F²MC-16L Series
MB90675 Series Evaluation Board

USER MANUAL

Printed: 12. August 1996

The information in this document has been carefully checked and is believed to be entirely reliable. However, Fujitsu Mikroelektronik GmbH assume no responsibility for inaccuracies.

The information contained in this document does not convey any license under the copyright, patent rights or trademarks claimed and owned by Fujitsu.

Fujitsu Mikroelektronik GmbH reserve the right to change products or specifications without notice. No part of this publication may be copied or reproduced in any form or by any means, or transferred to any third party without the prior written consent of Fujitsu.

Disclaimer

Use the Evaluation Board at your own risk. Read the document, follow instructions, and make backups. Fujitsu can not guarantee that the board and code will work with every possible combination of hardware and software. While we have done our best to produce well-written, bug-free code, oversights and omissions can occur.

Fujitsu is not responsible for any damages that might occur while using the Evaluation Board or belonging software. We can not guarantee it will be "safe" or "harmless" in all possible applications, and any loss of time, data, hardware or software you incur as a result of using the Evaluation Board is your responsibility.

LIMITED WARRANTY AND REMEDIES

THE DEVICE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND , EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL FUJITSU BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OF THE DEVICE OR PROGRAMS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LICENSE, UNDERSTAND IT AN AGREE TO BE BOUND BY ITS TERMS AS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US, SUPERSEDING ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS LICENSE.

Acknowledgements

We mention other people's products in this document, and would like to give credit where credit is due.

MS-DOS is a trademark of Microsoft Corporation

I²C is a patent owned by Philips

1. Introduction

The MB90675 Evaluation Board and the associated software makes it easy to evaluate and demonstrate almost all features of Fujitsu's F²MC-16L microcontroller series.

Along with the Fujitsu F²MC-16L software tools, it can also be used as a low cost development system for user software design.

Monitor software and some on-board peripherals (used to realise a RS232 interface to a PC terminal program) support program downloads, sophisticated debug functions such as breakpoint settings, memory dumps, symbol handling and single step execution.

The monitor and the user's test program have to share a special memory area to enable both the monitor and the user's test program to be started by a Reset. Therefore, a part of memory area provided for the test program is mapped to the shared area. Thus, the user can develop and test program code within the same address range as the final user program which would reside in a PROM or mask ROM.

The user memory mapping can be activated by a separate reset button, allowing restart of the user program, beginning from a real external hardware reset.

This user manual is divided into 6 chapters.

Chapter 2, gives hints on the installation and basic information on the board hardware.

Chapter 3 contains a detailed description on all monitor commands. To get started quickly, this chapter may be skipped and serve just as a reference.

Chapter 4 specifies a monitor software interface (monitor BIOS) that can be accessed by user programs for terminal input/output functions, may be skipped by novice users.

Chapter 5 contains various examples and shows how to get started.

Chapter 6 contains further details regarding the board hardware, connector assignments and a functional description to enable external hardware extension.

Finally, the Appendix contains a board schematic, example program listings and notes on the monitor software implementation.

2. EVAKIT Hardware

2.1. Installation

The installation of the evaluation board is straightforward.

A power supply, capable of supporting a fairly stable 7-8V DC and about 450 mA, is required. Note, that the power connector must be '+' at the shield and '–' in the centre.

As a first step, make sure that the small switches located nearby the LEDs are in marked position ON. Connect the power supply and check that the red Power LED (see fig. 1) remains on and that the green RUNning-User LED and the red LED Displays light up briefly after Monitor Reset has been pressed and released.

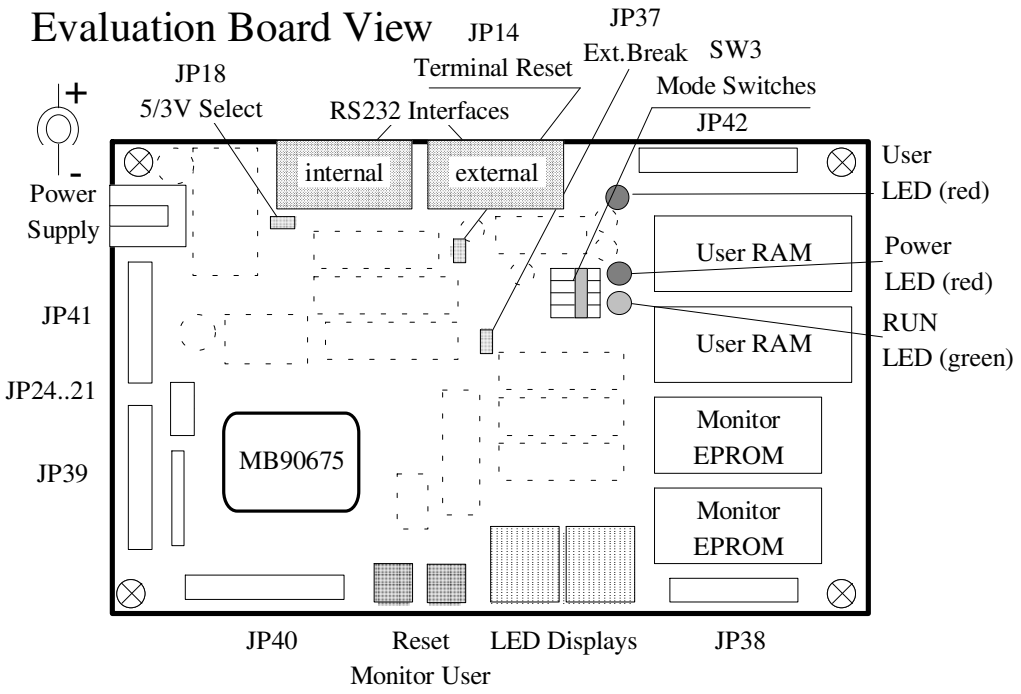


Fig. 1: Evaluation board view

While pressing the User Reset button, the green RUNning-User LED should be on. After releasing the User Reset button the LED Displays should light up again.

If these installation checks were successful, the power supply should then be unplugged and the serial connection to a PC via the 9-pin RS232 cable established at the external port (labelled at solder side of the board) before re-connection of the power supply.

Now, the LEDs should behave in the same manner. – If not, it is most likely that the DTR signal of the serial connection, which is used as an external Monitor Reset control line is activated for some reason.

If a terminal program provided by Fujitsu (the **ProMan©** Vers. 3.5 terminal window or the stand alone mini terminal program "mt.com") is used, the problem should disappear as soon as the software is started.

(If "mt.com" is used, ensure that the COM1 port is initialised by the "mode com1: 9600,n,8,2" command first.)

If a different terminal program is used which does not support this option (e.g. the TERMINAL program of Windows 3.1), this effect can be disabled by removing the "Terminal Reset" jumper JP 14 (see fig. 1, p 7).For jumper information and setting see page 14.

On the terminal, the following message from the evaluation board monitor should appear:

```
==============================================
==   _____                 _                 ==
==  |       |     |  /   \    Evaluation     ==
==  |-----  \   /  |_____|   Board           ==
==  |_____   \_/   |     |   Monitor V16.5   ==
==                                            ==
==  MB90675-Board  Version 1.4                ==
==  MB90675-controller found                  ==
==  (c) Fujitsu Microelectronics, 1996        ==
==============================================

(C) <ModeByte,ResetVector : NONE>
```

The ´(C)´ status message indicates that the monitor executed a "cold" reset after power up.

On following "warm" resets, an abridged headline and "(W)" message will appear.

```
==============================================
==  Evaluation Board V1.4 / Monitor V16.5   ==
==  MB90675-controller found                ==
==  (c) Fujitsu Microelectronics, 1996      ==
==============================================

(W) <ModeByte,ResetVector : 88,FC0046>
```

If a program has been downloaded, the status message will contain an address indicating the mode byte which should be 88h and the reset vector of the program downloaded.

2.2. Technical Characteristics

Power Supply	7V ... 8V, 250 ... 450 mA (normal operation), 5V/3V output selectable, drives up to 1 A for additional devices
RAM Area	256 KB (including 10 KB used by monitor)
ROM Area	128 KB Monitor ROM
Emulated Controllers	MB90675-Series
Serial Interface	2 x RS232 compatible (1 x controller internal and 1 x external controller independent) **Parameters :** 9600 Baud, No Parity, 8 Data-Bit, 2 Stop-Bit (300...38400 Baud selectable for the external port)
Debug Functions	All functions are realised in software including Breakpoints, Single-Step, Watch-Points A "keyboard break" function occupies one hardware interrupt, if enabled.
Board Restrictions	**Software:** INT9 can not be used, because it is needed for the breakpoint system implementation. **Hardware:** Ports 0, 1, 2.0-2.3, 3.0-3.3 and 3.7 are used for the external bus system and are not available as general I/O-Ports.
MB90675 Controller Specific Features	- UART (2 channels) - A/D converter (8 x 10/8 Bit) - 2 x 8-bit PPG timer - 2 x 16-bit reload timer - ICU (4 x Input Capture Unit) - I²C interface* - OCU (8 x Output Compare Unit)

Table 1: Technical Parameters

CE Conformity

The STARTERKIT_16 is conform to the 89/336/EEC - electromagnetic compatibility EEC law. The following generic test procedures were applied:

- EN 50081 Part 2
- EN 50082 Part 2

It was tested in a typical setup using the MB2141A from Fujitsu Mikroelektronik GmbH. Thus, it is suitable for industrial environment.

2.3. Evaluation Board Resources

The Evaluation Board was designed in order to leave as many microcontroller resources available for user evaluation and applications as possible.

All of the I/O ports (port 2.4 - 2.7, 3.4 - 3.6, 4 - 9, A, B) are available for external use via the on-board connectors. The other ports are available too, but already taken by the external bus and control signals. These can be used to expand the system (e.g. additional memory).

* I²C is a PHILIPS patent

The board has various modes to use the on-board RAM in different configurations. On User Reset the memory mapping is changed to User Mode. So, the user RAM that holds the application program can be executed in that case. On Monitor Reset the board is initialised to its defaults, after that it switched to User Mode mapping.

Only a small RAM area (90000h..927FFh) is reserved for monitor variables and data. A few memory mapped control registers (see memory map) are located in a so-called control bank (bank 7). Three of the four control registers linked to LEDs can be used for simple user program indicators.

The lower memory area allows access to the controller built-in RAM, general register banks, EI²OS descriptors (Extended Intelligent I/O Service) and the controller peripherals. Within the I/O area 64 Bytes of external memory are accessible via special I/O instructions.

The board provides also control signals for memory areas not taken by any device of the board.

NOTE: Although the MB90670/675 controllers have a 24 bit address size only 20 bits are used for accessing external memory. Therefore, only 16 physical pages, each 64 KB of size, are available. Address bits 16 ...19 select one of these pages. Hence, it follows that address FF1234h and address 0F1234h accesses the same cell in page F.

Only page 0 is exceptional. If the upper four address bits (20 ... 23) are all zero the absolute page 0' is selected. This page contains the controller peripherals, internal RAM and parts of external RAM of external page 0. Any other value than zero of the highest four address bits selects the external page 0. The upper bits (20...23) have no function referring to the mapping if any other page than page 0 is addressed.

However, the controller provides an "Auto-Cycle" function to prolong external access time (see MB90675 User Manual). Referring to this function the upper address bits (20...23) select three areas (0...7, 8...F and absolute page 0') with separate programmable "Auto-Cycle" number.

2.4. Evaluation Board Controls

Memory mapping is introduced in this concept as a means of allowing the user to run his own application as it would be in the final target system, in the final memory area. This is the so-called User 0 Mode, which is the default operating mode for all users. All other modes (User 1a, User 1b) can be used by advanced programmers in order to start application software from one's extension EPROM for example.

From now on the terms "bank" and "page" are used as follows: "Bank" refers to a 64 KB area of an specific external component. E.g. the two RAM chips contains four RAM banks each 64 KB of size. "Page" refers to a specific 64 KB memory area indicated by the bits 16 ... 19 of an address. E.g. xAxxxxh accesses page A. That means, every bank is existing only once, but is accessible in different pages. However, the CPU bank register always refer to the bits 16 ... 19 of an address.

The monitor software resides in two fixed ROM banks on memory pages C and D (see fig. 2). This area (128 KB) is not available for the user. However, there are 256 KB RAM (pages 8..B) provided for user program and data.

At the moment of a Monitor Reset (or due to register settings) the monitor ROM is mapped to the upper area (memory pages E, F) to enable the controller to read the reset vector. After User Reset these upper pages are taken by mapped RAM or so-called "Module" banks.

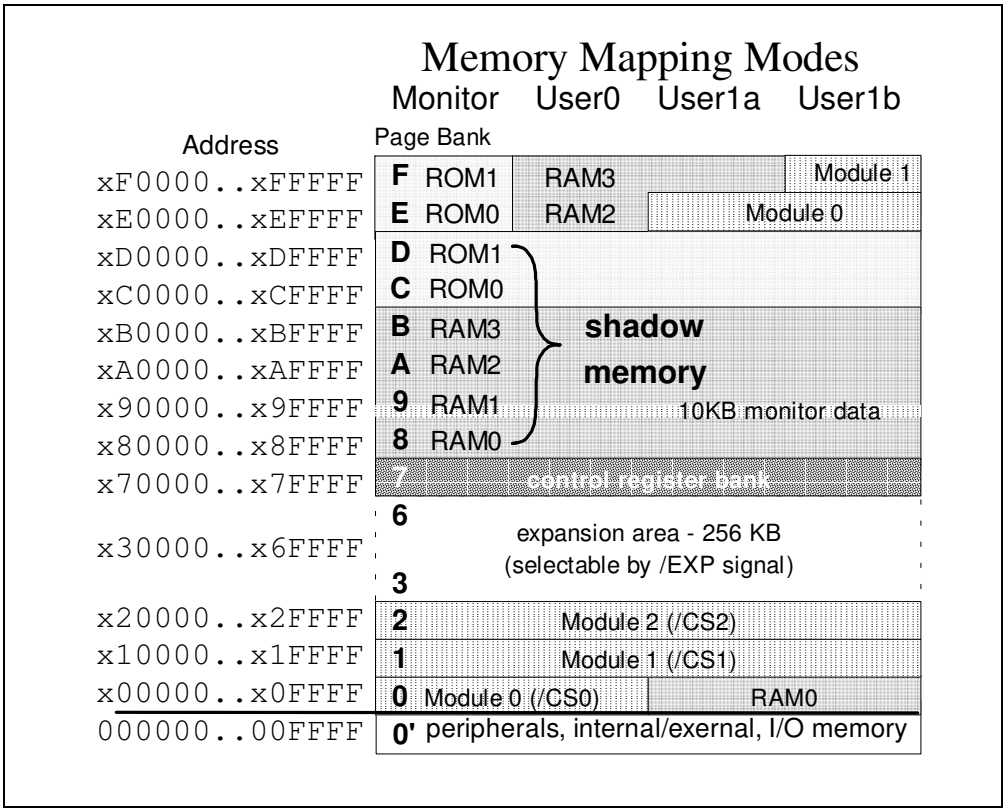


Fig. 2: Memory mapping of the Evaluation Board

Three different 64 KB blocks (Module 0..2) are selectable by additional chip select signals. Module 0 is selected by $\overline{CS0}$, $\overline{CS1}$ selects Module 1 and $\overline{CS2}$ selects module 2. Module 1 and 2 are always located on page 1 respectively page 2. In Monitor Mode and User Mode 0 the Module 0 is mapped to page 0.

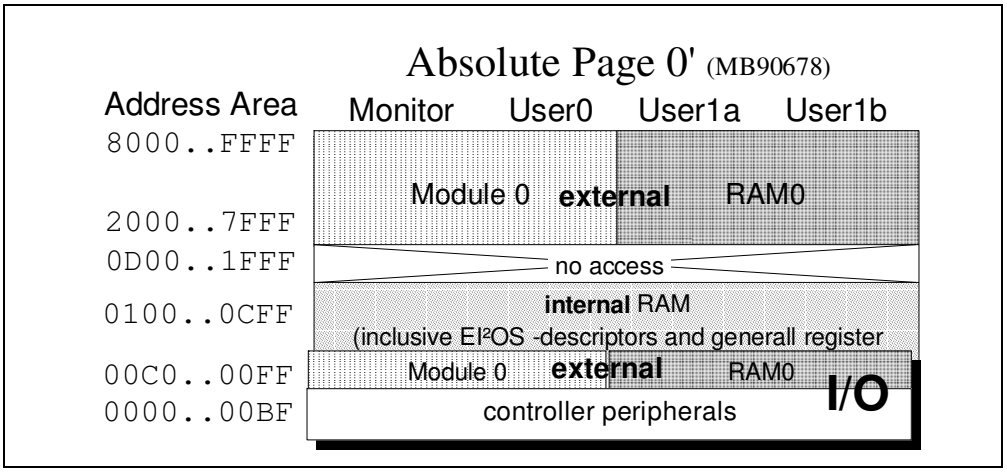


Fig. 3: Memory area 000000..00FFFFh. This page is accessed when address bits 16..23 (20..23 not output by the controller) are all zero. The addresses 00xxxxh access this page. However, 10xxxxh accesses the complete external page 0.

In User Mode 1 (sub-mode "a" and "b") RAM 0 is mapped to page 0, so RAM 0 can be accessed as well on external page 0 as within the external I/O area and upper area on absolute page 0. In return for it Module 0 is mapped on page E. In sub-mode "b" this area is enlarged to 128 KB so that Module 1 could be accessed on page F.

The RAM on page 0 (in User Mode 1) might be used for accessing the external I/O-area. A program fixed on ROM (including its own interrupt vector table and plugged as Module 1) could be started in User Mode 1b by pressing User Reset button.

The low active signal $\overline{EXP}$ (expansion) set by the Evaluation Board peripherals might be used to select the remaining external memory area not occupied by the board itself. These signals ($\overline{EXP}$, the module chip selects $\overline{CSx}$, the address bus provided by the Evaluation Board and the address/data bus, the bus control signals provided by the controller itself) might be used for system expansions.

Eight control bank register and four switches control the evaluation board and its memory mapping. Three jumpers are needed for further functionality. The control bank register resides at physical page 7 (see fig. 4). The four switches are placed left beside the Power and the User RUN LED (see fig. 1).

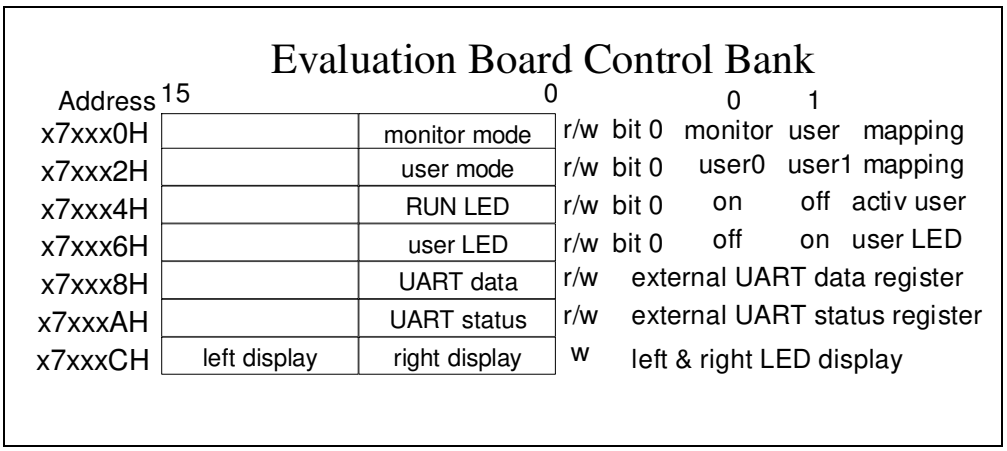


Fig. 4: The eight registers of the Evaluation Board control bank

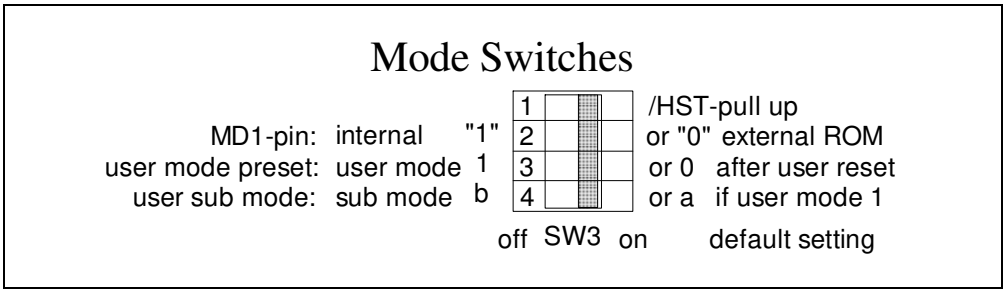


Fig. 5: on-board mode switches block

The memory mapping is controlled by the two control bank registers and two switches: Monitor Mode register, User Mode register, SW3/3 and SW3/4.

The Monitor (or Master) Mode Register (70000h) selects between Monitor Mode and User Modes. If bit 0 is set to "0" Monitor Mode mapping is active and the other mapping controls have no function at all. If bit 0 is set to "1" User Mode mapping is active. It is cleared on Monitor Reset and set on User Reset.

The User Mode Register (70002h) selects between User Mode 0 and User Mode 1 if the Monitor Mode register is set to "1" (User Mode). It is cleared on Monitor Reset. After User Reset (User Reset button) it is set equivalent to User Mode Preset (SW3/3 position). The User Mode Preset function is triggered off only by pressing the User Reset button. Software User Reset (Monitor command) sets the User Mode register only if requested by parameter. Software Reset (controller register) keeps this mapping unchanged.

The User Sub-Mode Switch (SW3/4) does only affect User Mode 1 and selects between User Mode 1a and 1b. If the Monitor Mode register and the User Mode register are set to "1" the selected sub-mode "b" (SW3/4 = off) causes the mapping of Module 1 to page F. In sub-mode "a" (SW3/4 = on) RAM 3 is mapped to page F.

The relation of the controls to the memory modes shows following table:

Memory Mode	Board Controls		
	Monitor Mode Register	User Mode Register	Sub-Mode Switch SW3/4
Monitor	0	-	-
User 0	1	0	-
User 1a	1	1	On
User 1b	1	1	Off

The Run LED Register (70004h) controls the RUN LED. It does not affect any other hardware. However, it is intended as a running-user-program indicator. The monitor clears this register (LED on, inverse logic bit 0) when it releases the controller for user program and sets the register (LED off) when it gains control back. Therefore, this register should not be used, by the User' program.

The User LED Register (70006h) controls the separated red LED and works similar as the RUN LED. This LED is intended for user's purpose. Writing "1" at bit position 0 switches the LED on and writing "0" switches it off.

The External UART Register accesses the (external) on-board UART controller. Due to this controller, the monitor is independent of the timing of the microcontroller's internal UART. Therefore, the external UART Data Register (70008h) and external UART Status Register (7000Ah) should not be accessed by the user.

The two LED Display Registers (7000Ch and 7000Dh) are two write-only register for controlling the LED Displays. The high byte controls the left LED Display and the low byte the right one. The different bits are assigned to the LED-segments as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P1	G1	F1	E1	D1	C1	B1	A1	P0	G0	F0	E0	D0	C0	B0	A0

Note, that any segment is lighted up if the appropriate bit is set to "0" (inverse logic). E.g. "A2" is coded as (7000Ch)=88A4h and "HI" is coded as (7000Ch)=89F9h.

The "Hardware Standby" pull-up switch (SW3/1) in default position connects $\overline{HST}$ to a pull-up resistor for normal operation. This switch can normally be kept in this position even if the $\overline{HST}$-pin of JP39 is used. As the $\overline{RST}$-signal of JP42 is only a probe signal, $\overline{HST}$ can be used for an external reset circuit. (Note, the differences between $\overline{RST}$ and $\overline{HST}$ might be found in the MB90670/675 user manual.)

The Mode Pin 1 Switch (SW3/2) sets the $MD1$-pin. SW3/2 in position "On" enables reading reset vector from external bus (default). Otherwise the vector is read from internal (on-chip) ROM.
$MD0$ is hard wired with "1" and $MD2$ is hardwired with "0". This configuration allows 16 Bit busses only . EPROM-write mode is not possible.

The Terminal Reset Jumper (JP14) connects the DTR-line of the external RS232 port to a special reset circuit. Leave it out, if this option is not requested or the terminal causes misoperation.

The Voltage Selector Jumper (JP18) reduces the whole board power supply voltage to a value suitable for most 3V and 3.3V circuits. It is intended for further modifications using 3V-devices. If disabled (default) the voltage is set to 5V.

The A/D-Reference Jumpers (JP21..24) are intended for configuring the external A/D circuit for test purpose. JP23 connects $AVr-$-pin to $AVSS$-pin and JP24 connects $AVr+$-pin to $AVCC$-pin. JP21 connects the $AVSS$ to GND. $AVr-$ is connected to GND too, if both JP21 and JP23 are enabled. JP22 connects the $AVCC$ to VCC. $AVr+$ is connected to VCC too, if both JP22 and JP24 are enabled. Disable these jumpers, if the appropriate pin of JP39 is used.

The Terminal Break Jumper (JP37) enables the external UART chip set to generate a hardware interrupt. So the user can break the program by pressing any key of the (master) terminal keyboard. This option occupies hardware interrupt #3 of the microcontroller. Remove the jumper, if this interrupt resource is needed. Removing the jumper in order to disable the keyboard break function is not necessary, because the monitor provides a special command (UBREAK).

The Monitor Reset Button activates the Monitor Reset. The Monitor Mode register and User Mode register are cleared, i.e. all user mapping modes are disabled.

The User Reset Button activates the User Reset. If the Monitor Mode register is set to 1, the User Mode register value depends on SW3/3 position (on - User Mode 0, off - User Mode 1). The reset vector is read from RAM 3 (User Mode 0 and User Mode 1 sub-mode "a") or from Module 1 (User Mode 1 sub-mode "b" - SW3/3 and SW3/4 off).

NOTE: On power-on reset the monitor writes a default vector table into the user interrupt vector area (1 KB from FFC000..FFFFFF ⇔ shadow RAM FBC000..FBFFFF).

The Forced Monitor Reset is intended for the case that the Monitor Reset button does not start the monitor. Depending on user program and what its initialisation routine sets the control signals for external bus, P37 can be used as I/O port instead as *CLK*. In order to trigger monitor mapping push the Monitor Reset button first, keep it pushed then press User Reset. Release User Reset first then Monitor Reset.

Finally, the RS232 connectors should be mentioned. The external port is intended for monitor communication. It is the default port of the monitor shell and uses an external (no controller resource) UART device. The internal port is the default user program port. So the user program can benefit from the microcontroller resources. It is possible to exchange the functionality of the ports or to set both functions the same port (e.g. there is only one terminal available). However, recommended configuration is a terminal on external UART for monitor control and a second terminal for user program I/O-operation on internal UART0.

3. Monitor Operation

3.1. General Operation

The monitor can be operated via a PC terminal, like the terminal window of **ProMan©** (Fujitsu's Programming Manager), the included mini terminal "MT.com" or any other terminal program.

The default communication parameters are 9600 baud, 8 bit, no parity, 2 stop bits. If possible, ANSI terminal emulation should be activated.

After pressing the Monitor Reset Button, only the Power LED on the evaluation board should be active and a message from the Monitor should appear on the terminal. The monitor is then ready for command input.

3.1.1. Command Input

Commands can be entered, if the monitor has posted the command prompt ">".

This is important since the RS232 interface is realised by a software polling procedure and characters will get lost if the monitor is busy while characters are typed in.
A command is input by typing the command word or a significant part of it plus appropriate parameters. The RETURN-key terminates input and executes the command.

The number of letters necessary for command identification depends on the other commands with same leading letters. For some commands a list with possible (defined) options can be displayed by typing a comma ',' as parameter (e.g. "baud ,↵").

The monitor provides the following edit functions:

- Mis-typed characters can be erased by using the BACKSPACE key.
- The whole line can be erased by using the DELETE key.
- Pressing the TAB key, will automatically re-type one of four previous commands. Pressing again re-types the next.
- A complete command line can be skipped by pressing the ESC key.

It depends on the terminal whether an edit function is available or not.

3.1.2. Number Formats

Most commands need parameters which are usually numbers. The monitor uses the following convention regarding the representation or base:

By default, all numeric inputs are regarded as HEXADECIMAL numbers

- Characters '0..9' and 'A..F' or 'a..f'.

- Decimal numbers can be input by prefixing the number with the two characters "**D ' "** or with the character "**!**".

Example

e.g. D ' 100 or !100 equate to 100 dec.

The default input format can be changed to decimal by the command "DECIMAL".

- Characters '0..9'.
  In this case, hexadecimal numbers can be input by the two prefix characters "H ' " or with the single character "$".

Example

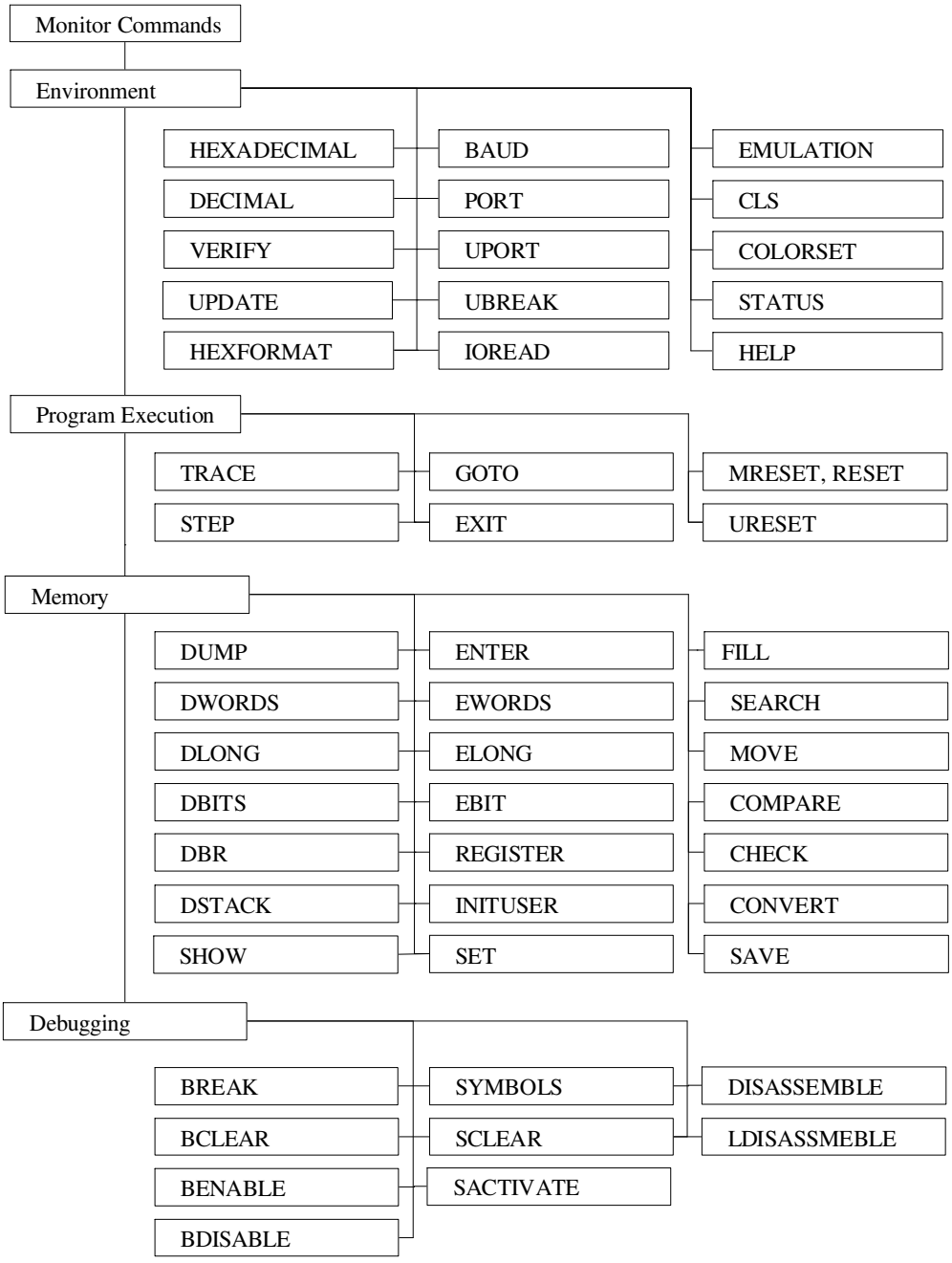
e.g. H ' F00 or $F00 equate to 0F00 hex.

NOTE: Numbers output by the monitor are always in HEX format !

The monitor features the capability of accepting symbol names for parameter inputs. To use this option, Symbol information must be available (see SYMBOL command group).

3.2. Monitor Commands

The Monitor Commands are grouped into different classes.
The figure below gives an overview:



3.3. Environment Commands

3.3.1. HELP, H, ? [Command]

Print Help Text

The HELP command prints a short overview of the most important monitor commands.
Shortcuts for this are H and ?.
If a command is named, specific help is displayed.

```
>help d↵
```

... Displays all commands with 'D' as first letter.

3.3.2. HEXADECIMAL, DECIMAL

Specify Default Input Mode

The HEXADECIMAL or DECIMAL command specifies the default format of numbers required for command parameters.
Power On default is HEXADECIMAL.

```
>decimal↵
```

... Changes to decimal input format.

3.3.3. VERIFY [ON | OFF]

Enable/disable Memory Write-Read-Verify mode

Most of memory accesses requested by the user via monitor are back-read if VERIFY mode it is set. Correct writing can be checked this way. Without parameter the current mode is displayed otherwise it is set as requested.
Power on default is ON.

```
>verify off↵
```

... Disables the write-verify-check.

NOTE: Some peripheral controller registers or board registers are write-only or have different function when read or are valid only for one read access. These registers will always cause an ERROR-message.
E.g. if UART data register is written to for sending one char (by EDIT-command) and it is re-read, a possible received character lost. Switch VERIFY off if such registers are manually edited.

NOTE: If VERIFY is switched off the correct writing of break/trace points is not checked. Therefore, VERIFY should be disabled only if necessary.

3.3.4. UPDATE [CHANGE | ALWAYS]

Change colouring mode of REGISTER command

UPDATE controls the highlighting of the changed CPU register (see commands REGISTER, EMULATION). If ALWAYS is selected the colouring is cleared after every display request. If CHANGE is selected it keeps on colouring the last changes until the next modifying has been done. Without parameter the current mode can be displayed.
Power on default is ALWAYS.

```
>update change↵
```
... Selects Change mode.

3.3.5. HEXFORMAT [INTEL | MOTOROLA]

Change record format of SAVE command

This command is used to change the format of a HEX record dump via SAVE command (see command SAVE). Without parameter the current mode can be displayed, otherwise the requested format is set.
Power on default is INTEL.

```
>hexformat moto↵
```
... Selects Motorola format

3.3.6. BAUD [*OPTION*]

Set transmission rate of the external UART port

This command sets the transmission rate of the external UART port. Without parameter the current mode can be displayed.
Options: 300, 1200, 2400, 4800, 7200, 9600, 19200, 38400

```
>baud 38↵
```
... Sets 38400 Baud for the external port.

If the rate has been changed, the terminal baud rate must be changed, too. If they can not immediately synchronise each other, the Monitor Reset button should be pressed.

NOTE: Another way to change the baud rate is the Auto-Connect function: The Escape key (ASCII 27) of the terminal connected to the external port of the Evaluation Board should be kept pressed to automatically repeat sending. Then the Monitor Reset must be pressed (do not release the Escape key). After this, the monitor checks the terminal baud rate and sets this rate as its own.

NOTE: The Auto-Connect function might be used for setting the user's favoured baud rate that is stored as default of the terminal program. It might also be used to re-synchronise the Evaluation Board, if an unexpected error occurred during download and if the previous baud rate could not be reset.

NOTE: The power on default is 9600 Baud. If the value was changed by command or Auto-Connect the monitor keeps this value also after a Monitor Reset. However, the next power on reset will reset the rate to 9600 Baud.

The baud rate functions are NOT available for the internal UART port.

3.3.7. PORT [INTERN | EXTERN]

Link Monitor I/Os to a RS232 port

Selects the internal UART (UART 0 of the microcontroller) or external UART (on-board chip device) for monitor communication. All monitor I/Os are done via this port. After changing the connector must be plugged in the appropriate interface. Without parameter the current mode can be displayed.
Power On default is EXTERN.

```
>port intern↵
```

... Sets the internal UART as monitor port.

NOTE: This setting is maintained also after Monitor Reset. However, Power On reset sets the default value.

3.3.8. UPORT [INTERN | EXTERN]

Link BIOS-I/Os to a RS232 port

Selects the internal (UART0 of the microcontroller) or external (on-board chip device) UART for user communication via BIOS calls. All BIOS I/Os are done via this port. Without parameter the current mode can be displayed.
Power On default is INTERN.

```
>uport extern↵
```

... Links the BIOS I/Os to the external UART. This way, only one terminal is needed for both, the monitor and the BIOS I/Os.

NOTE: This setting is maintained also after Monitor Reset. However, Power On Reset sets the default value.

3.3.9. UBREAK [ON | OFF]

Enable User Break function

User program execution can be stopped by pressing any key of the external port terminal, if JP37 is enabled. Without parameter the current mode can be displayed.
Following controller resources are used: The vector in the user interrupt vector table, the interrupt control register ICR01, the external interrupt port, the trigger level register, the interrupt level register and the interrupt enable flag are set. Before every user execution step, interrupt signalling for external interrupt #3 is cleared.

NOTE: The CPU interrupt level register and the interrupt enable flag within the processor state should not be changed. In case the Monitor wants to continue the User program (Trace, Go, Exit or Step) and the processor state is not properly set, the User is warned and the processor state reset.
Do not use this function, if the external interrupt #3 is already used.
The User Break interrupt level is "1".

3.3.10. IOREAD [ON | OFF]

Enable/Disable I/O addressing

When the TRACE command is executed some addresses and values are displayed next to the disassembled instructions. This can be completely disabled for I/O-instructions (transfer, bit, e.g. MOVX I:10,A), if the OFF option is selected. However, accessing via other instruction (e.g. physical address) does not avoid to read the I/O-register.
Default setting is ON.

```
>ioread off↵
```

... Disables address checking for I/O instructions.

3.3.11. EMULATION [ON | OFF]

Specify ANSI emulation features.

Some terminal programs are able to decode special escape sequences which change the actual colour or result in other screen manipulations. The monitor can be configured to generate such sequences to print its output messages using different colours and to clear the terminal window on certain points. The current mode is displayed, if no option is specified.

```
>emulation on↵
```

... Enables the monitor to use ANSI-sequences.

Note: These terminal emulation functions are not supported by ProMan !

3.3.12. CLS

Clear Screen

The CLS command will print the ´Clear Screen´ escape sequence if ANSI functions are enabled (see EMULATION).

3.3.13. COLORSET [*OPTION*]

Select Monitor colour set

One of four various colour sets can be set here. They are called as their background colour. Without parameter the current mode can be displayed.
Power On default is BLACK.

Options: BLACK, BLUE, GREY, CYAN.

```
>colorset cyan↵
```

... Changes to cyan background colour.

3.3.14. STATUS

Display Information

The STATUS command can be used to display information on the monitor status.
For example whether the monitor is in HEX or DEC input mode.

3.4. Program Execution Commands

3.4.1. TRACE [*nINSTR*]

Trace Instruction

The TRACE command can be used to single step an User program, beginning at the PCB:PC register address. 'nInstr' specifies the number of instructions which shall be executed automatically. If not specified, the trace will start in interactive mode.

```
>trace 11↵
```

... 17 (decimal) instructions are executed after this.

NOTE: The Program Address Counter can be set to a specific address using the "R IP *val*" command prior to using the trace command.

```
>register IP fb3305↵
```

The interactive operation works according to the following loop.

LOOP

- The monitor displays the actual register values and prints the disassembled instruction. If source addresses data or destination addresses are used (e.g. direct 16 bit address or EA-address), these are displayed with address and the value before execution. (See IOREAD command.)
- Then the User is prompted to press the 'SPACE', 'RETURN', 'O', 'R', 'V', 'B' or 'N' key to execute the instruction or press any other key to abort the trace operation.
- Continue loop if not aborted.

This way, the register values are always displayed before and after executing an instruction.

Special key functions:

- **'SPACE'** or **'RETURN'** will execute one instructions at a time.
- If **'O'** is used at a CALL subroutine instruction, the subroutine is executed without tracing (trace over).
- If **'R'** is used at a Branch instruction, a taken branch will not be traced. This means, the Trace point is set only at the instruction following the current one. As long as the program reaches this branch instruction and the condition causes to branch to the alternative following address, control will not be gained back by the Monitor. Only if the condition is not met the program will reach the Trace point and return to the Monitor
- **'V'** is the counterpart of 'R'. Only the branch is traced. The Trace point is set only at the alternative following address. The program is executed as long as the branch condition is not met.
- **'B'** causes to branch regardless of whether the condition is met or not.
- **'N'** is the counterpart of 'B'. The execution is continued at the next instruction after the conditional branch regardless of whether the condition is met or not.

NOTE: If conditional branches are traced with special function 'B' or 'N' the normal processing is changed. This might be used to influence processing. However, this can also cause miss-operation of the User's program.

Instruction Group	SPACE RETURN	**O** (Over)	**V** (oVer)	**R** (oveR)	**B** (Branch)	**N** (Next)	**X** eXit
INT	trace	trace over	abort				abort
CALL							
common	trace						
JMP/JCTX							
RET/RETI							
BRA							
Bcc	trace both branch and no branch		trace only branch	trace over branch	branch always	branch never	

Important:

NOTE: The monitor uses the INT9 instruction and the corresponding vector in the user vector table to realise the Breakpoint and Trace operation. Thus, this instruction must not be invoked by the user program and the vector at FFFFD8h (equal to the content of shadow RAM at xFBFFDh) must not be overwritten.

NOTE: As TRACE, STEP, BREAK are realised by INT9, these functions need a valid system stack when the user program is running or traced. TRACEing, STEPping or BREAKing an uninitialised program may cause malfunction.

This is important, too, when a valid stack is changed to another valid value. Because two values (bank and offset) might be changed, there can be an invalid "gap" when the one has changed but the second part is still to be changed. Depending on whether this undefined stack pointer is pointing to unused RAM or not, executing an INT9 instruction can cause misoperation.

E.g.: The current system stack pointer value is F80157h in the lower area of page 8 (RAM 0). This value is to be changed to an area in page A, to FA8000h. After setting SSB to the new value (FAh) the system stack pointer is pointing to FA0157:

```
MOV     A,#H'FA
MOV     SSB,A
```

After this instruction the system pointer of the user program points within the monitor data area. When the INT9-instruction is executed (TRACE step or breakpoint), the stack operation may destroy monitor data.

```
MOVW    A,#H'80000
MOVW    SP,A                    ; S-flag set of course
```

After this instruction the stack is defined and all trace functions can be used.

This problem will normally occur only in the start routine (INIT16L.ASM-module), that can be skipped by "GO _start _main" command. The advanced programmer can use register setting commands before critical trace steps to define a useful temporary system stack. If the monitor command

>register SSP 8000↵

is requested before tracing the "MOV SSB,A" instruction, there will be no problem.

See also INITUSER command.

3.4.2. STEP [OVER] [*NINSTR*]

Trace without displaying information

This is the hidden version of TRACE (see TRACE). There are no outputs at all, unless a breakpoint is reached. The parameter OVER avoids to jump in subroutines by CALL or INT. The execution stops when the set number of steps is done, a breakpoint is reached or an instruction could not be traced (e.g. BIOS-calls).

```
>step over 15↵
```

... Executes 21 instructions and does not follow subroutines.

See also NOTE of Trace command.

3.4.3. GOTO [*STARTADDR* | ,] [*STOPADDR*]

Continues execution of User Program

The GOTO command is used to switch control from the Monitor to the User program. The user program execution is started with the current CPU register contents (see 'R' command) at the location specified by the PC register. If a *StartAddress* is specified, the PCB and PC registers are set to this value.

A *StopAddress* can be specified which sets a temporary breakpoint. If the User program stops at any breakpoint before the *StopAddress* is reached, this temporary breakpoint is automatically removed.

```
>goto ff3266↵
>goto ff3266 ff32a7↵
>goto,80219↵
>goto,_main↵
>goto _start _main
```

... The first example starts program execution at address FF3266h. The second one sets an additional temporary break at FF32A7h. The third example continues execution at current PCB:PC position and stops at 080219h. The last examples use symbolic addresses.

NOTE: The address parameter relate to the first byte of any multiple byte instruction. Using another address (e.g. second byte of the instruction) causes misoperation.

3.4.4. EXIT

Exit Monitor Shell

If the monitor is called by a User Program via the BIOS interface as a shell program (indicated by ">>" prompt), program control can be returned to the user program using the EXIT command.

```
>exit↵
```

See also NOTE of Trace command.

3.4.5. [M]RESET [POWERON]

Monitor Reset

This command is equivalent to pressing the Monitor Reset button. The parameter "poweron" causes the monitor to proceed the Power On start-up. The monitor is completely re-initialised. However, this effects only the monitor program. This does not cause a Power On for the microcontroller device.

```
>reset↵
>reset poweron↵
```

... The first example resets the monitor. The second example completely re-initialises it.

3.4.6. URESET [0 | 1]

User Program Reset

The monitor will initiate a hardware reset, so the user program will start at the address pointed to by its reset vector. The parameter indicates the User Mode (USER0/USER1) to be set and causes different memory mappings (see fig. 2, p 11).

```
>ureset↵
>ureset 1↵
```

... The first example keeps current User Mode and starts at the address stored at FFFFDCh. The second example sets User Mode 1 before hardware reset.

NOTE: This command is similar to pressing the User Reset button. However, the User Mode register (see fig. 4, p 12, board controls) is not preset to the SW3/3 value, but to the mode parameter, if specified.

3.5. Memory Commands

3.5.1. DUMP [*STARTADDR* [*ENDADDR*]]

Dump memory area in byte format

The Dump command displays the memory contents of the specified address range.
If no *EndAddress* is specified, a memory area of 64 bytes is displayed.
If no *StartAddress* is specified the dump is continued at the following location of a previous dump.

```
>dump↵
>dump a1000↵
>dump a1000 a1100↵
>dump,a1200↵
```


3.5.2. DWORD [*STARTADDR* [*ENDADDR*]]

Dump memory area in word format

The Dump Words command dumps memory contents in 16 bit word format.
(See also "DUMP" command).

3.5.3. DLONG [*STARTADDR* [*ENDADDR*]]

Dump memory area in long format

The Dump Long Words command dumps memory contents in 32 bit word format.
(See also "DUMP" command).

3.5.4. DBIT [*STARTADDR* [*ENDADDR*]]

Dump memory area in bit format

The Dump Bit command dumps memory contents in single bit format.
(See also "DUMP" command).

3.5.5. DSTACK [*NWORD*] [*L*]

Stack Dump

The Stack Dump command dumps the actual stack area contents. The number of words can be specified. Using the 'L' flag, the monitor evaluates the frame pointer to get the return address. Note, that interrupt or far call return addresses differ from that. Specifying only the 'L' starts dumping four bytes before frame pointer (SPB:RW3).

```
>dstack l↵
>dstack 10 l↵
```

... First example shows the stack beginning at (FP-4) up to (SP). The second example shows 32 bytes from (SP-2*16) up to (SP).

3.5.6. DBR [*FIRSTBANK*],[*LASTBANK*]

Dump Register Bank

The DBR command can be used to dump the register bank memory area.
Without parameters, the current active register bank is printed.
A specific bank (0..31) or range can be specified by operands.

```
>dbr↵
>dbr 9↵
>dbr 9 1a↵
>dbr,1a↵
```

3.5.7. EDIT [*BYTEADDR* [*DATA*]]

Edit byte data in memory

The Edit command allows the modification of single memory locations, beginning at the specified *ByteAddress*. (If not specified, the previously specified address is used.) The monitor will display the actual address and the contents. The user can enter new data values or simply press RETURN to move to the next address location. To quit the edit procedure press the ESC key or enter /↵. If *Data* is given, it is written immediately.

```
>edit fa1299 30↵
>edit fa1290↵
>edit _Counter !100↵
```

... The first example writes 30h at FA1299h. The second starts interactive mode at FA1299h. The third example writes 100 (64h) at symbolic address "_Counter".

3.5.8. EWORD [*WORDADDR* [*DATA*]]

Edit word data in memory

The Edit Word command allows modification of 16 bit word memory locations. It operates in the same manner as the EDIT command.

3.5.9. ELONG [*LONGADDR* [*DATA*]]

Edit long word data in memory

The Edit Long command allows modification of 32 bit word memory locations. It operates in the same manner as the EDIT command.

3.5.10. EBIT [*BYTEADDR*[*:BITADDR*] [*DATA*]]

Edit data in memory bit by bit

The Edit Bit command allows modification of single memory bits. This command works in a similar manner to the EDIT command. However, a bit address behind the *ByteAddress* might be specified.

```
>ebit a1:2 1↵
```

... This example disables the microcontroller PLL-clock by setting bit 2 to 1.

NOTE: Because the values are read first and the modified value (bit set/reset) are written back, some I/O registers with different read/write function are not set properly.

3.5.11. REGISTER [*REGNAME* [*VALUE*]]

Show Registers, Change Register Contents

The register command, without parameters displays the CPU register contents (AH, AL, SP, IP, FP, the current register bank RW0..7, ILM, RP, CCR, ADB, DTB, DPR, USB, USP, SSB, SSP, PCB, PC):

```
 AH   AL    SP     IP     FP     RW0  RW1  RW2  RW3  RW4  RW5  RW6  RW7
0000 0000 000000 000000 000000  0000 0000 0000 0000 0000 0000 0000 0000
ILM RP   CCR       DTB  ADB  DPR      USB  USP  SSB  SSP  PCB  PC
 0  00 -S-----      00   00   00       00 0000   00 0000   00 0000
```

If ANSI has been enabled (EMULATION ON) all registers that have been changed are different coloured with different background to easily identify the changes.

If *RegisterName* and *Value* are specified, the register will be updated.

If the value is not specified, the monitor will enter an 'edit register mode'. It will print the current value and prompt for a new value. The user can either specify a new value or simply press RETURN to keep the actual value. Then the next CPU register and its content is displayed and can be changed. To exit this edit mode, press the ESC key or type /↵. This mode is available only for pure CPU register, but not for combined (e.g. IP).

```
>r al f000↵
>r rw0↵
>r ip ff0553↵
>r t 0↵
```

NOTE: Some "registers" are no actual register. SPP (stack pointer offset, SSP or USP), SPB(stack pointer bank, SSB or USB), SP (stack pointer, SSB:SSP or USB:USP), IP (instruction pointer, PCB:PC) and FP(frame pointer, SSB:RW3 or USB:RW3) are composed of a bank register and a belonging pointer. Others are part of the processor state PS (ILM, RP, CCR, flags I, S, Z, C, T, V, N of CCR).

3.5.12. INITUSER

Pre-define system stack and instruction pointer

Initialises stack (SSB, SSP, RW3) and/or instruction pointer (PCB:PC) in order to use a stack area within the monitor data RAM and to set the instruction pointer to the address pointed to by the current reset vector.

```
>inituser↵
```

Use this command to initialise and test your application software after a program download, when the stack is not set yet. See NOTE of Trace command.

The two operations of this command must be confirmed with 'Y' or ENTER or cancelled by any other key.

3.5.13. SHOW *OPTION* [*VALUE1* [*VALUE2*]]

Show data

This command displays various types of data or tables. If a data type is requested the necessary bytes are converted and displayed.

Option: BYTE, WORD, LWORD, FLOAT, DOUBLE, STRING:
various data formats are converted,
value1 is expected as address or symbol
(any string is skipped after 1,024 chars)

EI2OS: descriptor table entries are output
(0...Fh = 16 possible channels),
values are start and end descriptor channel number

VECTOR: interrupt vector table entries are output,
(0...FFh = 256 possible vectors)
values are start and end vector number

```
>show float ff0487↵
>show string _message↵
>show ei2os 7
>show vector 8 a↵
```

... The third example displays EI2OS-descriptor of channel 7. The last example displays interrupt vector 8 to 10.

3.5.14. SET *OPTION* [*VALUE1* [*VALUE2* [*VALUE3*]]]

Sets various types or tables.

Options: BYTE, WORD, LWORD, FLOAT, DOUBLE, STRING:
various data formats,
value1 is expected as address or symbol,
value2 is the new value (note, that strings finished with ' " ' are extended with zero termination else not).

EI2OS: EI2OS descriptor table entry,
value1 is descriptor channel number (4 Bit),
value2 is the transfer counter (16 Bit),
value3 is the I/O address (8 Bit),
value4 is the status byte (5 Bit),
value5 is the buffer address (24 Bit).

VECTOR: interrupt vector entry,
value1 is the vector number (8 Bit),
value2 is the address (24 Bit),
value3 is the mode byte (8 Bit).

```
>set float ff0043 3.66e-2↵
>set string _message "hELLO World"↵
>set string _message "Hello↵
>set ei2os 4 f 22 10 f80300↵
>set vector 8 _start 88↵
```

3.5.15. FILL *STARTADDR ENDADDR DATA*

Fill Memory area.

The Fill command fills a specified memory area with the specified data value.

```
>fill fa0000 fafff 0↵
```

... Clears RAM bank 2.

3.5.16. SEARCH [*STARTADDR*] [[*D1* [[*D2*] ...[*D8*]]] | "*ANYTEXT*"]

Search for a specific byte pattern

The Search command will search for a specific data pattern in memory beginning at the *StartAddress*. The data pattern is specified by up to 8 byte values or short text.
Note: The *StartAddress* must by specified using at least 4 characters (e.g. '00F0'), otherwise the first parameter is interpreted as a data pattern and the default address is used.
If a matching data pattern is found, the address is displayed. In this case, if the search command is used again without parameters, the search will be continued after the previous matching location.

```
>search ff0000 ff1000 00 83 f7↵
>search ff0000 ffffff "Hello"↵
```

... First example searches for the 24 Bit value F78300h. The second example searches for the five bytes "Hello" (case sensitive).

3.5.17. MOVE *STARTADDR ENDADDR TARGADDR*

Move memory area.

The Move command moves the contents of a specified memory area to a different location beginning at the *TargetAddress*.
The original memory area remains unchanged unless source and target area overlap.

```
>move 80000 81ffff 82000↵
```

3.5.18. COMPARE *STARTADDR ENDADDR TARGADDR*

Compare memory areas

The Compare command compares one memory area specified by StartAddress and EndAddress with a second memory area beginning at the TargetAddress. Memory locations with different contents are displayed.

```
>compare 80000 81ffff FF3357↵
```

3.5.19. CHECK *STARTADDR ENDADDR COUNT*

Examine memory area

Checks the memory area specified by StartAddr and EndAddr. The write and read back test is proceeded *Count* times.

```
>check 84000 8ffff 4↵
```

NOTE: The memory area is completely overwritten after this operation. Do not check areas containing valid data.

3.5.20. CONVERT *VALUE*

Print value in various number formats

The CONVERT command can be used to print a different representation of a value.
For example, if a hex number is specified as operand value, the decimal and binary equivalents are printed.

```
>convert 777↵
```

3.5.21. SAVE *STARTADDR ENDADDR* [*PROGSTARTADDR*]

Generate a hex record dump

This command dumps the specified memory area formatted as HEX- or S-record lines. This might be used for saving smaller areas in order to re-load them in a later session. The records are output to the terminal without any protocol. It should be stored using the ability of most terminal programs to create a LOG-file. Further hints are given on use.

ProgStartAddr is a possible program start address coded in special record lines. It is transferred to the PCB and PC register when it is re-loaded. Note, that the record format can be changed with the HEXFORMAT command.

```
>save 80000 81fff↵
```


3.6. Debug Commands

3.6.1. BREAK [*BNUMBER BADDR* [*OCCCOUNT*] ['*S*'] ['*W*' *WADDR WCNT*]]

Set Breakpoint/Snapshot/Watch-area

If no parameters are specified, the Breakpoint command will display the current breakpoint list. If a *BreakpointNumber* and a *BreakpointAddress* is specified, a breakpoint is set at the specified program location. The BreakpointNumber (0..E) is maintained as a reference. It will be displayed when using the DISASSEMBLE command and must also be specified to clear a specific breakpoint.

When the user program is executed, it will stop when a breakpoint is reached and the breakpoint information (CPU registers, Breakpoint location) will be displayed.

Specifying an *OccurCount* value allows the processor to 'run over the breakpoint' *OccurCount*-times before the breakpoint is activated.

Additionally, by setting the watch memory area flag ´W´, the breakpoint information can be extended to add a dump of a specified memory area.

´W´ must be followed by a start address *´wAddr´* and a byte count *´wCnt´*.

Specifying a snapshot flag 'S' converts the breakpoint into a 'Snapshot-point'. That is, the breakpoint information is displayed whenever the snapshot-point is reached, but program execution continues.

Using the occur counter in this case, program execution will stop when it reaches zero.

```
>break ff0122↵

>break 1 ff0152 10 s w ff0004 b↵
```

... The first example sets a simple breakpoint at ff0122. The second one sets a Counter-Snapshot Watch: Eleven bytes beginning at ff0004h are displayed sixteen times, then the User program is stopped.

NOTE: - If a previously used *BreakpointNumber* is specified again, the previous breakpoint is overwritten.

- Do not specify the same *BreakpointAddress* more than once !

- The monitor stores its breakpoint information in a Monitor RAM section which is not initialised (cleared) after a Monitor Reset. A check sum mechanism makes sure the information is valid. Thus, if an 'out of control' user program has to be stopped by Reset, the specified breakpoints remain valid unless the user program has destroyed its own code or the Monitor RAM.

NOTE: - The breakpoints are removed temporary if other monitor commands want to access the original contents.

See also NOTE of Trace command.

3.6.2. BCLEAR [BNUMBER]

Clear Breakpoint

If no BreakpointNumber is specified, this command will clear all breakpoints otherwise the break point with BNumber is cleared.

```
>bclear↵
>bclear 1↵
```

3.6.3. BDISABLE [*BNUMBER*]

Disable Breakpoint

If no *BreakpointNumber* is specified, this command will disable all breakpoints. Their data are still defined, but the breakpoint is not written into the user program RAM.

```
>bdisable↵
>bdisable 1↵
```

3.6.4. BENABLE [*BNUMBER*]

Enable Breakpoints

If no *BreakpointNumber* is specified, this command will enable all breakpoints. This command also resets the breakpoint occur counter.

```
>benable↵
>benable 1↵
```

3.6.5. SYMBOLS

Type Symbol information

This command will type all symbol names and values, if available.

The monitor can accept symbol names as command parameters and will display symbol names within in the 'DISASSEMBLE' and 'TRACE' commands, if symbol information is available. Symbol information can be included within the User Program (see example programs) or can be separately downloaded into any free RAM area.

```
>symbols↵
```

NOTE: The default area of the symbol table starts at F92800h above the monitor data, if the "MAP2SYM.EXE" program is used. Do not link any programs or data areas to this region.

3.6.6. SACTIVATE

Activate Symbols

In the case where the symbol table has not been activated during program download, the SACTIVATE command can be used to scan the memory for a symbol table and initialise the symbol management.

```
>sactivate↵
```

NOTE: If another symbol table was already active and a new one should be activated, the old one must be cleared (see SCLEAR) if it starts at a lower address. Otherwise, the old table will be found again.

3.6.7. SCLEAR

Erase Symbol Table

The Clear Symbols command will delete the symbol and thus disable the symbolic debug features.

```
>sclear↵
```

3.6.8. DISASSEMBLE [*START* [*NINSTR*]]

Disassemble Memory

The Disassemble command disassembles the instructions beginning at address *Start*. If *nInstr* is specified this number is disassembled. Sixteen instructions are displayed if *nInstr* is not given. Without *Start* parameter a default address is used.

```
>disass ff01022↵
>disass _main 30↵
```

3.6.9. LDISASSEMBLE [*START* [*NINSTR*]]

Disassemble one Instruction

The Ldisassemble command works similar as the Disassemble command. However, if *nInstr* is not specified, only one instruction is displayed.

```
>l ff01022↵
>l,5↵
```

... The first example disassembles one instruction at address FF01022h. The second example displays five instructions.

3.7. Program Download

Programs developed on the PC and symbol information can be downloaded onto the evaluation board using software tools found on the associated CD-ROM.

One download function is enclosed in ProMan within the "Monitor" menu.
A stand alone download program is also provided called "HEXLOAD.EXE". This program can also be invoked using a batch file, which specifies some default parameters.

Examples are given in the "how to get started" chapter 5.

Detailed knowledge of the download protocol is not needed to operate the board.
Should a special download program be required, the protocol can be found in the appendix.

4. Monitor BIOS Interface

The monitor incorporates a software interface for user programs providing functions for basic terminal input / output operations.

When developing application or user programs, it is common practice to include some additional code which prints out debug or other status information during the first trial runs, to verify the proper operation of certain procedures.

To relieve the user from coding appropriate input/output procedures (which will be needed just for debug purpose and not in the final program), the BIOS interface can be used.

The following is a detailed description on the available BIOS functions.

Examples can be found within the example programs discussed in chapter 5.

The BIOS interface is realised as a call entry table (24 Bit address) located at the bottom of the monitor (shadow) ROM and has the following structure:

Table Header at FC0000h "EVAKITBIOS V0.4"+00h	offset 00	The first 16 Bytes contain a signature Call the following location for:
I/O Function Calls (ASM)	10	BIOS Functions (parameters in registers)
I/O Function Calls (C)	14	BIOS Functions (parameters on stack)
Print CPU register	18	prints all CPU register (like R command)
Monitor Shell	1C	Call a monitor shell
Monitor Reset	20	Monitor (soft) Reset (like MRESET command)
reserved		

The start address of the table is at location **FC0000**h. The table header should be checked before calling a table entry.

NOTE: After return from an I/O call the register ADB might be changed. After C-function calls the working registers RW4 and RW5 might be changed too. The I/O calls keep on using the stack of the user program.

BIOS functions must be called with "physical" addresses (ASM) respectively as "__far" functions (C).

All I/O requests caused by BIOS calls are done via the user port (without ANSI support). If no second terminal is connected (e.g. to the internal port) the user port should be linked to the monitor port (e.g. "UPORT EXTERN" command) after every Power On reset (soft/hard).

4.1. I/O Function Calls (ASM/C)

4.1.1. I/O Function Calls Parameter Passing

The most commonly used function vectors are those for ´I/O Function Calls´.
Specifying a function code byte and additional parameters will execute a specific I/O function.
See also "TINYBIOS.H" for easy to use macro definitions .

Two different table entries are available.

The (ASM) entry is optimised for assembly language calls, which pass the required parameters in CPU registers.
The (C) entry is optimised for C-program calls, which pass the parameters via the stack.

The following convention is used to specify function code and parameter:

a) Function Call (ASM)

Register AH :	first parameter (byte/word)
upper byte AL :	function number
lower byte AL:	second parameter (byte)
RL2:	32 Bit value if needed

```
e.g.  MOV     A,#('*')      ; ASCII code of '*'
      MOV     A,#H'100      ; function number shifted 8 bit left
      CALLP   FC0010        ; call BIOS function
```

b) Function Call (C)

1st word on stack:	first parameter (byte/word/lower 16 bits)
2nd word on stack:	second parameter (byte/upper 16 bits)
3rd word on stack:	function number

```
e.g.  int (* __far BiosCall)(int, int, int) = 0xFC0014L;

      BiosCall('*',0,1);
```

On return, a 16 bit value is passed back to the calling routine via the AL register.
Generally, a negative return value is regarded as an error exit code.

NOTE: C-function calls always expect 3 bytes parameter on stack. After returning from C-subroutines, the register RW4, RW5 and ADB may have different values.
Pointers are expected to point within the bank indicated by DTB.

The following table lists the functions provided via I/O Function Calls:

Function	1st Param.	2nd Param.	FCode.	Return Value
Poll Character Received*	-	-	00	1:Rec, 0:No
Send Char on RS232*	Output Char	-	01	char sent
Receive Char on RS232*	-	-	02	Input Char, -1
Print a String*	DTB:Address	-	03	0, -1 (error)
Input a String*	DTB:Buffer	max. len	04	(Esc, ↵, Tab)
Print a HexByte*	Byte	-	05	-
Print a HexWord*	Word	-	06	-
Print a HexAddress	LongWord		07	-
Print a HexLongWord	LongWord		08	-
Wait n milliseconds	milliseconds	-	09	-
En/Disable BreakPnt.	1 : En., 0 : Dis.	-	0A	-

* NOTE: The port specified by UPORT command is used. The default port is the internal port. This port is pre-defined for 9600 Baud, no parity, 2 stop bits only on power on reset. User may change this parameter (see MB90600 User Manual).

However, the monitor port is re-initialised during a reset (to 9600 Baud, if internal port is linked to the monitor, or set to the selected BAUD-value, if the external port is used). In this case it does no matter, whether it is linked to the external or internal port. This is important to note, if monitor and user are linked to the same port.

4.1.2. I/O Function Calls Detailed Description

4.1.2.1. [00] Poll Character Received

This function works in a similar way to PC-BIOS functions. It checks, if a character is available in the keyboard buffer or not.

Since the evaluation board uses a polling type of software UART, the functionality is restricted. The function will just scan the Rx-receive line for some time and see if a character is detected.

A function call to actually read a character will immediately return the buffered character if the previous poll request had returned "1" else the read character request waits for the next character.

4.1.2.2. [01] Send Character on RS232

This function will transmit a single character (byte) via the RS232 transmit line, to be displayed on the host computer terminal.

4.1.2.3. [02] Receive Character on RS232

This function will poll the RS232 receive line until a character is detected and finally return the received character.

4.1.2.4. [03] Print a String

This function will transmit a complete string of character via RS232, to allow message outputs on the computer terminal. The string termination character is 00h.

The pointer parameter refers to the bank set in DTB.

4.1.2.5. [04] Input a String

This function reads a complete string from the RS232 interface.

The input is terminated when the RETURN, TAB or ESC key is pressed. If the max. string length is exceeded the following characters are ignored as long as a termination char or BACKSPACE is input.

The user is responsible for the size of the provided buffer. It has to have the length of the specified (second) parameter plus one byte. A CR-LF sequence is not returned after terminating input.

4.1.2.6. [05] Print HexByte

This function converts the byte parameter into a 2-digit hex-ASCII number (capital letters) and transmits it to the terminal.

4.1.2.7. [06] Print HexWord

This function converts the word parameter into a 4-digit hex-ASCII number and transmits it to the terminal.

4.1.2.8. [07] Print HexAddress

This function converts the lower 24 bits of the long word parameter (**RL2** if ASM-call) into a 6-digit hex-ASCII number and transmits it to the terminal.

4.1.2.9. [08] Print HexLongWord

This function converts long word parameter (**RL2** if ASM-call) into a 8-digit hex-ASCII number and transmits it to the terminal.

4.1.2.10. [09] Wait for n Milliseconds

This function waits for the specified number of milliseconds (+5 %). It does not use any interrupt or any timer. The timing is realised using special instructions with known sum of cycles. Therefore, changing the machine cycle or the wait cycles (auto ready function - see MB90670 User Manual) of the upper memory area will change the precision.

4.1.2.11. [0A] Enable/Disable Breakpoints

Depending on the byte parameter, this function can temporarily deactivate or re-activate all breakpoints enabled by the operator.

4.2. Other BIOS Function Calls (ASM)

The following BIOS function vectors can be used to simplify debugging.

Calls to these vectors can be placed at critical program locations as an alternative of manually setting a breakpoint.

4.2.1. Print Register Values

The print register value entry can be used to display the current CPU register values without modifying any registers or flags.

```
e.g.  CALLP     H'FC0018          ; Print CPU Registers

or    void (* __far MyRegister)(void) = 0xFC0018L;
      ...
      MyRegister();
```

The register values are output to the user port without colouring, even if ANSI has been enabled.

4.2.2. Monitor Control

The Monitor Control entry can be used to call a monitor shell. A debug version of a user program can use such calls at critical points.
The monitor can then be used to display or modify the memory and register contents.
From this point onwards, it is also possible to single step the following user program.
As long as the requested shell is active the prompt is changed to ">>" instead of ">". The monitor command EXIT will finally return control to the user program.

```
e.g.  CALLP     H'FC001C                ; Call Monitor Shell

or    void (* __far Shell)(void) = 0xFC001CL;
      ...
      Shell();
```

NOTE: Make sure the monitor was initialised (Power On Reset, Monitor Reset) before this entry is called the first time! The Monitor Shell is NOT a second shell. On this BIOS call the original shell is reactivated. It uses neither the stacks of user program nor initialises it a second monitor environment.

NOTE: Using the monitor control function is an alternative to setting a breakpoint .

5. How to get started, Example Sessions

This chapter provides some hands-on examples on utilising the available software tools and shows how to operate the board.
The first kind of examples show a more comfortable, higher level type of approach, using the C-language programs and batch utilities.
The second example shows a low level approach to become familiar with specific problems.

All example programs are also listed in appendix B.

Before trying to re-compile/assemble the example programs, make sure the associated software is installed properly. A set "SET TMP=tempdir" command should be placed in your AUTOEXEC.BAT file, pointing to a directory (preferably a RAMDISK) used to store temporary files.

e.g. SET TMP=E:\

The search path should include a path to the ProMan directory

e.g. PATH=C:\PROMAN;...

Further last minute information can be found in the README.1ST file.

It is also useful to include a `MODE COM1:n,8,2` in the AUTOEXEC.BAT when using the standalone terminal program.

5.1. First Approach

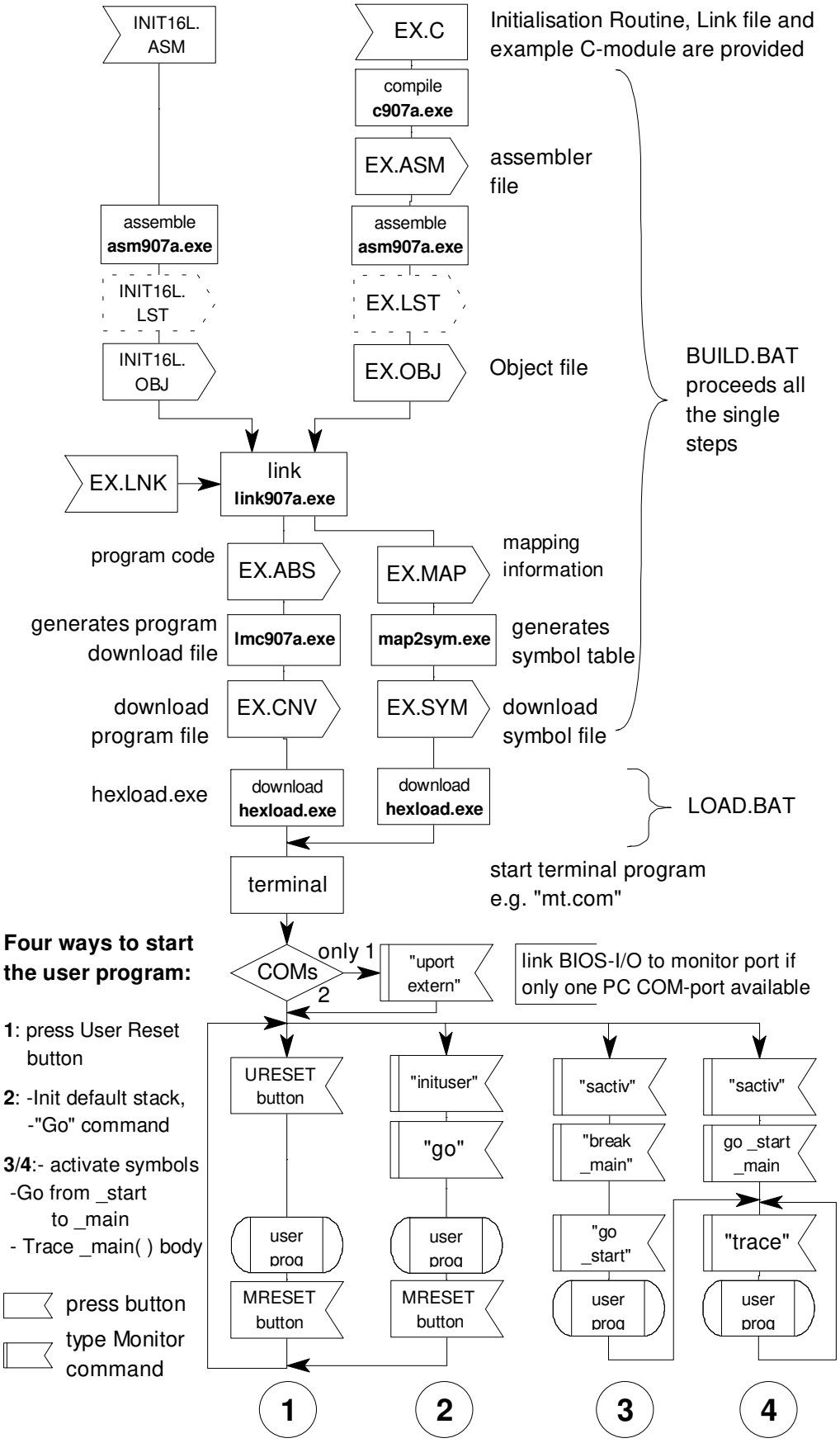


Figure 6: First Approach - necessary steps

In the example directories there are some batch files. These files expect the environment variable MB675 to be set. The instruction

```
SET MB675=C:\FUJITSU\EVABOARD
```

must be included in the AUTOEXEC.BAT. This is the path to the program directory (compiler, assembler ...). The correct structure of the path depends on your installation. Within this directory one must find the INCLUDE directory containing the standard header files.

Each Link file (EX.LNK) must be updated with the correct Library directory (which is mostly located within the same path as the previous set).

```
-LIB    C:\FUJITSU\EVABOARD\LIBRARY\CLIB\RAMCONST\C907S.LIB
```

After booting the PC the batch file "BUILD" in the EXAMPLE directory must be started. Now the start routine and the example should be compiled, assembled, linked and converted to record files (EX.CNV or EX.HEX). A symbol file should also be generated (EX.SYM).

```
C:\>BUILD
```

Further iterations can be build faster, if the example source file has been changed only (e.g. to correct a mistake). In that case it is not necessary to assemble the INIT16L.ASM-module.

```
C:\>BUILD EX
```

... does only compile, assemble and link the specified module.

Now the Evaluation Board should be prepared (power, RS232 connector). Then the program data and symbol table must be uploaded to

```
C:\>LOAD EX
```

After downloading the program and symbol table, the terminal program should be started.

```
E.g.  C:\>MT.EXE
```

The examples demonstrate some I/Os via BIOS calls. Therefore, the user I/Os should be linked to the monitor, if no second terminal is available. Monitor command:

```
>UPORT EXTERN
```

The symbol table should be activated. A break might be set after the start routine.

```
>SACTIVATE
>GOTO _start _main
```

This program can be started also with User Reset button.

NOTE: For a first session two download files (EX.CNV and EX.SYM) are already available. So the steps compiling, assembling, linking, record file generation could be skipped. They need only to be downloaded.

5.2. Lowest Level Example

As a further step the Evaluation Board should be used without downloading anything to become familiar with some commands. A simple ">" refers to Evaluation Board monitor commands and "C:\>" refers to DOS commands. The Evaluation Board should be prepared (power, RS232 connector).

The configuration should be set to ANSI, if supported by the terminal program.

```
>EMULATION ON↵
```

A very simple program doing nothing can be created by writing NOP codes (No Operation) in page E. The command

```
>FILL FE0000 FEFFFF 0↵
```

is the fastest way to do that. The code can be checked by disassembling this area:

```
>DIS FE0000↵
```

There are only NOP instructions. The "program" is started by:

```
>GO FE0000↵
```

Of course, once started nothing happens anymore. The controller proceeds all 65,536 NOP-instructions and continues at the first one again. The program must be finished with Monitor Reset button. To get back control without reset a breakpoint should be set at FE0100h.

```
>BREAK FE0100↵
>BREAK↵
 ==== Breakpoint List ====
No. T.  Addr.  OCnt.IniC.  Watch.Cnt Act.(Symbol)
00. B  FE0100  0000 0000   ------ --  *
>GO FE0000↵
```

However, the execution is not stopped. The reason is the invalid stack. Since the monitor and user program has to share one microcontroller, the breakpoints require the user's system stack (see NOTE of TRACE command). The INT9 instruction is called, but the stack has no data. Reset must be used again.

```
>REGISTER↵
 AH   AL    SP     IP     FP     RW0  RW1  RW2  RW3  RW4  RW5  RW6  RW7
0000 0000 000000 FC0042 000000  0000 0000 0000 0000 0000 0000 0000 0000
ILM RP   CCR       DTB  ADB  DPR      USB  USP  SSB  SSP  PCB  PC
 0  00 -S-----      00   00   01       00 0000   00 0000   FC 0042
>
```

The above listing shows the current stack pointer SP = 000000 (SSB and USB are initialised to 0 on reset, SSP and USP are undefined). The next data will be stored at 00FFFFh. However, the current mapping provides no RAM there.

```
>REGISTER SSB F8↵
```

sets the system stack pointer anywhere within page 8;

```
>GO FE0000↵
```



```
 AH   AL    SP     IP     FP     RW0  RW1  RW2  RW3  RW4  RW5  RW6  RW7
0000 0000 F80000 FE0100 F80000  0000 0000 0000 0000 0000 0000 0000 0000
ILM RP   CCR       DTB  ADB  DPR      USB  USP  SSB  SSP  PCB  PC
 0  00 -S-----       00   00   01       00 0000   F8 0000   FE 0100
FE0100   00              NOP
```

The GO starts the program. If it reaches address F80100h it is interrupted and curent register set is shown.

```
(     >INITSTACK↵       )
```

would initialise a default stack above the monitor data RAM. This might be used instead of setting "REGISTER SSB 8".

Some different breakpoints should be set:

```
      >BREAK 2 FE0120 S↵
      >BREAK 1 FE0110 W F81234 3↵
      >GO↵

 AH   AL    SP     IP     FP     RW0  RW1  RW2  RW3  RW4  RW5  RW6  RW7
0000 0000 F80000 FE0110 F80000  0000 0000 0000 0000 0000 0000 0000 0000
ILM RP   CCR       DTB  ADB  DPR      USB  USP  SSB  SSP  PCB  PC
 0  00 -S-----       00   00   01       00 0000   F8 0000   FE 0110
FE0110   00              NOP
W2: D F81234: 00 00 00
```

The snapshot breakpoint shows the registers but continues. The watch point dumps the requested bytes. A combination of snapshot, watch point and an occur counter creates a breakpoint that dumps the memory area every time, the breakpoint is passed, as long as the counter is not zero.

```
      >TRACE↵

W2:
 AH   AL    SP     IP     FP     RW0  RW1  RW2  RW3  RW4  RW5  RW6  RW7
0000 0000 F80000 FE0120 F80000  0000 0000 0000 0000 0000 0000 0000 0000
ILM RP   CCR       DTB  ADB  DPR      USB  USP  SSB  SSP  PCB  PC
 0  00 -S-----       00   00   01       00 0000   F8 0000   FE 0120
FE0120   00              NOP                                          <
```

... Starts single step. RETURN causes execution of one instruction. After this ESCAPE finishes tracing.

```
      >BDISABLE
      >UBREAK ON
```

... disables the breakpoints and activates the User Break option. JP3 must be enabled. A register request shows the changes for the enabled interrupts.

```
      >REGISTER
 AH   AL    SP     IP     FP     RW0  RW1  RW2  RW3  RW4  RW5  RW6  RW7
0000 0000 F80000 FE0121 F80000  0000 0000 0000 0000 0000 0000 0000 0000
ILM RP   CCR       DTB  ADB  DPR      USB  USP  SSB  SSP  PCB  PC
 2  00 IS-----       00   00   01       00 0000   F8 0000   FE 0121

      >GO
```

continues the program. Execution stops as soon as any key is pressed.

```
<<User Break>>
 AH   AL    SP     IP     FP     RW0  RW1  RW2  RW3  RW4  RW5  RW6  RW7
0000 0000 F80000 FE0094 F80000  0000 0000 0000 0000 0000 0000 0000 0000
ILM RP   CCR       DTB  ADB  DPR      USB  USP  SSB  SSP  PCB  PC
 2  00 IS-----      00   00   01       00 0000   F8 0000   FE 0094
FE0094   00              NOP
```

To start a program by reset a vector must be placed:

```
      >BENABLE
      >SET VECTOR 8 FE0000 88
 # ICR         Name/Source       VecAddr  Vector Mode  ICR: Level EI2OS
 08 -- Reset (all types)          FFFFDC  880000  88
```

However, if User Reset is pressed the program does not stop. The program has to initialise its system after reset:

external bus ($\overline{WR}$, *CLK*, wait cycles) and MCU speed (PLL clock), stack, data areas.

A start routine is provided within the INIT16L.ASM file, which should be linked to all projects programmed in C. This routine can be set as User Reset vector by including "__set_vect(8, start, 0x88);" (compiler built-in function) in C-source code.

However, as long as the previous example is executed using the GOTO command, the system configuration does not need to be set. The default configuration of the monitor is used.

NOTE: Critical system settings (e.g. disabling A16..19 address lines by EDIT command) can cause miss-operation. If an user program initialises the system, these settings are used by the monitor, too. However, only the CKSCR register (A1h) is watched and reset to 16 MHz mode for monitor shell operation. CKSCR is restored, if the user program is continued.

5.3. Examples using ProMan:

In the following example sessions, instructions are given on how to use ProMan to demonstrate the examples, which will give just a brief impression of the ProMan features. To learn more about ProMan and its features please refer to the ProMan documentation.

When installing the software onto the hard disk, a directory called "C_EXAMP" will be created.
Before starting ProMan, this should be made to the current working directory.

e.g. `CD C:\FMC_16L\PROJECT\C_EXAMP`

Start ProMan by typing `PROMAN35`

When the initial copyright message window appears, press RETURN to continue.

If ProMan is started for the first time, the screen should be empty apart from the menu and function bars.
If ProMan has been started before, it will come up with the previously used project settings and edit windows which were active when exiting ProMan the last time.

To continue the example, a sample project should be opened by selecting the **Project** main menu and the **Open Project File** item. (In case ProMan was started previously, the current project might have to be closed first by selecting **Close Project File**.)

6. Evaluation Board Hardware

The evaluation board provides various microcontroller signals which can be connected to external devices. The controller resource functions are available on the connectors JP39, JP40, JP41, JP42 and JP38/18..21 (the lower address bus is no controller resource).

External peripheral devices can be connected via the address/data bus on JP38 and JP42 and the select signal /EXP, /CS0..2, /CTRL. For more information please refer to the following pin assignments and the schematics in appendix A.

6.1. Pin Assignment of the MB90675 Microcontroller

6.2. Pin Assignment of the Board

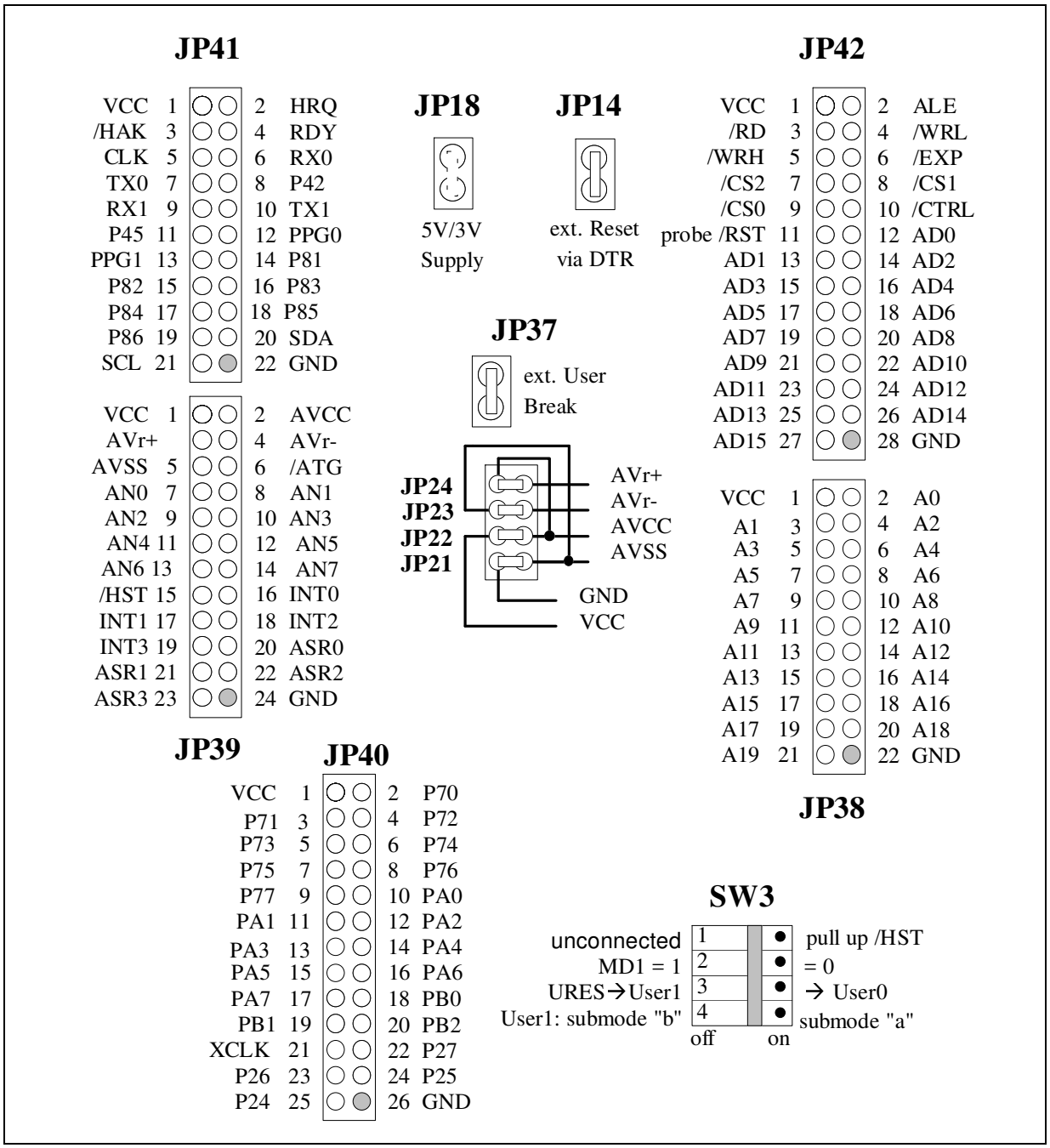


Fig. 7 Evaluation board connector and jumper assignment

The MB90675 evaluation board was designed, that it can be used for the MB90675 series of microcontrollers. If a 100 pin QFP socket is mounted, it is possible to use either a MB90676, MB90677, MB90678 controller circuit or the F²-MC16 series emulator via a special pod.

NOTE: The pull-up restores RPACK3 (on socket next to JP39) has to be removed or shifted (common pin of pack unused) for the user's application circuit if the A/D converter is used.

The schematics and PCB layout can be found in appendix A.

Ports P0, P1, P2 and P20..33 of the MB90675 microcontroller are used as an external microprocessor bus, connecting the Monitor EPROMs U10, U11 and the User RAMs U22, U23, Address latches U2, U3 are used to de-multiplex the combined address/data bus.

The PAL or GAL U24 is used as a memory decoder and provides chip select signals for the EPROM (/ROM), User RAM (/RAM), Memory mapped control registers in GAL U16 (/CTRL) and a select signal (/EXP) which can be used for expanded memory. The enable signals to set the LED displays latches are also generated there.

The mode registers within U16 control the operation of U24. GAL U6 implements eight control registers selected by (/CTRL, A1, A2, A3). Four of them are bit wide and read/write capable using /WRL, /RD, and AD0 (mode register, single LED register). The remaining four registers are read/write capable using /WRL, /RD and AD0..7 (external UART), write capable using /WRL, /WRH, AD0..15 (LED displays).

GAL U16 is also used to generate a reset signal on /RES if either the Master Reset Button (/MRES) or the User Reset Button (/URES) is pressed.

If one of these reset buttons is activated, the Mode registers are either cleared or set (see Board Controls).

7. Appendix

7.1. Appendix A: PCB Layout and Schematic

7.2. Appendix B: Example Program Listings

7.2.1. Example 1

```
/*
                  First Example for Monitor Test
*/
#define MAXLED 20

#include "global.h"
#include "tinybios.h"


/* some global variables to work with symbols */

BYTE  LEDtext[MAXLED] = {0,
0x76,0x79,0x38,0x38,0x3f,0x0,0x71,0x3e,0x1e,0x6,
                        0x78,0x6d,0x3e,0,0,0,0,0,0};
LWORD       Delay = 0x10000L;
int   LEDctr = 0;
char Msg[] = "\15\12\tPress any key and set new message or ESC to
reset\15\12";

int LEDfun(int);
void Input(void);
__interrupt extern void start(void);      /* init routine */

int LEDfun(int upps)
{
      LWORD ctr = Delay;
        LEDDISP1 = ~LEDtext[LEDctr++];
        LEDDISP0 = ~LEDtext[LEDctr];
        if (LEDctr >= MAXLED-2)
            LEDctr = 0;
        while(ctr--);
      return (LEDctr);
}

void Input(void)
{
      bios_put('#');
      bios_inp(Msg+3,sizeof(Msg)-4);
      bios_str("\15\12New message is:");
      bios_str(Msg);
}

void main(void)
{
      int ctr;                      /* stack vari */
      int ch;
      __set_vect(8, start, 0x88);         generates the reset vector
      bios_str("\15\12\t Hello User, look for the text on LED displays");

      while (1) {
                  bios_str(Msg);
                  for (ctr = 3*MAXLED; ctr; ctr--) {
                        LEDfun(ctr);
                        if (bios_poll()) {
                              if (bios_get() == 27)
                                    bios_res();
                              Input();
                        }
                  }
```

```
      }
}
```

7.2.2. Example 2

```
/*
                  Example for searching bad addresses
*/

#include "global.h"
#include "tinybios.h"


#include <ctype.h>
#include <samples\mb906fm.h>

#undef EXTERN
#define EXTERN
#include <samples\ic.h>
#include <samples\tb.h>
#include <samples\low.h>
#undef EXTERN
#define EXTERN extern


#define TBMAX    8                        /* number of timebase timer
interr*/
#define TBLEV    6                        /* level of tb interrupt */

#define POLLCTR 100000L;

extern __interrupt void start(void);
void Poll(void);
__interrupt void Timer(void);
void initWDT(void);
void SetClk(BYTE cks);

/* some global variables to work with symbols */

BYTE LEDhex[] ={0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xD8,
                0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E};
long ctr;
int keep = 0;


const char Msg[] = "\15\12\t----------------------------------------"
                "\15\12\t             MemTest at page 8"
                "\15\12\t----------------------------------------\15\12"
                "\15\12\t     ESC: Monitor Reset        "
                "\15\12\t       K: keep bad address&value "
                "\15\12\t 0-9,A-F: upper 4 bits of value   "
                "\15\12\15\12Press any key"
                "\15\12\t Write/Verify 64K block with: ";

void Poll(void)
{
         int ch;
         if (bios_poll()) {
             switch(ch = bios_get()) {
                 case 27:
                             bios_res();
                        break;
                case 'K':
                case 'k':
                        keep ^= 0xFFFF;
                        break;
```

```
                default:
                        if (isdigit(ch)) {
                                ch = ch-'0';
                                ctr &= 0xf; ctr |= ch << 4;
                        } else if (isxdigit(ch)) {
                                ch  = (ch&0xdf) -'A'+0xA;
                                ctr &= 0xf; ctr |= ch << 4;
                        } else
                                bios_put(7);            /* ding dong */
                 }
        }
}

__interrupt void Timer(void)
{
        static int tctr = 0;

        if (!TBTC_TBOF)                                 /* no request */
                return;
        if (!(tctr++ % 2))
                USERLED ^= 1;           /* switch every 4. occurr.*/
        TBTC_TBR = 0;

        Poll();
}
void initWDT(void)
{
        __set_vect(0x22, Timer, 0xFF);        /* anywhere in this module*/
        ICR11 = TBLEV;                            /* level, no EI2OS */
        TBTC = 0x9b;                               /* enable, reset, 2^19
cycl*/
}

void setClk(BYTE cks)
{
        CKSCR_MCS = 1;                                  /* switch off PLL */
        CKSCR_CS0 = cks & 1;
        CKSCR_CS1 = cks>>1 & 1;
        CKSCR_MCS = 0;                                  /* switch PLL on */
        while (CKSCR_MCM);

}

void main(void)
{
        int eflg, ch, keep2;
        unsigned errctr = 0, tmp;
        union {BYTE __far * fptr;
               WORD ctr ;
        } addr;

        __set_vect(8, start, 0x88);             /* reset vector */

        setClk(3);                              /* PLL clock 16 MHz */
        initWDT();                              /* Init timer interrupt */
         __set_il(7);                           /* ILM setting */
        __EI();                                 /* enable interrupts */
        LEDDISP0 = 0xFF;LEDDISP1 = 0xFF;        /* clear LED displays */

        while (1) {
            bios_str(Msg);
            addr.fptr = (WORD __far *)0xF80000L;
            for (ctr = 0; ctr <= 0xff; ctr++) {

               bios_hexb((BYTE)ctr);             /* test byte */
               bios_str("\b\b");
```

```
                        eflg = 0;                   /* no error yet */

                        do {
                            *addr.fptr = ctr; /* write */
                                if (*addr.fptr != (BYTE)ctr) {/* re-read */
                                    if (!eflg) {
                                         tmp = ++errctr;
                                         while (tmp & 0xFF00)
                                                 tmp >>= 4;
                                         LEDDISP0 = LEDhex[tmp&0xF]
                                         &(errctr & 0xF000 ? 0x7F:0xFF);
                                         LEDDISP1 = LEDhex[tmp>>4]
                                         &(((errctr&0xFF00)<0x0100
                                         && errctr&0xf00) ? 0x7F:0xFF);

                                         bios_str("    \15\12");
                                         bios_hexa((long)addr.fptr);
                                         bios_put(':');
                                         bios_hexb((BYTE)ctr);
                                         bios_put(' ');
                                         eflg = 1;
                                    }else if (eflg == 2){
                                         bios_put('.');
                                             if (++eflg == 0xff)
                                                 eflg = 2;
                                    }
                                    if (keep ) {       /* test as long as */
                                         if (eflg == 1) /* error ocurrs */
                                              eflg++;
                                              addr.ctr --;
                                    }
                                } else if (eflg)
                                    eflg = 0;
                                addr.ctr ++;
                        } while(addr.ctr);
                }
        }
}
```

7.3. Appendix C: Program Download Protocol

Download code files can be generated by converting the "*.ABS" file to "*.CNV" using:

```
C:\>lmc907a.exe.exampl.abs".
```

A file containing so called "S-record" or "HEX-records" should be written.

The evaluation board provides two different protocols in two formats for program download.

7.3.1. HexLine Command

e.g. >:10200000FFEEDDCCBBAA99887766554433221100F8↵

can be entered as a command. The ':' is interpreted as a download command and the specified data will be transferred into memory. The correct format and included checksum are checked! Note, that a download program using this mechanism must read the echo of each character and that the completing CR is echoed as CR-LF.

NOTE: This record contains only a 16 Bit address. Therefore a special record indicating the bank has to be sent, before an undefined address is written to.

Another format is started by 'S':

e.g. S2FF200011FFEEDDCCBBAA99887766554433221100D7↵

7.3.2. HexFile Download (HEXDWL)

The HexFile Download function works similar as the previous command, however :

- a dedicated download function is activated by sending the command 'HEXDWL↵
- Note, that the printable characters of the command are echoed, the CR (ASCII 13)is not echoed!
- a hex-line is transferred (but no character is echoed !)
- when the monitor has processed the hex-line, it will send the acknowledge character (06H).
- then the next hex-line can be transferred.
- The hex download function terminates if ESCAPE (ASCII 27) or a record with length 0 was downloaded.
- A checksum is built over each line and the monitor will respond with a not-acknowledge character (05H) if an error occurs.

NOTE: This function is used by ProMan© .

To download with the HEXLOAD.EXE program recommended use is:

```
C:\>hexload 2 -a -f examp1.cnv
```

This loads the examp1.cnv record file over PC-port COM2. HEXLOAD tests the current baud rate (-a) of the Evaluation Board and after connecting it switches to 38400 Baud for transmission. Finally, it returns to the previous baud rate.

7.4. Appendix D: Monitor Software Notes, Restrictions and Recommendation

As mentioned in chapter 2, the Monitor software resides in an EPROM memory bank in the upper address area XC0000h-XDFFFFh . Therefore, any address with upper bits (23..16) ----110- is accessing the monitor ROM.

In addition, some User RAM locations are reserved for monitor variables. Monitor reserved RAM sections are:

a) A small section in the User RAM area (x90000h-x92800h). This section is used for monitor variables

b) The control bank on page 7 contains only a few register but they are not completely decoded. So, only the odd bytes of the control register are free (excluding the LED Display registers).

c) The vector table entry for the INT9 instruction, (User RAM Address: FFFFD8-FFFFDBh) is initialised with a vector to monitor ROM. The monitor does also use the Exception interrupt (0Ah), but it may be occupied by the user. The external interrupt #3 (0Eh) and the associated register ICR01 at B1h are used for the User Break function (see User Break), if enabled.
External interrupt #3 shares ICR01 with external interrupt #2. Therefore, the User Break handler checks if an external interrupt #3 has occurred. If not, it jumps to interrupt 0Dh handler.

Other restrictions:

1.) Single Stepping certain code sequences:

Due to the implementation of the single step mechanism, the following assembler-code constructions can not be single stepped:

Conditional Branch to the same or following address

```
e.g.  LABEL1:   BBC 10:3,LABEL1

                BNC FOLLA
e.g.  FOLLA:    xxx yyy,zzz
```

Use constructs like:

```
LABEL:      NOP
            BBC 10:3,LABEL

            BNC FOLLA
FOLLA:      xxx yyy,zzz
```

The single step and breakpoint handling needs a valid stack segment. Therefore, set SSB:SSP/USB:USP to user RAM areas before reaching any breakpoint if the user program starts on reset. On Monitor Reset SSB, USB are zero and SSP and USP are undefined.

3.) Processor Speed, Power Management Functions

When developing programs which utilise or activate special on-chip registers, be aware that the monitor generally disables interrupts when it is active.

Note also, that switching the microcontroller into the STOP, SLEEP or WATCH modes will disable monitor control via the terminal. After reset the user break function is always disabled.

When the monitor is active, it always configures the clock control for max. speed. The user program speed is saved and restored when control is given to the user program.

The speed is pre-defined to 16 MHz machine clock (4 MHz main clock).

Recommendations:

1.) Hex download

Depending on the version of "LMC907A.EXE" (the tool to generate a HEX-file from an ABS-file) a start vector which points to program entry point is appended. However, the bank of this vector is always 0Fh, even if the "_start" function has been linked to another bank.

The download procedure of the monitor uses this vector (if appended) to set it as PCB and PC register. In order to start a program include the __set_vector() function and use the User Reset instead of typing "GOTO" without parameters.

8. Index