F^2MC-8L
MB89620/630/850 Series Evaluation Board

USER MANUAL

Read this first

This book describes the Fujitsu F^{2}MC-8L Starterkit_8 and how to use it with the provided tools.

How to use this manual

The goal of this book is to help you learn how to develop your own software in ANSI-C or Assembler with the Windows-based environment SOFTUNE. This book is divided into 6 parts. Parts 1-3 contains hands-on information so that you can start using the evaluation board the same day you receive it. Part 1 is a general description of the board. Part 2 contains installation instructions and part 3 will guide you through example sessions. Parts 4-6 contain detailed information about the monitor operation, the bios interface and the hardware details.

Important notice

This Starterkit contains an evaluation board, documentation and software on a CD-ROM. For last minute changes, please refer to the "Readme.1st"- Document in the Starterkit_8 section on the CD.

Fujitsu reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice.

Trademarks

Contents

1. Introduction

The Starterkit_8 is a low-cost, stand-alone application board that makes it easy to evaluate and demonstrate almost all features of FUJITSU's MB89620, MB89630 and MB89850 micro controller series.

Along with the supplied Windows-based development tools, it can be used as a system for user program developments. The board can be configured to use a MB89T625, MB89T637 or MB89T855 controller by simply exchanging the chip. By default, the board is equipped with an MB89T637.

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

A unique feature of this board is a “bank switching mechanism”, which allows the Monitor- EPROM to be disabled and to map a User RAM into the same address range. 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.

Each memory bank can be activated by a **separate Reset button**, allowing restart of either the Monitor or the User Program, beginning from a real hardware reset.
(Note: The Monitor is able to switch the banks automatically by software in order to perform special debug functions.)

Key Features

- Can be configured with MB89T625, MB89T637 or MB89T855
- 64 kB RAM
- On-board monitor featuring breakpoints, single-stepping, trace etc.
- All unused ports available
- Integrated Windows-based development environment

2. General Description

2.1. Installation

What you'll need :

- ✓ Host : IBM(-comp) PC with Windows 3.11 or Windows 95 and CD-ROM drive
- ✓ Power requirements : A power supply, capable of supporting 7-8V DC at about 150 mA, is required. Note that the power connector must be + at the shield and - in the centre
- ✓ RS232 : A 'Nullmodem' RS232 cable with a DB9 connector

To install the development environment, insert the provided CD-ROM and click on **TOP.PDF** from Windows. From the main menu, select "**Software Installation**", "**8-Bit**" and then "**Starterkit_8 Installation**". The following procedure will create a directory named "C:\FETOOL\" and automatically start the individual setup programs of :

- Softune (8-Bit MCU Version) : Fujitu's integrated development manager for Windows
- Language Tools : C-Compiler C96, Assembler ASM96, Linker LINK96, Librarian LIB96
- Utilities and examples

To setup the hardware, connect the power supply to the board and check that the User LEDs light up briefly and that the small red 'monitor program' LED (**MP)** remains on. When pressing the User Reset button, the small green 'user program'-LED (**UP)** should be on whilst the reset button is pressed (see figure1).

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 before re-connection of the power supply.

Now, the LEDs should still 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.

To access the board monitor for testing, any terminal program can be used (e.g. the provided terminal in the FMG_UTIL directory). The communication parameters are **9600 Baud, 8 Databits, no Parity, 2 Stopbits**. If a different terminal program is used and the DTR-line can not be set individualy, this effect can be disabled by removing the "Ext. Reset" jumper, located as in figure 1.

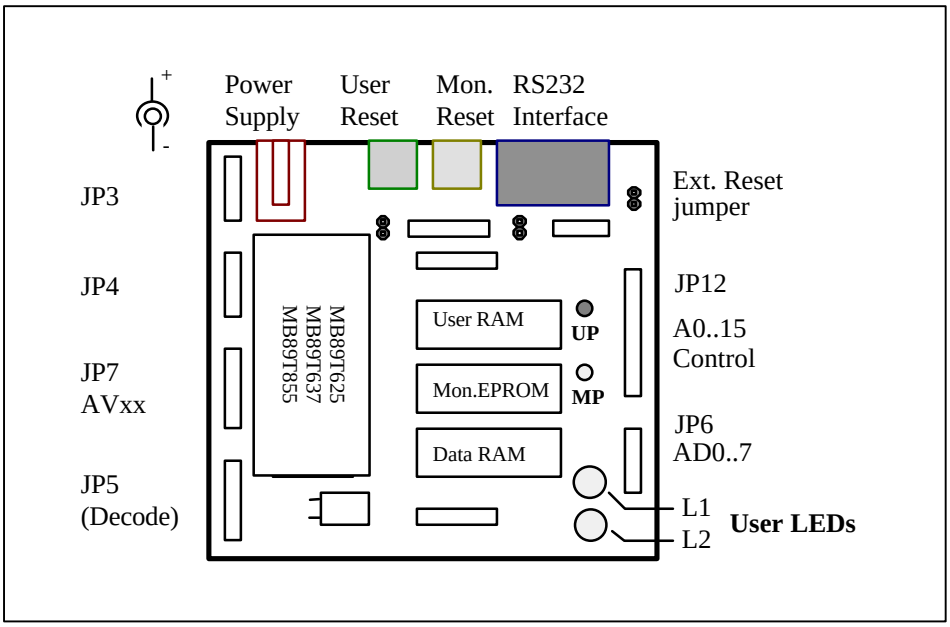


Fig. 1 : Evaluation Board Layout

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

```
*******************************************
**  Monitor for the FFMC8L Series        **
**  MB89xxxx Evaluation Board Vers. x.x  **
**  (C) Fujitsu Mikroelektronik GmbH 199x **
*******************************************

(C)
>
```

The ´(C)´ status message indicates that the monitor executed a ´cold´ reset after power up. Press the monitor reset button to generate a ´warm´ reset, and a ´(W)´ message will appear.

The evaluation board is now ready to use.

2.2. Technical Characteristics

Supply Voltage	7V ... 9V
RAM Area	aro. 62KB
ROM Area	32KB Monitor ROM
Microcontrollers	MB89T625, MB89T637, MB89T855
Serial Interface (RS232)	Half-Duplex, 9600 Baud, No Parity, 8 Data-bits, 2 Stop-bits
Debug Functions	All functions are realised in software including Breakpoints, Single-Step, Watch-Points
Board Restrictions	**Software:** CALLV#7 can not be used because it is needed for the breakpoint system implementation. **Hardware:** Port 0, 1, 2 are used for the external bus system and are therefore not available as I/O-Ports.
Controller Specific Features	**MB89T625** - 8-bit PWM Timer - 8-bit PWC Timer/Counter - 20-bit TimeBaseTimer - 16-bit Timer/Counter - 2x 8-bit SIO - 8x8 bit A/D - 5 ext. Interrupts - STOP, SLEEP mode - 512 Bytes of internal RAM **MB89T637** - 2x 8-bit PWM Timer - 8-bit PWC Timer/Counter - 20-bit TimeBaseTimer - 16-bit Timer/Counter - 8-bit SIO - 8x10 bit A/D - UART - STOP, SLEEP, WATCH mode - 4 Ext. Interrupts - Gear-function - Buzzer output - 32KHz Sub-Clock - 1 kByte of internal RAM **MB89T855** - 2x 8-bit PWM Timer - 20-bit TimeBaseTimer - Timer unit (dead timer/ ud counter/ 3ph. motor control) - 8-bit SIO - 8x10 bit A/D - UART - STOP, SLEEP mode - 512 Bytes of internal RAM

2.3. Evaluation Board Resources

The evaluation board was designed in order to leave as many micro controller resources available for user evaluation and applications as possible. All of the IO ports (port 3,4,5 and 6) are available to the external world via the on-board connectors.
The monitor software resides in a bank-switchable EPROM, thus not demanding any of the 64K external program or data RAM address space.
Only a small RAM area is reserved for monitor variables and memory mapped control registers (see memory map). Two of the control registers are linked to the LEDs L1 and L2 and can be used for simple user program indicators. Fig. 2 gives an overview of the memory structure of the evaluation board.

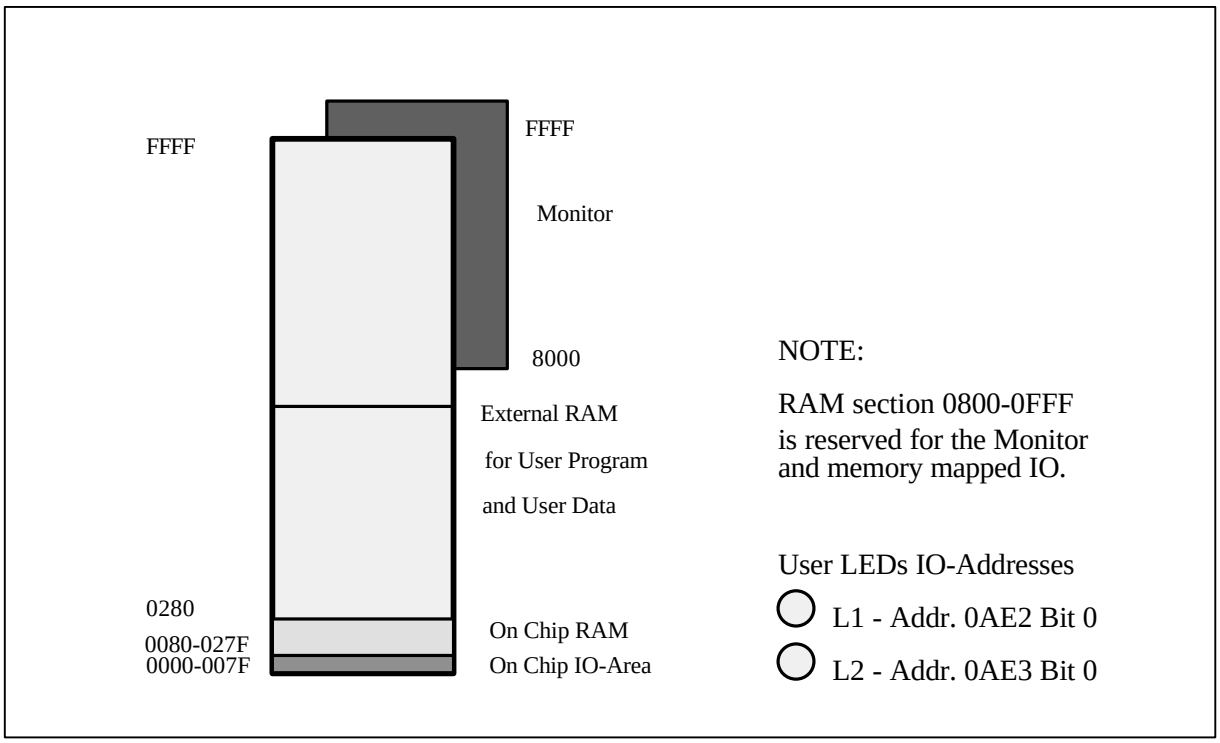


Fig. 2 : Memory Structure of the evaluation board

3. How to get started, Example Sessions

This chapter will provide some hands-on examples on utilising the available software tools for developing software and shows how to download code and data to the board.

3.1. Examples using Softune:

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

Please be sure to have the evaluation board powered up and connected to COM1 of your PC as described in chapter 2.1. After the software installation, double-click the Softune-icon (MAN896) to start the program.

3.1.1. The "LED8L" Example

Select "**Project/Open Project**" and open "LED8L.PRJ" in the "SAMPLE"-directory. A member list will appear as shown in Fig. 3 which contains all registered files :

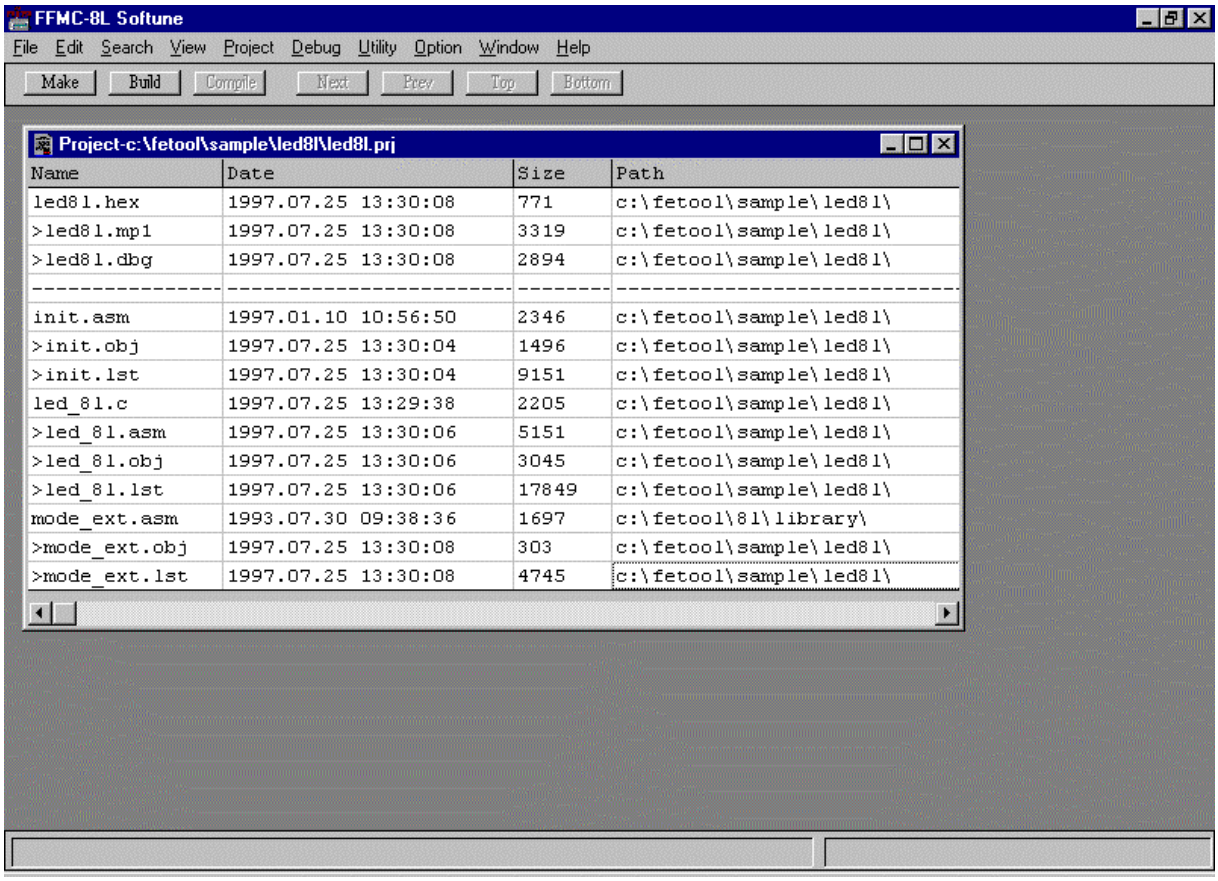


Fig.3. : Softune with member-list of the LED8L-project

The listed files in the member-list are :

- LED8L.HEX **the linker output file this module can be downloaded to the evaluation board**
- BEGIN.ASM **is a library file containing initialisation functions for C programs**
- LED8L.C **is the source file**
- MODE_EXT.ASM **is a library file containing reset vector and the processor mode byte**

NOTE: The "MODE_EXT.ASM" file is used when port 0..2 of the controller are used as an external bus interface, as on the evaluation board.
If projects migrate from the evaluation board onto user-developments, eventually using the "single chip mode" in conjunction with an In-Circuit Emulator or as an OTP, the "MODE_SNG.ASM" must be specified for proper mode byte setting.

To edit one of the files, double-click on the name and the editor will come up with the source file. As a first example, double-click on "**LED8L.C**" to see the sourcecode of the program.

Using the cursor keys the window can be scrolled and the program can be studied. The main program produces different patterns on the user LEDs, calling a time delay subroutine after each setting. Close the edit window again, before proceeding.

To compile, assemble and link the project files to an executable program file that can operate properly on the evaluation board, use the **Build**-button. (Note that there are two other functions available : **Make** creates the load module by compiling, assembling and linking only *updated* files and **Compile** affects individual C-files in the developing process). With these mouse-driven functions, the software designer can concentrate on coding and debugging programs with improved quality and efficiency, but with no knowledge of how to start and configure a compiler or linker !

When the compilation process is completed, a result window appears showing any errors and warnings. This example should compile without problems and complete assembling and linking without any errors. In case of errors, a double-click onto the error message activates the error-jump-facility and locates the appropriate line in the source file (try it by generating a simple syntax-error …!).

After linking, an executable program file (LED8L.HEX) is available. This program can be downloaded to the starterkit. To do so, open the **Utility**-menu and select "**LoadProgram**". You should see a progress-counter while the program is loading and a "RUN"-button will appear once the program is ready to execute. After executing the program this way, you finally should see the two LEDs flashing on the board…

3.1.2. Debugging examples

For simplified debugging, an additional symbol file can be generated. The **Utiliy**-item "**LoadSymbols**" will generate and download symbol information to the board, which can be utilised by the board monitor. After loading symbol information, the monitor can access the table via a pointer which is included in the information.

After successfully downloading program and symbol information, select the "**Terminal**" from the **Utility**-menu. A simple terminal will be called to access the monitor. (Any other terminal-program can be used instead)

When pressing RETURN, the monitor prompt ´**>**´should appear, or when pressing the monitor reset button, the initial message from the monitor should appear.

To see if the example program still works, use the G command. (Type G and RETURN)
Now the example program is executed on the evaluation board, thus ending the control via the Terminal. The LEDs should start flashing now.

A Monitor Reset must be issued, for example by pressing the monitor reset button, to enable communication with the evaluation board monitor again.

In the following description, some of the **monitor-debug functions** are demonstrated:

Use the **TS** (type symbol) command to **list the available symbol information**. This will give an overview of most interesting program labels, like _main, _wait etc.

Enter the **T** command to activate the **trace** mode. A program label ´===start===´ and some register information is displayed and the instruction at the program counter address is disassembled. This ´start´ location is the reset entry point of the example program. The code is that of the BEGIN.ASM source code module which contains the C-program initialisation procedures.

Some instructions may be traced by pressing RETURN.

The monitor will disassemble the actual instruction and wait for a key entry ('SPACE' or 'RETURN') to execute the instruction and to continue with the following one. Once you step into the "call _wait" instruction , it will take a long time to finish it.

To avoid to single step time consuming subroutines, you can enter the 'C' key at locations where such subroutines are called to execute them in real time.

So if the Delay Subroutine was already entered, it might be necessary to reset the PC register as in the beginning and start again.

Press the **ESC** key to quit the trace mode. Then enter the go,break command

G, _main

to complete execution of the initialisation procedures.
Program execution will stop at the beginning of the main program.
Now, trace again the next 9 instructions to see how the user LEDs are switched.

Finally, set a **break point** at the wait() subroutine by entering the

B 0 _wait

command.

Now, the example program can be reset by entering the user reset command **UR**. The program execution will always stop at the wait() subroutine entry.
Enter the **G** command several times to continue program execution and to observe the different user LED states.

Apart from the “GO” (g) command, there are many **ways to start a program:**

a) Press the UserReset Button → User Program activated (until Monitor Reset)

b) Use the monitor UR command to start the UserProgram at its Reset Entry
e.g. UR → User Program activated (until Monitor Reset)

c) Use the monitor GO command to start UserProgram at a specific location (in this example the Reset Entry point (main program loop)).
e.g. G 8000 → User Program activated (until Monitor Reset)
or G start
or G (starts at actual PC location which is also 8000 after a reset)

To demonstrate the **Breakpoint with the OccurCounter** function, try

B 0 _wait 4

This will execute the main loop 4 times. The example program will stop at the 5th time the breakpoint is hit.

To demonstrate the **Breakpoint/Snapshot** function, use

B 0 _wait S

This will effectively just set a snapshot point and the example program will run until the monitor reset is pressed. Each time the Snapshot location is executed, the monitor will dump the CPU registers.

These were some basic steps in evaluation board operation and program debug. To learn about all possible features please refer to the detailed command description in chapter 5.

3.1.3. The "BIOSDEMO" Example

The BIOSDEMO.C example program shows how to utilise I/O functions provided by the monitor software for user program purposes.
To get access to these functions, the EVABIOS.H header file is included and the EVABIOS.ASM library file is linked to the program.

The example program is self explanatory and can be compiled and tested as described in the previous example. Use a terminal to execute the program then you will see this simple dialog :

```
>g

****************************************
**  Demonstration of BIOS Functions  **
****************************************
'puts'-function :
 This is an output string

'gets'-function :
 Please enter your name : Master of the universe

 Thank you, Master of the universe

'getch'-function :
 Press any key !

 You pressed x

'putch'-function : M a s t e r   o f   t h e   u n i v e r s e

 The byte-check sum is : 53
 The word-check sum is : 0853

 That all folks ! Press any key for Monitor Reset
```

3.1.4. Program Creation

3.1.5.1 Using the NEWPROJ-example

The easiest way to develop software for the evaluation board is to modify the provided "newproj"-project. This is an already configured example for a MB89T637-equipped evaluation board which can be used to start developing without any modifications in Softune. It has the following registered files :

- INIT.ASM : An initialization file dedicated for developing software in C for the evaluation board
- NEWPROJ.C : A C-file with all nessesary definitions, but with an empty "main"-routine
- MB89630.ASM : The library file for all IO-definitions (includefile is "MB89630.H")
- INT89630.ASM : The library file for all Interrupt-vectors (includefile is "INT89630.H")
- MODE_EXT.ASM : The reset-vector / mode-byte library file suitable for the evaluation board

This means, all IO-registers (e.g. PDR3=0xFF) can be accessed in C and the interrupt service routines are already defined, but left empty in the C-source. The installed memory-map can also be used for the two other controllers.

3.1.6.2 Configuring Softune

To create completely new software, select "**New Project**" from the **Project** menu. Simply specify a name and select the microcontroller family you want to develop the program for. After that a new project-directory and –file (.PRJ) will be generated and a new member-list appears.

NOTE: At startup, Softune reads the initial file "man898.ini". This file contains information about directories, projects, utilities etc. Softune reverts to the previous usage conditions by reading this file. Edit this file only offline ! Most of the stored information can be changed menu-driven from Softune. Refer to the manuals or press F1 for help whenever you need it.

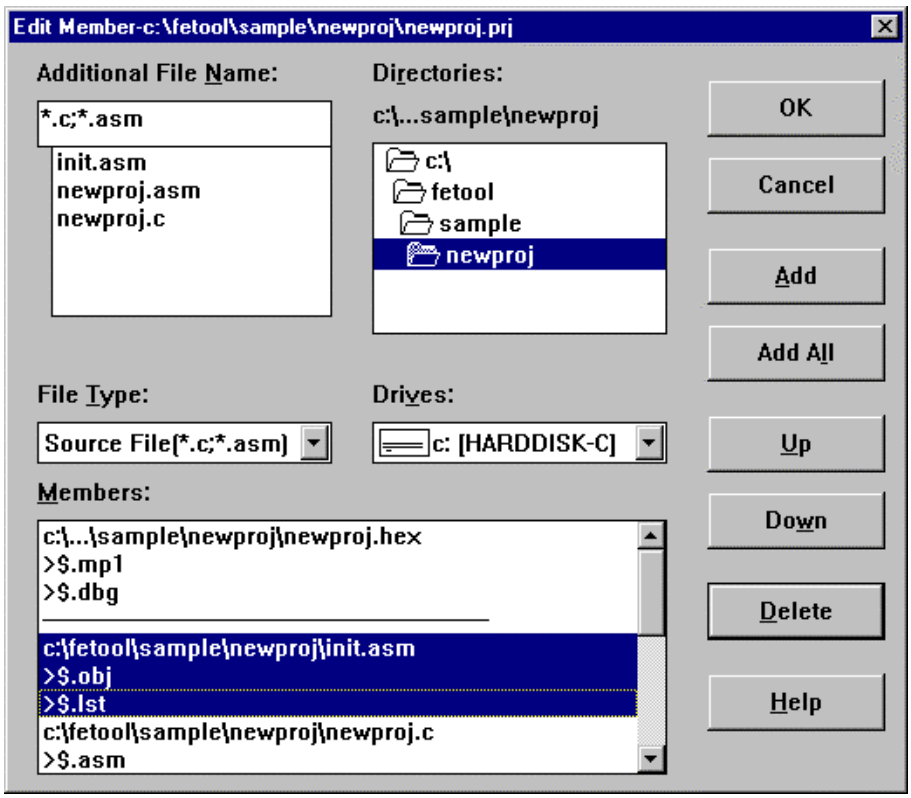


Fig. 5 : Configurating a Softune-project

To add source and library files to the member-list, type a new file name or select an existing file – then press ADD and OK to close the dialog. Source file types (.C .ASM) will be recognized automatically. **NOTE : The INIT.ASM-file should be the first registered file in the member list**

Always specify the memory map for a new project. This information is needed for the linker to generate absolute code that is executable on your target system. To specify the memory setup, select "**Edit Memory Map**" from the **Project** menu. Insert RAM and ROM size, the number of register banks used and all segment locations as shown below.

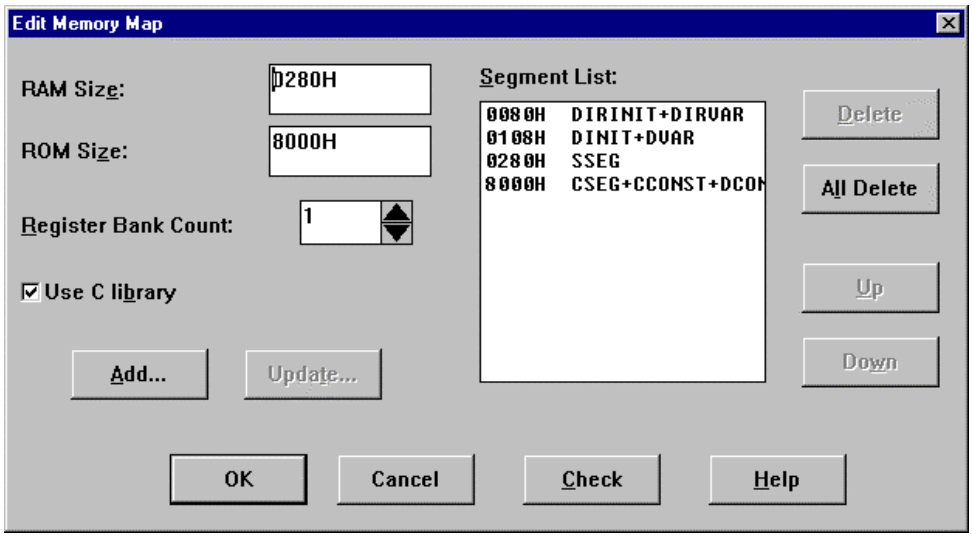


Fig. 6 : Configurating the memory map in Softune

The table below gives an overview of a typical memory-map used for the board :

Address space	**Memory area**	**Segment name**
8000 – FFFF	External RAM for Code, Constant Data etc.	CSEG,CCONST,DCONST ,DIRCONST
0280 - 7FFF	External RAM for Monitor	(not used by default)
0108 – 027F	Internal RAM: Variables (Data) and Stack	SSEG (Stacktop=27F), DINIT, DVAR
0100 – 0107	Registerbank #0	(always at 100)
0080 – 00FF	Internal RAM: fast access	DIRINIT, DIRVAR
0000 – 007F	IO-Area	(always at 0)

When writing programs which access these registers, dedicated header and library files which specify the processor specific control registers should be used.

For example for an MB89620 evaluation board, the linker file for all IO-definitions

```
C:\FETOOL\8L\LIBRARY\MB89620.ASM
```

should be used and has to registered in the member list. In the source-code the line

```
#include <MB89620.h>
```

should be inserted to declare the functions. Default path names can be changed in "**Set Environment Variables**" from the **Option** menu.

Further information regarding the C-compiler, Assembler and Linker can be found in the online-manuals or the full documentation which can be ordered separately.

NOTE : All language tools (C-Compiler, Assembler, Linker etc.) are DOS-programs. Therefore implementation of Fujitsu language tools in any individual development environment or the use of batch-files is possible.

4. Monitor Operation

4.1. General Operation

This chapter describes the *monitor* of the board in detail. The monitor can be operated via any PC terminal, like the included terminal in the FMG_UTIL-directory. The 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, the Monitor 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.

4.1.1. Command Input

Commands must only 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 plus appropriate parameters and then pressing RETURN to terminate input and execute the command.

The monitor provides the following edit functions:

- Mis-typed characters can be erased by using the BACKSPACE key.
- Pressing the TAB key, will automatically re-type the previous command.
- A complete command line can be skipped by pressing the ESC key.

4.1.2. Number Formats

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

By default, all numeric inputs are regarded as HEX 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 DEC by a monitor command.

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

Example

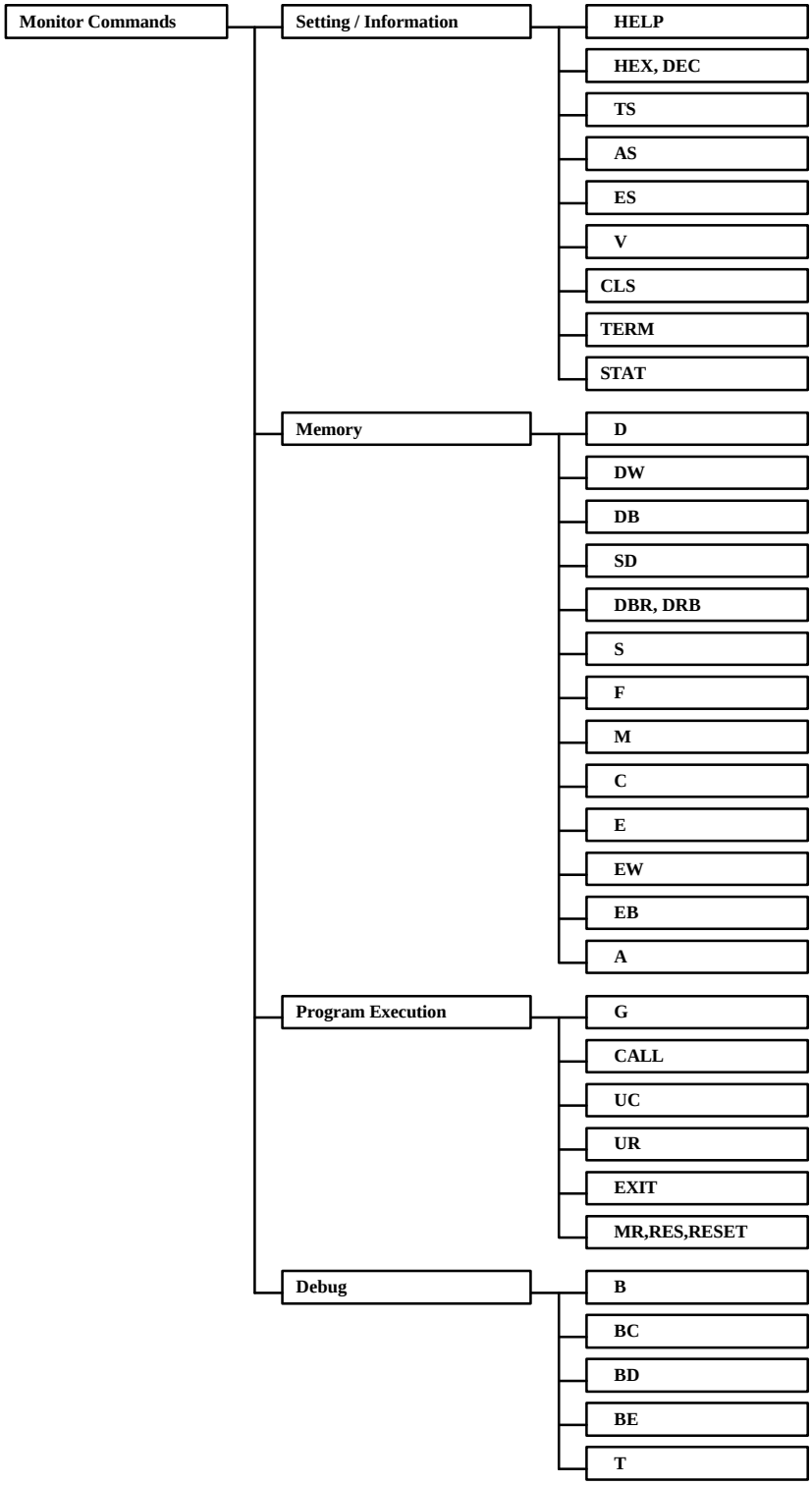
e.g. H ' F00 or $F00 or F00H equate to 0F00 hex.

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 TS command).

4.2. Monitor Commands

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



4.2.1. Setting / Information Commands

4.2.2. 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.

4.2.3. HEX, DEC

Specify Default Input Mode

The HEX or DEC command specifies the default format of numbers required for command parameters.

4.2.4. TS

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 'L' and 'T' 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.

4.2.5. AS

Activate Symbols

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

4.2.6. ES

Erase Symbol Table

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

4.2.7. V *VALUE*

Print Value in various number formats

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

4.2.8. TERM *FLAG*

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. To activate these features, the TERM command is used to set related flags:

Flag values:	0	use no escape codes
	1	utilise Clear Screen code
	2	utilise ANSI colour codes
	3	utilise Clear Screen & ANSI codes

Note that these terminal emulation functions are not supported by ProMan !

4.2.9. CLS

Clear Screen

The CLS command will print the ´Clear Screen´ escape sequence if enabled (see TERM).

4.2.10. STAT

Print Information

The STAT command can be used to print information on the monitor status.
For example if the monitor is in HEX or DEC input mode and additional information about the monitor BIOS version.

4.2.11. *Memory Commands*

4.2.12. D [*STARTADDR*] [*ENDADDR*]]

Dump Memory Area

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.

4.2.13. DW [*STARTADDR*] [*ENDADDR*]]

Dump Memory Area in Word format

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

4.2.14. DB [*STARTADDR*] [*ENDADDR*]]

Dump Memory Area in Bit format

The Dump Words command dumps memory contents in single bit format.
(See also 'D' command).

4.2.15. SD [*NWORD*] [*L*]

Stack Dump

The Stack Dump command dumps the actual stack area contents. A number of Words can be specified. Using the 'L' flag, the monitor evaluates each stack value and checks if it might be a return address. In that case the available symbol nearest to that address is printed.

4.2.16. DBR, DRB [*FIRSTBANK*],[*LASTBANK*]

Dump Register Bank

The DBR or DRB 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.

4.2.17. S [*STARTADDR*] [*D1* [[*D2*] ...[*D8*]]]

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.
Note: The StartAddress must by specified using 4 characters (e.g. '00F0'), otherwise the first parameter is interpreted as a data pattern and a 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.

4.2.18. F *STARTADDR ENDADDR DATA*

Fill Memory Area

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

4.2.19. M *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.

4.2.20. C *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.

4.2.21. E [*EDITADDR*]

Edit Byte Data in Memory

The Edit command allows the modification of single memory locations, beginning at the specified EditAddress. (If not specified, a 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 /↵.

4.2.22. EW [*EDITADDR*]

Edit Word Data in Memory

The Edit Word command allows modification of 16 bit word memory locations. Otherwise, it operates in the same manner as the E command.

4.2.23. EB [*EDITADDR*]

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 E command.

4.2.24. A [*STARTADDR*]

On-line Assembler

The A command enters the on-line assembler mode. Instructions can be entered using the F2MC8L/8LC assembly language.
To escape the on-line assembler simple press return on an empty line or press the ESC key.

4.2.25. R [*REGNAME* [*VALUE*]]

Show Registers, Change Register Contents

The register command, without parameters displays the CPU register contents (A, T, IX, EP, PS, SP, PC) and the current register bank registers R0..R7)
If a RegisterName and Value is 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 /↵.

- The register named 'CY' -which is not an actual processor register- can be changed in this mode. The 'CY' register is the monitor cycle counter used within the trace command.

4.2.26. *Program Execution Commands*

4.2.27. G [*STARTADDR*] [,*STOPADDR*]

Go to start User Program

The Go command is used to switch control from the Monitor to the UserProgram. The UserProgram 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 PC register is set to this value.

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

Examples:

e.g. G ,main
or
G reset,main

4.2.28. CALL *TARGADDR* [*PARAM1* [*P2*...]]

CALL Subroutine

The CALL subroutine command allows the calling of a User Program or Procedure starting at the Specified TargetAddress.

Parameters passed to the subroutine can be specified optionally and will be passed to the procedure on the stack (as done by the C-compiler function calls).
When returning from the procedure a message and the return value in register EP is automatically displayed.

4.2.29. UC

User Program Call

The User Program Call works in a similar way to the 'CALL' command with the difference that no target address is specified in the command line.
To use this function, an appropriate JMP instruction must be entered into a specific BIOS table location (see BIOS Interface) when downloading the program.

4.2.30. UR

User Program Reset

The monitor will activate the user program RAM bank and initiate a hardware reset, so the user program will start, beginning its reset vector.

Note: This command is equivalent to pressing the User Reset button.

4.2.31. EXIT

Exit Monitor Shell

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

4.2.32. MR, RES, RESET

Monitor Reset

This command is equivalent to pressing the Monitor Reset button.

4.2.33. *Debug Commands*

4.2.34. B

B [*bNUMBER BREAKADDR* [*OCCCOUNT*] ['*S*'] ['*W*' *wADDR wCNT*]

Set Breakpoint/Snapshot/Watch-area

If no parameters are specified, the Breakpoint command will display the current breakpoints list. If a BreakpointNumber and a BreakpointAddress are specified, a breakpoint is set at the specified program location. The BreakpointNumber (0..9) is maintained as a reference. It will be displayed when using the 'L' 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 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 space.
´W´ must be followed by a start address ´wAddr´ and 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 OccurCounter in this case, program execution will stop when it reaches zero.

NOTES:
- 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.

4.2.35. BC [BNumber]

Clear Breakpoint

If no BreakpointNumber is specified, this command will clear all breakpoints.

4.2.36. BD [*BNUMBER*]

Disable Breakpoint

If no BreakpointNumber is specified, this command will disable all breakpoints.

4.2.37. BE [*BNUMBER*]

Enable Breakpoints

If no BreakpointNumber is specified, this command will enable all breakpoints.

4.2.38. T [*NINSTR*]

Trace Instruction

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

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

The interactive operation works according to the following loop.

LOOP

- the monitor displays the actual register values and prints the disassembled instruction
- then the User is prompted to press the 'SPACE', 'RETURN', 'C' or 'R' 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 before and after executing an instruction are always displayed.

Special key functions:

- **'SPACE'** or **'RETURN'** will execute one instructions at a time.
- If **'C'** is used at a CALL subroutine instruction, the subroutine is executed without tracing.
- If **'R'** is used at a Branch instruction, a taken branch will not be traced. This means, if a branch is taken, the program will continue to execute until reaching the instruction following the branch (branch not taken). This has applications in skipping long polling or timing loops.

Important:

The monitor uses the CALLV #7 instruction and the corresponding vector in the UserRAM to realise the Breakpoint and Trace operation. Thus, this instruction must not be invoked by the user program.

5. 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 located in a reserved area of RAM (below the bankswitch-able ROM / RAM) and has the following structure:

In the following table, entries are termed `function vectors`.

Table Header at 0800h		The first 8 Bytes contain a signature
"BIOS Vxx"	00	Call the following location for :
I/O Function Calls (ASM)	08	BIOS Functions (Parameters in Registers)
I/O Function Calls (C)	0C	BIOS Functions (Parameters on Stack)
Print Register Values	10	Print CPU Registers
Monitor Control	14	Call a Monitor Shell
Monitor Reset	18	Monitor (Software) Reset
Reserved	1C	(Monitor internal use for Trace & Break)
User Program Start	20	This entry can be set by the user program
Pointer to Symbol Table	24	This entry can be set by the user program
	└──	Table Offset

The start address of the table is at location **0800**h. The table header should be checked before calling a table entry.

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

5.1.1. 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.

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

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

a) Function Call (ASM) Register AH : Function code number
AL : Byte Parameter
EP : 2nd Parameter (address pointer)

e.g. :

```
MOVW  A,#(H'0100 | '*')  ; Function Code 01 => AH, '*' => AL
CALL     H'0808          ; Call BIOS function
```

b) Function Call (C) 1st word on stack : HiByte Funct.Code, LoByte Param.
2nd word on stack : 2nd Parameter (address pointer)

e.g. :

```
char (*BiosCall)(int,int) = 0x080C;
BiosCall( (0x0100 | '*'), NULL);
```

On return, a 16 bit value is passed back to the calling routine via the EP register.
Generally, a negative return value is regarded as an error exit code.
Note that as with C-Subroutines, only the registers IX, R2..R7 will be saved, all other registers can have different values.

NOTE: An include file with macro definitions for assembly programs (EBIOS.INC), or a header and library file for C programs (EVABIOS.H, EVABIOS.C) contain the declarations which simplify the usage of I/O functions. These files are also used in the examples described in chapter 5.

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

Function	FCode.	Byte Param.	2nd Parameter	Return Value
Poll Char Received*	00			1:Rec, 0:No
Send Char on RS232	01	Output Char	-	-
Receive Char	02	-	-	Input Char
Print a String	03	-	Adr Pointer	-
Input a String	04	b6..0=SLen b7=CrLf	Adr Pointer	Inp.Termn. Char
Print a HexByte	05	HexByte	-	-
Print a HexWord	06	-	HexWord	-
(reserved)	07			
(reserved)	08			
Enable/Dis. BreakPnt.	09	1:En., 0:Dis.	-	-

* Restricted Functionality

5.1.2. I/O Function Calls Detailed Description

5.1.2.1. [00] Poll Character Received

This function works in a similar way to computer BIOS functions, in checking whether a key was pressed and whether the character is available in a 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 the character will then wait for the next received character.

5.1.2.2. [01] Send Character on RS232

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

5.1.2.3. [02] Receive Character on RS232

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

5.1.2.4. [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”.

5.1.2.5. [04] Input a String

This function reads a complete string from the computer terminal.
The input is terminated when the RETURN or ESC key is pressed, or the max. string length is exceeded.

If specified, a CR-LF sequence will be sent to the terminal to position the cursor onto the next line. The max. string length (bit 0..6) and the CrLf flag (bit 7) are specified by the parameter byte.

5.1.2.6. [05] Print HexByte

This function will convert the byte parameter into a 2-digit hex-ascii number and transmit it to the terminal.

5.1.2.7. [06] Print HexWord

This function will convert the word parameter into a 4-digit hex-ascii number and transmit it to the terminal.

5.1.2.8. [09] Enable/Disable Breakpoints

Depending on the byte parameter, this function can temporarily disable or re-enable all breakpoints set by the operator.

5.2. 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 manually setting a breakpoint.

5.2.1. 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.    CALL    H'0810   ; Print CPU Registers
```

5.2.2. Monitor Control

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

```
e.g.    CALL    H'0814   ; Call Monitor Shell
```

NOTE: Make sure the monitor was initialised (Power On Reset, Mon.Reset) before this entry is called the first time !
NOTE: Using the monitor control function is an alternative to breakpoint setting.

5.2.3. Monitor Reset

This entry can be called by the user program if it wants to terminate. In this case the ROM bank containing the monitor is activated and a Monitor Reset is executed.

5.2.4. User Program Start

The user program must write a 'JMP UserEntry ' into this table entry. (This can be part of the object code which is downloaded .) The monitor command 'UC' can then simply be used to activate and CALL the user program at the UserEntry location.

5.2.5. SymTab Vector

If symbol information is available, a pointer to the symbol table can be loaded into the first word of this table entry during program download.

6. Evaluation Board Hardware

The evaluation board provides various IO-signals which can be connected to external devices. The controller resource functions are available on the connectors JP3, JP4, JP7 and JP5.
External peripheral devices can be connected via the address/data bus on JP6 and JP12 and the select signal /EIO on JP5. For more information please refer to the following pin assignments and the schematics in appendix A.

6.1. Pin Assignment

The F^{2}MC-8L evaluation board PCB was designed such that it can be used for the MB89620, MB89630 and MB89850 series of Fujitsu's 8-bit micro-controllers.
Thus, the MB89T637 can be simply replaced by a T625 or T855 to derive an evaluation board for the device family. The main differences between the board versions are different I/O signals on ports 3,4 and 5 and the associated connectors.

6.1.1. MB89630 Configuration

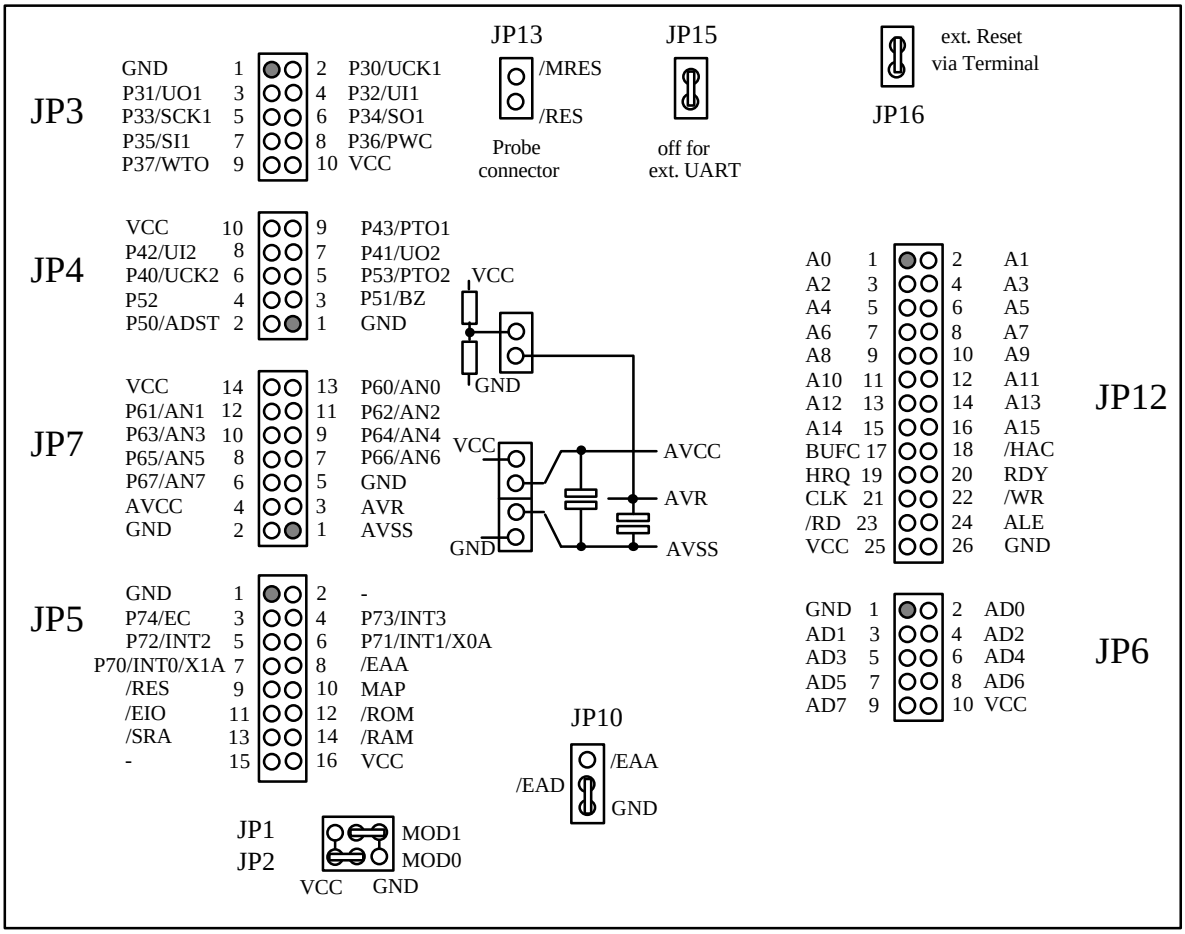


Fig. 7 : MB89630 Evaluation board connector and jumper assignment

6.1.2. MB89620 Configuration

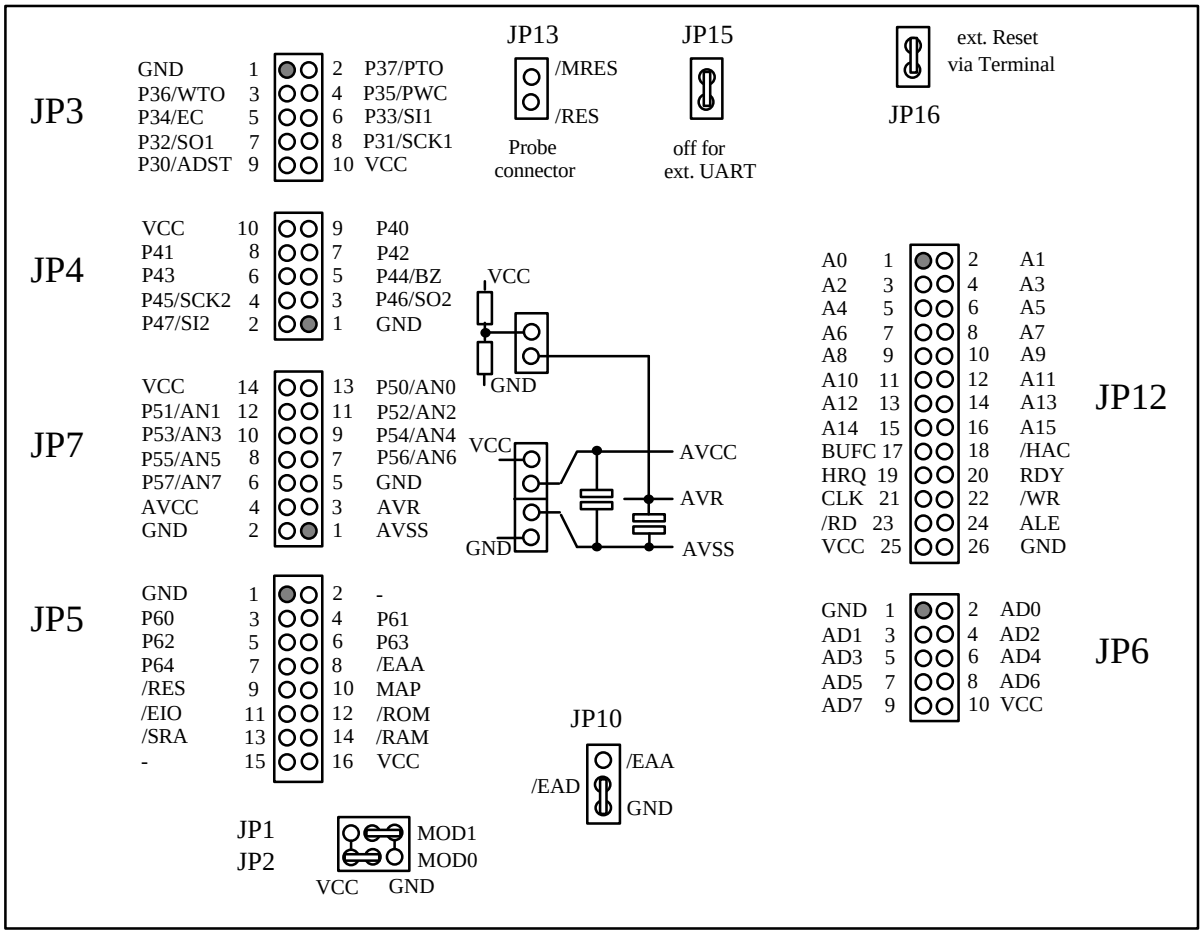


Fig. 8 : MB89620 Evaluation board connector and jumper assignment

6.1.3. MB89855 Configuration

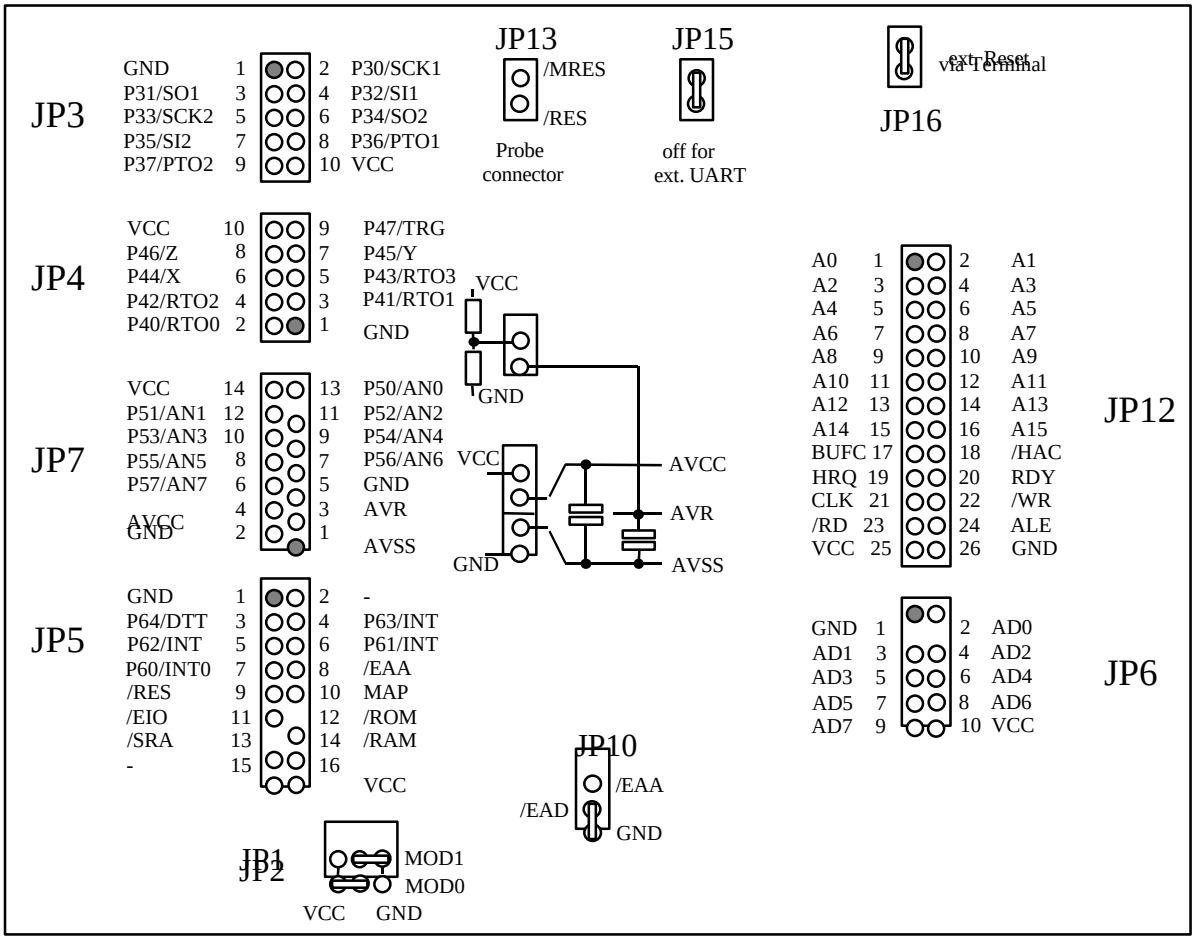


Fig. 9 : MB89850 Evaluation board connector and jumper assignment

NOTE: The MB89620 and MB89850 series doesn't have any sub-clock inputs, so the 32KHz sub-clock crystal which is mounted on the board for the MB89630 configuration could be disconnected from P63 and P64 (see schematics) by cutting the solder-bridges SB1 and SB2 on the solder side of the PCB (closely located to the 32 KHz crystal).

Since the crystal connection to the port inputs should not have much influence on the port functions, we recommend to keep the solder-bridges in for an easy upgrade back to the MB89630 version.

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

Ports P0, P1 and P2 of all micro controllers (MB89T625, MB89T637 and MB89T855) are used as an external microprocessor bus, connecting the Monitor EPROM U1, the User RAM U3 and the Data RAM U4 . Address latch U2 is used to de-multiplex the combined address/data bus.

The PAL or GAL U5 is used as a memory decoder and provides chip select signals for the EPROM, User RAM, Data RAM, Memory mapped control registers in GAL U6 (/SIO) and a select signal (/EIO) which can be used for extensions.
U5 also makes sure that either the EPROM or the User RAM is selected when read from, depending upon a MAP input signal. For write accesses, the User RAM will always be selected.

Memory map:

0000-027F On Chip Area (/RD/WR won't be activated when accessing this area)
0280-0ABF Data RAM
0AC0-0ADF External memory mapped IO (/EIO)
0AE0-0AFF Control Registers (/SIO)
0B00-7FFF Data RAM
8000-FFFF User RAM (MAP=1) or Monitor EPROM (MAP=0)

GAL U6 implements four control registers selected by (/SIO,A0,A1) each one bit wide and read/written using /WR, /RD, and AD0.

Control Register Map:

0AE0 : (WR Only) MAP register selecting EPROM or User RAM
0AE1 : (R/W) Tx, Rx register used to implement RS232 interface
0AE2 : (R/W) Control User LED 1 (0x00=On, 0x01=Off)
0AE3 : (R/W) Control User LED 2 (0x00=On, 0x01=Off)

GAL U6 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 MAP register is either cleared or set.

7. Appendix

7.1. Appendix A: Board Schematics

REPLACED
BY
PICTURE

Chip Pin	JP Pin	MB89T625	MB89T637	MB89T855
1	JP3.3	P36/WT0	P31/UO1	P31/SO1
2	JP3.2	P37/PT0	P30/UCK1	P30/SCK1
3	JP4.9	P40	P43/PTO1	P47/TRG
4	JP4.8	P41	P42/UI2	P46/Z
5	JP4.7	P42	P41/UO2	P45/Y
6	JP4.6	P43	P40/UCK2	P44/X
7	JP4.5	P44/BZ	P53/PTO	P43/RTO3
8	JP4.4	P45/SCK2	P52	P42/RTO2
9	JP4.3	P46/SO2	P51/BZ	P41/RTO1
10	JP4.2	P47/SI2	P50/ADST	P40/RTO0
11	JP7.13	P50/AN0	P60/AN0	P50/AN0
12	JP7.12	P51/AN1	P61/AN1	P51/AN1
13	JP7.11	P52/AN2	P62/AN2	P52/AN2
14	JP7.10	P53/AN3	P63/AN3	P53/AN3
15	JP7.9	P54/AN4	P64/AN4	P54/AN4
16	JP7.8	P55/AN5	P65/AN5	P55/AN5
17	JP7.7	P56/AN6	P66/AN6	P56/AN6
18	JP7.6	P57/AN7	P67/AN7	P57/AN7
19	JP7.4	Avcc	Avcc	Avcc
20	JP7.3	Avr	Avr	Avr
21	JP7.1	Avss	Avss	Avss
22	JP5.3	P60/INT0	P74/EC	P64/DTTI
23	JP5.4	P61/INT1	P73/INT3	P63/INT3
24	JP5.5	P62/INT2	P72/INT2	P62/INT2
25	JP5.6	P63/INT3	P71/INT1	P61/INT1
26	JP5.7	P64	P70/INT0	P60/INT0
27	JP5.9	RST	RST	RST
28	JP2	MOD0	MOD0	MOD0
29	JP1	MOD1	MOD1	MOD1
30		X0	X0	X0
31		X1	X1	X1
32		Vss	Vss	Vss
33	JP12.24	P27/ALE	P27/ALE	P27/ALE
34	JP12.23	P26/RD	P26/RD	P26/RD
35	JP12.22	P25/WR	P25/WR	P25/WR
36	JP12.21	P24/CLK	P24/CLK	P24/CLK
37	JP12.20	P23/RDY	P23/RDY	P23/RDY
38	JP12.19	P22/HRQ	P22/HRQ	P22/HRQ
39	JP12.18	P21/HAK	P21/HAK	P21/HAK
40	JP12.17	P20/BUFC	P20/BUFC	P20/BUFC
41	JP12.16	A15	A15	A15
42	JP12.15	A14	A14	A14
43	JP12.14	A13	A13	A13
44	JP12.13	A12	A12	A12
45	JP12.12	A11	A11	A11
46	JP12.11	A10	A10	A10
47	JP12.10	A9	A9	A9
48	JP12.9	A8	A8	A8
49	JP12.8	A7	A7	A7
50	JP12.7	A6	A6	A6
51	JP12.6	A5	A5	A5
52	JP12.5	A4	A4	A4
53	JP12.4	A3	A3	A3
54	JP12.3	A2	A2	A2
55	JP12.2	A1	A1	A1
56	JP12.1	A0	A0	A0
57		Vss	Vss	Vss
58	JP3.9	P30/ADST	P37/WTO	P37/PTO2
59	JP3.8	P31/SCK1	P36/PWC	P36/PTO1
60	JP3.7	P32/SO1	P35/SI1	P35/SI2
61	JP3.6	P33/SI1	P34/SO1	P34/SO2
62	JP3.5	P34/EC	P33/SCK1	P33/SCK2
63	JP3.4	P35/PWC	P32/UI1	P32/SI1
64		Vcc	Vcc	Vcc

Pin Assigments depending on type of controller

7.2. Appendix B: Example Program Listings

Example Listing "LED8L.C"

```
/*===========================================================================*/
/*                          F U J I T S U                                    */
/*                                                                           */
/*              M i k r o e l e k t r o n i k   G m b H                      */
/*                                                                           */
/*                                                                           */
/*  Filename:       LED8L.C                                                  */
/*  Description:    LEDs flashing on eva-board                               */
/*  Series:         MB89630                                                  */
/*  Version:        V02.00                                                   */
/*  Design:         Markus Mierse '97                                        */
/*===========================================================================*/
                              /* from FETOOL\8L\INCLUDE-directory :    */
#include <TYPEDEFS.H>         /* some usefull type definitions         */
                              /* some example definitions :            */
                              /* "direct"-variables stored directly in */
direct BYTE DirVarByte;       /* lower RAM (DIRVAR) for faster access  */
direct BYTE DirInitByte = 0;  /* initialized direct variable (DIRINIT) */

BYTE VarByte;                 /* standard variables stored in DVAR     */
BYTE InitByte = 0;            /* initialized (DINIT)                   */
const int constdummy = 0;     /* Constant definition                   */
BYTE *LED1 = 0x0AE2;          /* Pointers to the LEDs (memory mapped)  */
BYTE *LED2 = 0x0AE3;

/*===========================================================================*/
/* Prototypes */
void wait(int counter);

/*===========================================================================*/
/* Main Module */

void main (void)
{
  while(1)
  {
     *LED1=1;
     *LED2=0;
     wait(1000);
     *LED1=0;
     *LED2=1;
     wait(1000);
     *LED1=1;
     *LED2=1;
     wait(1000);
     *LED1=0;
     *LED2=0;
     wait(1000);
  }
}
/*===========================================================================*/
void wait(int counter)
{
  for (; counter > 0; counter--);    /* very simple delay loop */
}
```

Example Listing "BIOSDEMO.C"

```
/*====================================================================*/
/*                          F U J I T S U                             */
/*                                                                    */
/*              M i k r o e l e k t r o n i k   G m b H               */
/*                                                                    */
/*                                                                    */
/*  Filename:       BIOSDEMO.C                                        */
/*  Description:    Demostration for evaluation board BIOS-functions  */
/*                  After downloading, start a terminal and execute   */
/*                  program by typing 'g' or push user reset button   */
/*  Version:        V02.00                                            */
/*  Design:         Markus Mierse '97                                 */
/*====================================================================*/

                              /* from FETOOL\8L\INCLUDE-directory :    */
#include <evabios.h>          /* BIOS-fctns; includes also TYPEDEFS.h  */

                              /* some dummy definitions :              */
direct BYTE DirVarByte;       /* to avoid linker warnings (DIRVAR)     */
direct BYTE DirInitByte = 0;  /* initialized direct variable (DIRINIT) */
BYTE VarByte;                 /* standard variables stored in DVAR     */
BYTE InitByte = 0;            /* initialized (DINIT)                   */
const int constdummy = 0;     /* Constant definition                   */
BYTE InpStr[82];
BYTE SumB;
WORD SumW;
                              /* ROM-constant strings            */
CSTR IniMsg[] = "\n"
           "**************************************\n"
           "**  Demonstration of BIOS Functions  **\n"
           "**************************************\n";
CSTR StrMsg[] =   "'puts'-function :\n"
           " This is an output string\n\n"
           "'gets'-function :\n"
           " Please enter your name : ";
CSTR  TyMsg[]=   "\n Thank you, ";
CSTR ChGMsg[]=    "\n\n'getch'-function :\n"
           " Press any key !\n ";
CSTR OcMsg[]=     "\n You pressed ";
CSTR ChPMsg[]=    "\n\n'putch'-function : ";
CSTR HexBMsg[]=   "\n\n The byte-check sum is : ";
CSTR HexWMsg[]=   "\n The word-check sum is : ";
CSTR EndMsg[]= "\n\n That all folks ! Press any key for Monitor Reset";


/*====================================================================*/
void wait(int counter);

/*====================================================================*/
/* Main Module */
void main()
{
 int ix;
 BYTE Ch1, Ch2;

      puts(IniMsg);           /* Print header   */
      puts(StrMsg);           /* Put String     */
      gets(&InpStr);          /* Input String   */
      puts(TyMsg);
      puts(InpStr);           /* print received string */
      puts(ChGMsg);
      Ch1 = getch();          /* Input Charakter */
      puts(OcMsg);
```

```
      putch(Ch1);             /* Output received Char */
      puts(ChPMsg);
        ix = 0;
      while (InpStr[ix])      /* go through string */
      {
        Ch2 = InpStr[ix];     /* type every single char */
        putch(Ch2);
        putch(32);
        wait(3000);           /* slowly...*/
        SumB += Ch2;          /* calculate the checksum */
        SumW += Ch2;
        ix++;                 /* next char */
      }
      puts(HexBMsg);          /* print the checksum */
      printHexByte(SumB);     /* in byte-format */
      puts(HexWMsg);
      printHexWord(SumW);     /* and in word-format */

      puts(EndMsg);           /* final message */
      getch();
      MonitorReset ();        /* monitor-reset the board */
}


/*=========================================================================*/
/* Procedures */

void wait(int counter)
{
  for (; counter > 0; counter--);    /* very simple delay loop */
}
```

Header File "EvaBIOS.h"

```
/*+-------------------------------------------------------------------+*/
/*¦                         F U J I T S U                             ¦*/
/*¦                                                                   ¦*/
/*¦              M i k r o e l e k t r o n i k   G m b H              ¦*/
/*¦                                                                   ¦*/
/*¦  Filename:       EvaBIOS.H                                        ¦*/
/*¦  Description:    Header file for EVA Board I/O Functions          ¦*/
/*¦  Series:         Independent                                      ¦*/
/*¦  Version:        V01.00                                           ¦*/
/*¦  Design:         Edmund Bendels 09.08.94                          ¦*/
/*+-------------------------------------------------------------------+*/

#include <TYPEDEFS.H>

void MonitorReset ();
BYTE getch ();
void putch (BYTE c);
BYTE *gets (BYTE *s);
void puts (const BYTE *s);
void printHexByte (BYTE b);
void printHexWord (WORD w);
```

Library File "EvaBIOS.C"

```
/*+-------------------------------------------------------------------+*/
/*|                          F U J I T S U                            |*/
/*|                                                                   |*/
/*|              M i k r o e l e k t r o n i k   G m b H              |*/
/*|                                                                   |*/
/*|  Filename:       EvaBIOS.C                                        |*/
/*|  Description:    Some functions to access the MB89637 EVA Board   |*/
/*|  Series:         Independent                                      |*/
/*|  Version:        V01.00                                           |*/
/*|  Design:         Jürgen Suppelt 09.03.94                          |*/
/*|                  Edmund Bendels 09.08.94                          |*/
/*+-------------------------------------------------------------------+*/

#include "evabios.h"

char (*BiosCall) (int, int) = 0x080C;
void (*MonReset) (void) = 0x0818;

BYTE *LED1 = 0x0AE2;
BYTE *LED2 = 0x0AE3;

BYTE getch ()
{
  return (BiosCall (0x0200, 0));
}

BYTE *gets (BYTE *s)
{
  BiosCall (0x0400 | 0xD2, s);  /* CrLf after Input, Max 82 Char */
  return (s);
}

void putch (BYTE c)
{
  BiosCall(0x0100 | c, 0);
}

void puts (const BYTE *s)
{
  BiosCall (0x0300, s);
}

void printHexByte (BYTE b)
{
  BiosCall (0x0500 | b, 0);
}

void printHexWord (WORD w)
{
  BiosCall (0x0600, w);
}

void MonitorReset ()
{
        MonReset ();
}
```

Example Listing "NEWPROJ.C"

```
/*====================================================================*/
/*                        F U J I T S U                               */
/*                                                                    */
/*             M i k r o e l e k t r o n i k   G m b H                */
/*                                                                    */
/*                                                                    */
/*  Filename:       NEWPROJ.C                                        */
/*  Description:    Example-project for 8L-evaluation board           */
/*  Series:         MB89630                                           */
/*  Version:        V01.00                                            */
/*  Design:         Markus Mierse '97                                 */
/*====================================================================*/

                            /* from FETOOL\8L\INCLUDE-directory :       */
#include <TYPEDEFS.H>       /* some usefull type definitions     */
#include <MB89630.H>        /* register definitions for 630-family  */
#include <INT89630.H>       /* interrupt definitions for 630-family     */

                            /* some example definitions :       */
                            /* "direct"-variables stored directly in*/
direct BYTE DirVarByte;             /* lower RAM (DIRVAR) for faster access     */
direct BYTE DirInitByte = 0;  /* initialized direct variable (DIRINIT)*/

BYTE VarByte;               /* standard variables stored in DVAR        */
BYTE InitByte = 0;          /* initialized (DINIT)                      */
const int constdummy = 0;   /* Constant definition                      */

BYTE *LED1 = 0x0AE2;        /* Pointers to the LEDs (memory mapped) */
BYTE *LED2 = 0x0AE3;



/*====================================================================*/

/* Prototypes */


/*====================================================================*/

/* Main Module */

void main (void)
{

}


/*====================================================================*/
/* Procedures */



/*====================================================================*/
/* Interrupt service routines */

#pragma interrupt
#pragma save_reg
 void WATCHINT11(){}
 void TBCINT10(){}
 void ADCINT9(){}
 void UART_T_INT8(){}
 void UART_R_INT7(){}
 void TC16INT6() {}
 void PWCINT5(){}
 void PWM2INT4(){}
 void PWM1INT3(){}
 void SIOINT2(){}
 void EXINT1(){}
 void EXINT0(){}
#pragma nointerrupt
#pragma nosave_reg
```

7.3. Appendix C: Software Tools

After the installation, there are software tools in the “FMG_UTIL”-directory, which allow communication and debugging from a DOS or Windows based envronment to the evaluation board.

NOTE : When using Softune, these tools should be already customized in the “UTILIY”-menu. If they cause any problems or should be changed (e.g. due to other locations), use the “OPTION – SET UTILIY”-menu to change settings.

7.3.1. Windows-tools

The HEXLOADW.EXE -program can be used for downloading program or symbol information to the board and for executing the program immediately after loading. To invoke the loader, specify at least the COM-port number and a file (INTEL-HEX-record format “.HEX”). Other options are :

```
HEXLOADW [1..8] -SK8 [-RD] [-R|-C] [-S [-SA:XXXX][-SO]] file.hex

  1..8 COM port number (default is COM1)
  -SK8 Download-mode for evaluation board 8L
  -RD do not support Starterkit (R)eset line function over DTR
  -R  after successfully loading, execute the program
  -C  close window after loading
  -S  create and load symbol file from .mp1 in same directory as file.hex
  -SA: symbol-table address as a four digit hex-number (default:1000h)
  -SO load symbols only (no program code)
```

Detailed knowledge of the download protocol normally is not needed to operate the board. In the case where a special download program needs to be developed, refer to appendix 7.3.

7.3.2. DOS-tools

For the described tasks (downloading, symbol convertion) there are two different programs for DOS, because using batch-files is a common way of handling development tools when not using Windows.

7.3.2.1. SYMBOL-converter

Before symbol information can be used by the monitor (e.g. “G, _MAIN”), the information has to be converted and downloaded to the evaluation board. The program **SYMBOL.EXE** (DOS) generates a symbol table suitable for downloading.

To execute the program, specify the mapping information, generated by the linker (.MP1), an output file (.SYM) which needs to be downloaded after the conversion and an address in memory where the table should be located e.g. 1000_{hex} (normally free RAM). The offset for suppressing a certain range is optional.

```
SYMBOL <InFile> <OutFile> <StartOfHex> [<StartOfSymbols>]

example1: SYMBOL test.mp1 test_s.hex 1000
          locates the HEX-file at 0x1000h with all symbols

example2: SYMBOL test.mp1 test_s.hex 1000 80
          locates the HEX-file at 0x1000h with symbols starting
          from 0x80h (suppress symbols in the range of [0..0x7Fh]
```

7.3.2.2. HEXLOAD for DOS

The download-utility for the DOS-environment is call HEXLOAD.EXE. Usage is similar to the described HEXLOADW.EXE, but with different options :

```
HEXLOAD [1..8] [-N|-NN] [-D] [-R] file.hex

 1..8 COM port number (default is COM1)
  -N  do (N)ot display the record lines, show percentage counter
  -NN do (N)ot display process indicator at all (recomm. for Softune)
  -R  do Not support Starterkit (R)eset line function over DTR
  -D  (D)ebug mode display all received characters
```

7.3.2.3. Terminal for DOS

A very simple terminal program “MT.COM” with default communication and emulation settings for the evaluation board is provided. This DOS-program uses the assigned COM-port (use the mode-command to assign a COM-port - e.g. “mode com1: 9600,n,8,2”).

7.3.3. Download Protocol

Three different formats can be used to download data to the evaluation board :

1.) Object code hex-lines (as produced by the linker *.hex file extension)

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 checksum are not checked ! Note that a download program using this mechanism must read each character's echo and that the completing CR is echoed as CR LF.

2.) HexFile Download function - works similar as the previous command, however :

a) download function is activated by sending the command 'HEXDWL↵
b) Note that the characters of the command are echoed, the CR is not !
c) a hex-line is transferred (but no character is echoed !)
d) when the monitor has processed the hex-line, it will send the acknowledge character (06H).
e) then the next hex-line can be transferred.
f) The hex download function terminates automatically if a hex-line containing an 'end' mark (hex-line with zero byte count) was downloaded.
g) A checksum is built over each line and the monitor will respond with a not-acknowledge character (05H) if an error occurs.

3.) The **High Speed Download** function is also activated by a special command ('HSDWL').
(Note that the characters of the command are echoed, the CR is not !)
In the following sequence, addresses, total byte counts and the data bytes are transferred as **binary** characters and are not echoed. The protocol can be illustrated as :

PC Download Program	**Monitor Operation**
a) send 'HSDWL↵'	echo each character (not the CR)
b)	send the Request Character 0xEB
c) send the Record Header 0xBE or a Termination 0DH char to exit the download function	
d) send Address Bytes (Hi,Lo) send Byte Count (Hi,Lo)	
e) send Data Bytes	(the monitor will store the data)
f) continue at b.	(at the specified locations)

7.4. Appendix D: Monitor Software Notes, Restrictions

As mentioned in chapter 2, the Monitor software resides in an EPROM memory bank in the upper address area 8000h-FFFFh . It is transparent to the upper User RAM bank. In addition, some User RAM locations are reserved for monitor variables.

Monitor reserved RAM sections are:
a) A small section in the lower User RAM area (0800h-0FFFh).
This section is used for some special RAM code procedures which are copied from the EPROM during the monitor initialisation.
It also contains the BIOS interface table, monitor variables, and the off-chip memory mapped IO registers.
b) The vector table entry for the CALLV #7 instruction, (UserRAM Address: FFCE-FFCF) which is initialised with a vector to a special RAM code section, supporting breakpoints and single step.

Apart from these reserved RAM areas, the user can utilise the on-chip/off-chip RAM and on-chip peripherals for his programming and evaluation requirements.

Other restrictions:

1.) Monitor BIOS Interface
When using BIOS interface functions (Chap.4,P. 23 and following), the processor stack area must be in the address range 0000h..1FFFh excluding the reserved area 0800h..0FFFh (Chap.7.4, P 56). Thus, the stack must be within 0000h..07FF or 1000..1FFFh.

2.) 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 Use constructs like:

```
e.g.  LABEL1:   BBC 10:3,LABEL1        LABEL:   NOP
                                                BBC 10:3,LABEL
              BNC FOLLA
e.g.  FOLLA:    xxx yyy,zzz                     BNC FOLLA
                                        FOLLA:  xxx yyy,zzz
```

Return from Interrupt : do not single step the RETI instruction !

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 micro controller into the STOP or SLEEP modes will disable monitor control via the terminal.
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.

7.5. Appendix E: List of development tools

MB89620 family

- Emulator for the F^{2}MC-8L Family Emulation Unit MSE1001C
- Emulator for the F^{2}MC-8L Family Power Supply PS5V1_2A0
- Evaluation Chip MB89PV620C-SH
- Piggyback-EPROM MBM27C256A-20CZ (or compatible)
- Probe cable MSE2144-201 (optional)
- OTP MB89P625/7/9P-SH
- Programmable MCU MB89W625/7C-SH
- OTP Programming Adapter ROM-64SD-28DP-8L

MB89630 family

- Emulator for the F^{2}MC-8L Family Emulation Unit MSE1001C
- Emulator for the F^{2}MC-8L Family Power Supply PS5V1_2A0
- Evaluation Chip MB89PV630C-SH
- Piggyback-EPROM MBM27C256A-20CZ (or compatible)
- Probe cable MSE2144-201 (optional)
- OTP MB89P637P-SH
- Programmable MCU MB89W637C-SH
- OTP Programming Adapter ROM-64SD-28DP-8L

MB89850 family

- Emulator for the F^{2}MC-8L Family Emulation Unit MSE1001C
- Emulator for the F^{2}MC-8L Family Power Supply PS5V1_2A0
- Probe header MB2144-212-01A
- OTP Adapter MSE1036
- OTP MB89P857P-SH
- Programmable MCU MB90W857C-SH
- OTP Programming Adapter ROM-64SD-28DP-8L

8. Index