash chip (Bulk Erase).

The user has to confirm the erase process by typing "Y".

The following table shows the sector addresses of the Am29F400T and the corresponding program page numbers:

<b>Sector Address</b>	<b>Sector Size</b>	<b>Memory Area</b>	<b>PPAGE</b>
00	64KB	00000–0FFFF	00-03
08	64KB	10000–1FFFF	04-07
10	64KB	20000–2FFFF	08-0B
18	64KB	30000–3FFFF	0C-0F
20	64KB	40000–4FFFF	10-13
28	64KB	50000–5FFFF	14-17
30	64KB	60000–6FFFF	18-1B
38	32KB	70000–77FFF	1C-1D
3C	8KB	78000-79FFF	1E (lower half)
3D	8KB	7A000-7BFFF	1E (upper half)
3E	16KB	7C000-7FFFF	1F