MARC4
4-Bit Microcontroller
User's Guide

1996

TEMIC
Semiconductors

Contents

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

1 Introduction

TEMIC Semiconductors

TEMIC is the microelectronics enterprise of Daimler-Benz. TEMIC's Semiconductor division is a leading manufacturer of application-specific, value-adding integrated circuits for communication equipment, automotive and industrial systems, computers and broadcast media. Discrete semiconductors and optoelectronic devices make the product range complete.

With a technology portfolio which includes bipolar, BiCMOS, GaAs, CMOS and DMOS processes, TEMIC Semiconductors provides a unique set of components and solutions.

TEMIC - a Microcontroller Specialist

TEMIC has been a technology leader in applications requiring minimum current consumption such as watches and clocks for twenty years , and has ten year's experience in the design of low-power microcontrollers. TEMIC offers 4-bit, 8-bit, extended 8-bit and 32-bit controllers. Our MARC4 products are highly sophisticated and have a firm standing as they have been adapted to ten different technologies up to now.

Choosing TEMIC as a partner means, you will have one independent source for components - transistors, diodes, optoelectronic devices including LEDs and IrDA components, integrated circuits and smart-power devices. Due to our state-of-the-art facilities worldwide, TEMIC's production resources are more than sufficient. TEMIC guarantees excellent application support which will reduce your time-to-market. The available software library for programming as well as the detailed documentation (see appendix) are all free of charge.

The History of MARC4

TEMIC Semiconductors started developing the MARC4 in 1986, based on experience with the former 4-µm CMOS core e3101. The aim was to design an easy-to-use, high-performance, 4-bit controller by selecting a high-level language for programming and to provide highly advanced and efficient development tools. Special effort was spent to realize a modular concept with a very small core design.

After developing MARC4 products in 3-µm and even 1.5-µm technologies, TEMIC started working with external foundries in 1989. Since 1993, the MARC4 family has been based completely on external foundries using 2-µm down to 0.6-µm technologies (volatile/non-volatile).

The MARC4 Family

TEMIC offers a complete family of cost-effective, single-chip CMOS microcontrollers, based on a 4-bit CPU core designed for 1.5-, 3- and 5-V applications. The modular MARC4 architecture is HARVARD-like, high-level language oriented and best designed to realize high-integrated microcontrollers with a variety of application- or customer-specific, on-chip peripheral combinations. The MARC4 controller's low voltage and low power consumption is perfect for hand-held and battery-operated applications.

The standard members of the MARC4 family have selected peripheral combinations for a broad range of applications.

Programming is supported by an easy-to-use, PC-based software development system with a high-level language qFORTH compiler and an emulator board. The stack-oriented microcontroller concept enables the qFORTH compiler to generate compact and efficient MARC4 program codes.

Applications

The very small 4-bit core combined with a versatile peripheral cell library enables the design of application-specific microcontrollers.

- 32-kHz subclock
- A/D converter
- Comparator
- EEPROM
- External interrupts
- High-current ports
- LCD drivers
- Low battery detection
- Power-on reset
- Prescaler
- Programmable I/Os
- Reset input
- Serial I/O
- Timers/counters
- Various system oscillators
- Watchdog timer

Features

- Very small 4-bit core combined with versatile peripheral cell library
- Various on-chip peripheral combinations available
- HARVARD structure - 3 parallel-operating buses (pipelining) enhance computing power (2 clock cycles per instruction only)
- 72 RISC-like, 8-bit instructions
- Stack architecture offers customized stack size and 'unlimited' subroutine nesting
- Unique 8 level interrupt controller leads to a very short (3 cycle) interrupt response time
- 'Brown-out' function and internal **P**ower-**O**n-**R**eset (POR) make external components unnecessary
- Small 4-bit periphery bus offers extraordinary flexibility
- 256 4-bit of RAM directly addressable
- Up to 9 KBytes of ROM
- Low-voltage operating range
- Low power consumption
- Hardware optimized to fit in with high-level language qFORTH
- Programming and debugging is supported by an integrated software development system

Programming in the high-level language qFORTH is simple, easy to understand and advantageous. From the hardware side, the expression and return stack have a user-programmable size. The qFORTH instructions correspond directly to the machine words and therefore, the program executes fast. The software code is compact and the sub-routine nesting is almost unlimited. In addition, programming is easy and safe due to the possibility of combining existing software modules.

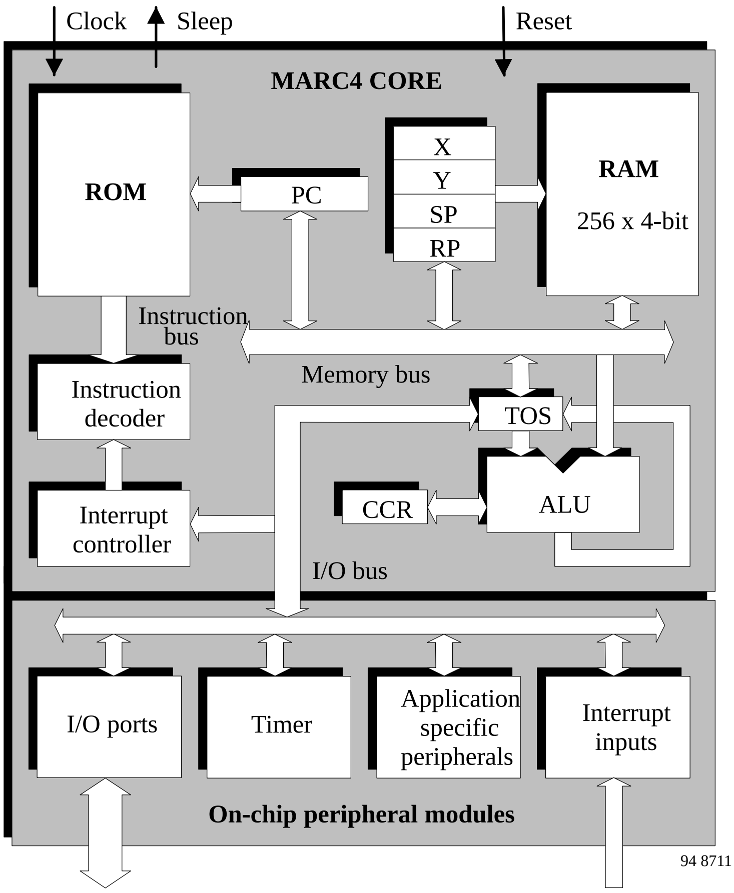
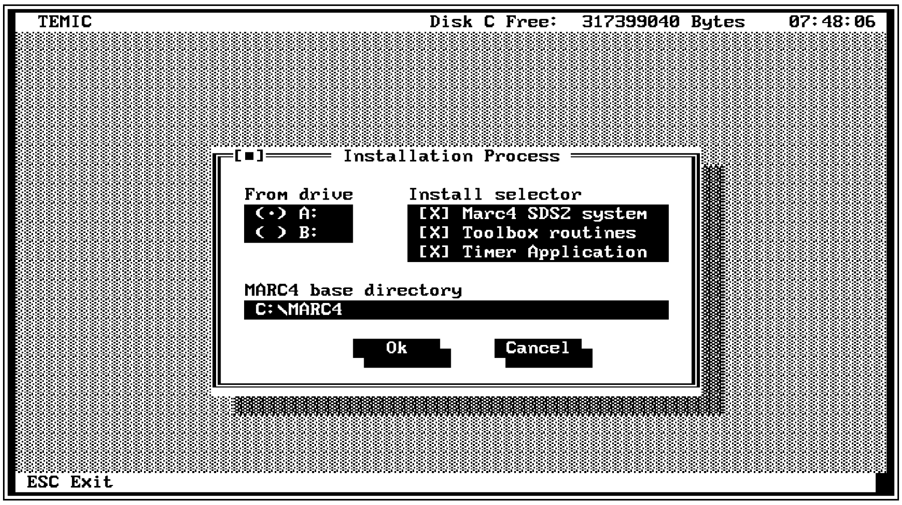


Figure 1. MARC4 core

2 Installation

Important

Please read the International Software and Hardware License Agreement at the last page of this book.



12513

Figure 1. Installation program

2.1 System Requirements

- IBM-compatible PC with 80386 or compatible CPU or greater
- MS-DOS operating system, Release 5.0 or above
- 4 MByte RAM
- 540 kByte of available system memory
- 2 MByte of hard disk space
- A free full–sized ISA bus slot
- A free parallel port (Centronics or EPP compatible) for OTP programmer

The development software runs in text mode and is therefore independant of the installed video adapter, however, a color monitor is recommended.

2.2 Hard Disk Installation

- Insert the SDS2 installation disk and change to this drive (e.g. A: or B:)
- Run the INSTALL program (see figure 1)
- Select the source drive of the installation program
- Use the install selector to choose modules you want to install (default setting: complete installation)
- Enter the name of base directory in the input line (default setting: C:\MARC4)
- By including the MARC4 directory in the AUTOEXEC.BAT search path, it is possible to invoke the MARC4 software development system "SDS2" from any subdirectory.
- Make sure that your CONFIG.SYS contains the following minimum settings:
  Buffers = 20
  Files = 20
- Change to the new MARC4 directory and start the SDS2 program. Use the pull-down menu Options and select Directories to set the search path for the compiler, simulator and emulator program (see figure 2). Please note that you have to type the directory path including the backslash delimiter (e.g., C:\MARC4\).

12514

Figure 2. Set SDS2 directories

2.3 Listing Directories and Files

The MARC4 software development system consists of the following files and subdirectorie to enable complete installation:

Software development system:

SDS2.COM	Start-up program
MARC4SDS.EXE	SDS2 user interface
MARC4SDS.DSK	Desktop configuration file generated on exiting the program

qForth compiler:

qFORTH2.EXE	qFORTH compiler, Release 2.10
qFORTH2.OVR	Overlay file of compiler
qFORTH2.LIB	qFORTH system library
qFORTH2.MSG	Error and warning messages

Software simulator:

SIM05.EXE	Software simulator
SIM05.DAT	Prescaler / interval timer implementation set-up
SIM05.HLP	On-line help file for simulator
SIM05.CFG	Simulator configuration file

Emulator software:

EMU3.EXE	Emulator control program
EMU3.CFG	Emulator configuration file
EMU3.HLP	On-line help file for emulator
EMU4.HEX	RAM dump utility program
M4XCxxx.CFG	Port display configuration setup

OTP programmer:

MARC4OTP.EXE	Startup program
MARC4OTP.TVR	Resource files
MARC4OTP.DEV	MARC4 files for describing the module
MARC4OTP.INI	Configuration file

Utility programs:

INTELHEX.EXE	Binary to Intel-Hex conversion program
UNARJ.EXE	De-archive / De-compression program
KUNDEOPT.EXE	Mask options ordering program
OPTHELP.HLP	On-line help file for mask ordering

Subdirectories:

TIMER	Switch timer software module
TOOLS	Test & demo routines

For your further convenience, it is possible to install your own editor, the qForth2 compiler, the MARC4 software simulator and the emulator control program in PCSHELL (i.e. PCTOOLS V6.0 or higher) or any other shell as executable programs.

2.4 Emulator Installation

To install the MARC4 emulator board, first switch off your PC's power supply.

Insert the plug-in card into a full-sized (AT) bus slot (see figure 4).

Table 1. Emulator card address selector

Card address	JP1 – A7	JP2 – A6	JP3 – A5	JP4 – A4
300h – 30Fh	GND	GND	GND	GND
310h – 31Fh	GND	GND	GND	V_{CC}
330h – 33Fh	GND	GND	V_{CC}	V_{CC}
340h – 34Fh	GND	V_{CC}	GND	GND
350h – 35Fh	GND	V_{CC}	GND	V_{CC}
360h – 36Fh	GND	V_{Cc}	V_{CC}	GND
390h – 39Fh	V_{CC}	GND	GND	V_{CC}

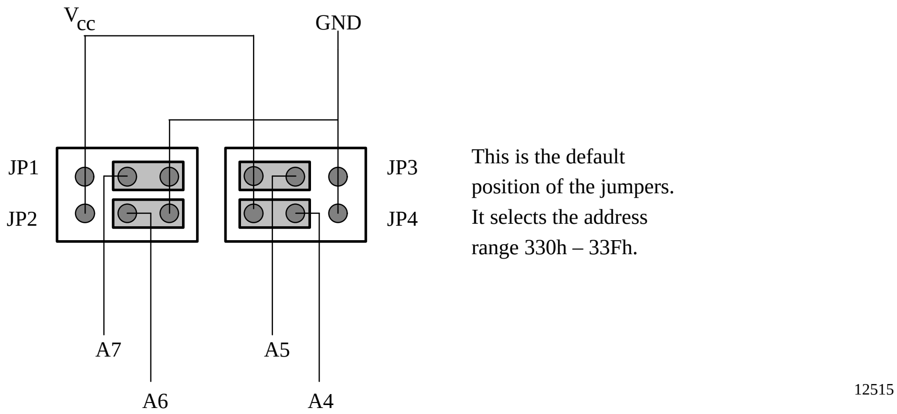


Figure 3. Card address select – default setting

The emulator card address is set to 330h by default and none of the interrupts is used up by this board. If the card address has to be changed, the control software will search automatically for the new card address (table 1). The search sequence is 330h, 310h, 300h, 340h, 360h and 390h. Please make sure that your Ethernet controller card is not in the address range below the MARC4 emulator board.

The location of the most important devices and the address jumpers can be found in figure 4. Figure 3 explains how to set a different card address.

After having installed the emulator card, attach the emulator cable to the DB37 (figure 4) connector. The other end of the cable is plugged onto the MARC4 Target Application Board (TAB). The signal assignment on the DB37 emulator interface connector is described in the chapter 'Target Application Board'.

Note: If you need to run the emulation at frequencies of more than 2 MHz at 5 V, you should connect the target application interface board directly to the emulator board without using the interface cable.

2.5 Operating System Support

OS/2

The MARC4 – Software Development System is supported by the OS/2 DOS emulation and it is therefore possible to run in as either a DOS fullscreen session or DOS window session.

Note: If the MARC4 simulation or emulation program is running in the background, it is possible that the program execution time display will be incorrect.

Windows 3.x or Windows for Workgroups

If you want to start the MARC4 development system under Windows and your PC is integrated in a network system, it would be possible that the installed network drivers require too much of free system memory. If this is the case, your simulation and emulation program will not be able to start.

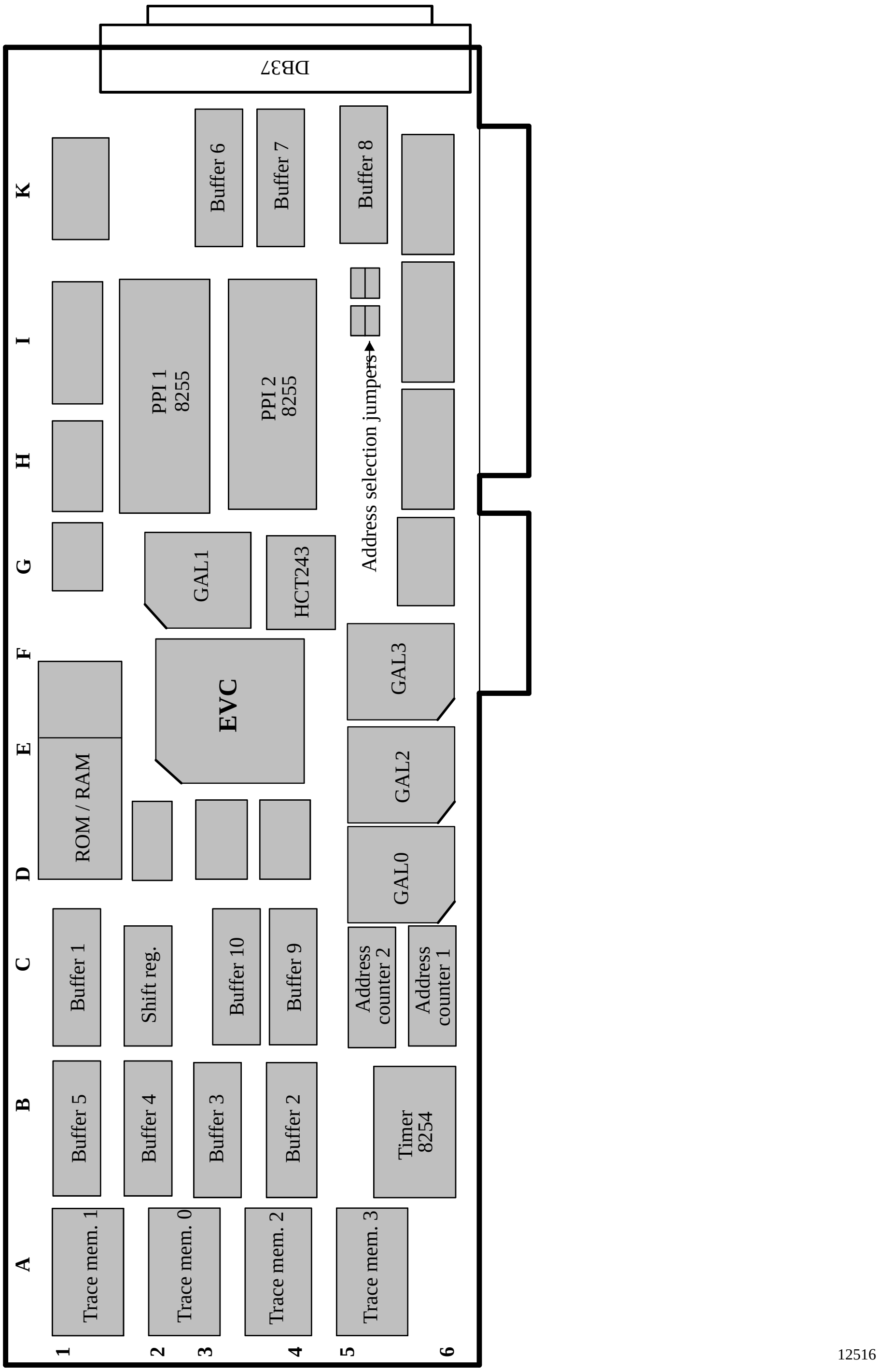


Figure 4. The emulator board

3 Software Development System

3.1 Introduction

The qFORTH compiler, MARC4 simulator, MARC4 emulator and OTP programmer are integrated in a comfortable development environment which includes an editor with functions to load and save text files and many other features. The look and feel of the SDS2 is similar to the Borland Pascal integrated development environment (IDE). The menu and mouse-supported user interface enables all required functions to operate easily and provides user-friendly programming.

Getting started:

By including the MARC4 directory in the AUTOEXEC.BAT search path, it is possible to start the SDS2–IDE from any subdirectory. If this is not the case, you have to change to the MARC4 system directory.

Enter the following command in your DOS PROMPT.

C:\>SDS2

SDS Features :

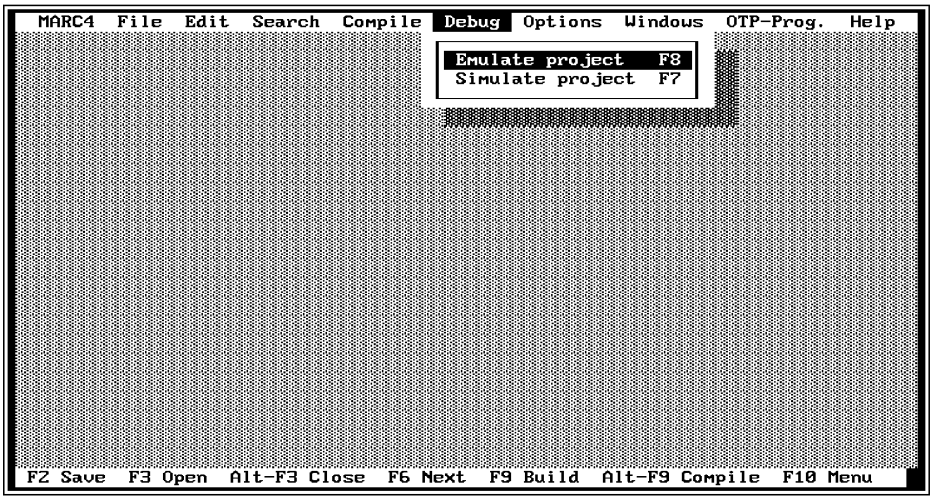
- All integrated tools are mouse-supported
- Editor
  - Multi-window editor
  - Clipboard operations: cut, copy and paste
  - Search and replace functions
- Integrated qFORTH compiler
- Integrated debugger
  - MARC4 Simulator
  - MARC4 Emulator
- Integrated OTP programmer
- Options for environment set-up
  - Setting of SDS2 directories
  - Setting of compiler switches

3.2 User Interface

The menu line appears at the top of the screen in all SDS2 commands embedded in pull-down menus. The field at the bottom of the screen describes the SDS function keys. All grey shaded function keys are either not available or inactive in the current application window.

3.2.1 Pull-down Menus

Pull-down menus can be opened by using the keyboard or the mouse. A letter in the name of each pull-down menu is highlighted. This letter can be used in combination with the <Alt>–key to open the desired menu.



12517

Figure 1. SDS2 user display

3.3 Menu Line and Menu Commands

Menu	Menu Command	**Function Key**	Description
MARC4			
	About SDS	———	Information about release and copyright
	Define mask	———	Defines customer mask options for ordering
	Exit to DOS	**Alt–X**	Exits SDS
File			
	Open	**F3**	Opens text file
	New	———	Makes a new text file
	Save	**F2**	Saves current active text file
	Save as	———	Saves text file with file and path name
	Save all	———	Saves all text files
	Quit	**Alt–Q**	Closes current window
	Change dir	———	Changes the directory
	DOS shell	———	Returns to DOS whithout exiting SDS
Edit			
	Undo	———	Cancels the last procedure
	Cut	**Shift–Del**	Cuts and copies text string to clipboard
	Paste	**Shift–Ins**	Inserts text string from clipboard
	Show clipboard	———	Shows clipboard
	Clear	**Ctrl–Del**	Deletes text string
Search			
	Find	**Crtl–QF**	Finds a text string
	Replace	**Crtl–QA**	Replaces a text string
	Search again	**Crtl–L**	Repeats the search function
Compile			
	Current file	**Alt–F9**	Compiles the current file
	Built project	**F9**	Compiles the project file
	Set project file	———	Sets name and path of project file
Debug			
	Emulate project	**F8**	Starts emulator program with project file
	Simulate project	**F7**	Starts simulator program with project file
Options			
	Directories	———	Installation settings about the directories
	Compiler	———	Compiler switch setup
	Save desktop	———	Saves current desktop settings
	Retrieve desktop	———	Replaces with stored desktop settings
Windows			
	Size/move	**Ctrl–F5**	Chooses and moves window
	Zoom	**F5**	Changes the window size
	Tile	———	Windows position: side by side
	Cascade	**F4**	Windows position: overlayed
	Next	**F6**	Changes to next window
	Previous	**Shift–F6**	Changes to previous window and activates it
	Close	**Alt–F3**	Closes current window
	Calculator	———	Calculator
OTP–Prog.	OTP–Prog.	———	Starts OTP programmer
Help	Help	**F1**	Short program description

4 qFORTH Compiler

By using the MARC4 qFORTH compiler, embedded-system designers no longer have to stick to assembly language; the compiler generates a highly optimized object code. The smart qFORTH compiler translates your high-level qFORTH program into the MARC4 processors native code. The compiler system selects the right assembly-language instruction sequences and addressing modes for optimal operation. The intermediate code passes through rule-based expert systems at different optimization stages. This code is optimized for local centers of reference (colon definitions, macros, loops) to minimize stack operations and register references; it actually scoreboards register references to eliminate redundancies.

The qFORTH compiler supports standard FORTH constructs such as: BEGIN .. AGAIN, BEGIN .. UNTIL, CASE .. ENDCASE, DO .. LOOP, IF .. THEN, IF .. ELSE .. THEN, BEGIN .. WHILE .. REPEAT, and the following 4-bit and 8-bit data types: constants, variables, arrays and 8-bit ROM look-up tables. qFORTH extensions are interrupt functions, direct I/O port access, in-line assembly language and direct register access. The compiler also generates a line-number reference file to support source-code debugging in the MARC4 simulator and emulator.

The compiler is available in two versions. The fully integrated version is run by selecting 'Compile' in the menu bar of the MARC4 integrated environment menu and the command-line version is run by typing QFORTH2, followed by options and the name of the file to be compiled at the DOS command line.

4.1 The qFORTH Program Structure

In order to compile your qFORTH program correctly the compiler expects that the program to be composed of directives, definitions and statements. Most qFORTH programs will contain at least a group of statements which will perform computational operations. These statments are edited according to the guidelines outlined in the qFORTH Programmer's Guide. Whether or not you add compiler directives and CONSTANT definitions is dependant on the requirements of your program. They are more or less optional when compiling a qFORTH program. Parameters are expected by the compiler, but not defined by the programmer. The compiler will substitute default values such as for stack size allocation.

At the end of this chapter you will find a section which lists the default values used by the qFORTH compiler. But first it is necessary to re-examine what the three sections are which make up a qFORTH program.

4.1.1 Compiler Directives

The directives are compiler switches used to control the way in which your program is compiled and to specify the format of your compiler generated file(s). The majority of the directives can be implemented as in-line commands appearing at the beginning of your program code.

4.1.2 Definitions

The CONSTANT and VARIABLE definitions which are values referenced by your program via names. They should be assigned before the CONSTANT or VARIABLE is referenced within the program.

4.1.3 Statements

A qFORTH program is composed of various statements grouped together to perform a particular task which your program invokes via a word. These words are called **colon definitions** because they appear in your qFORTH program as starting with a colon ('**:**'), followed by a space and the name assigned to these group of statements. A statement group is a sequential list of MARC4 instructions, words found in the qFORTH system library or words which have been defined in your program before invoking this subroutine.

Note: All colon-definitions end with a semi-colon ('**;**').

Sequences of functionally grouped words are called modules. Modules used to perform the underlying computational tasks of the MARC4 are often caused by from interrupt service routines. These are predefined names according to the naming conventions described in the qFORTH Programmer's Guide and are identified by '**: INT<x>**', whereby <x> is replaced by the priority number 0 to 7.

The program entry point is identified as the **$RESET** service routine since it is the first word which the MARC4 processor will execute after power-on reset. Normally, this colon definition is located at the end of your source program and consists of two parts: the register and the application initialization section. After the initialization of the stack pointers, the on-chip peripherals and the RAM variables of the application have to be put in a well-defined state.

4.1.4 Kicking the Assembler Habit

This short description has been intended as an overview to program composition as required by the qFORTH compiler.

To achieve a tighter code with your high-level language, remember the following rules and apply them more or less in order.

- Rethink your approach to problems to see if you can’t find a more elegant solution.
- Make sure you are storing and manipulating your data efficiently. Accessing data using a pointer requires almost three times the number of instructions required to access the same information using array indexing.
- Make your code less abstract and take advantage of hardware-specific shortcuts wherever possible, always weighing the tradeoffs between speed and development time.
- Optimize your algorithms, eliminating all redundant and unnecessary operations. Use the address activity profiler in the emulator or simulator and optimize where it will do the most good.
- To maximize the limited on-chip RAM, minimize the usage of local variables and too much nested subroutine calls.
- To reduce the stack usage, check your parameter passing and subroutine nesting as well as the number of concurrent interrupt service routines.
- Use assembler instructions for the time-critical code but do not fall back on writing whole modules in assembler.

Stick to these approaches and you will be writing applications that will keep your competition awake at night, not you.

4.2 Using the Compiler

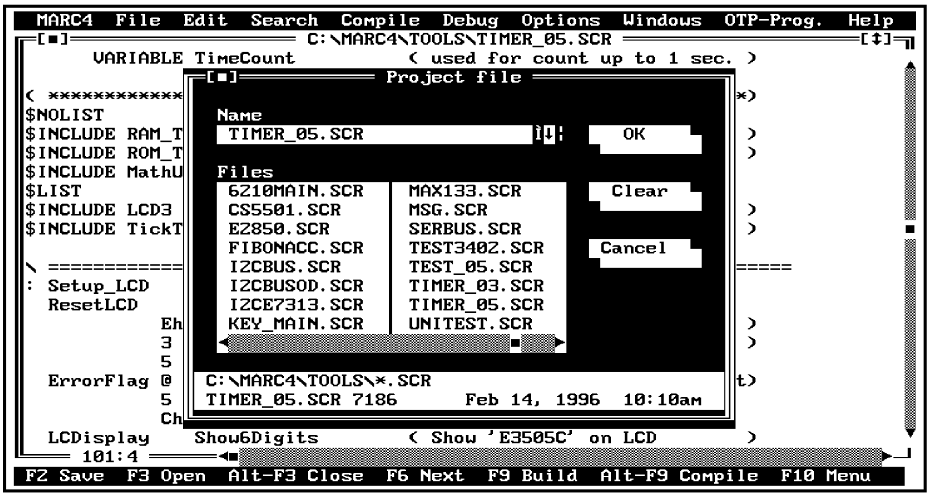
Check that the correct directory path for qFORTH has been entered in the setting window ’Directories’ (see installation guide).

- Edit your program file(s)
- Setup the project’s file name
- Setup the compiler options
- Invoke the compiler

To set the project’s filename use the pull-down menu ’Compile’ and select ’Set project file’. The project’s filename means the leading filename of the project which will be compiled (see figure 1).

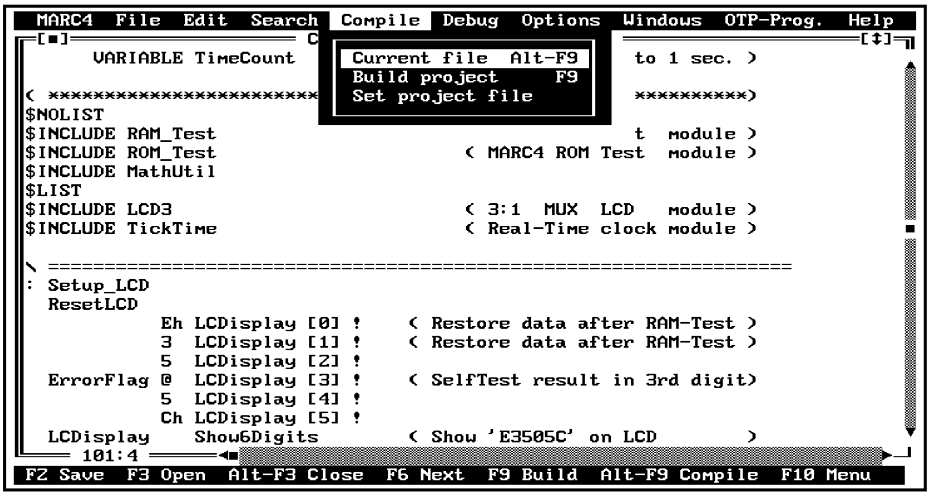
To invoke the compiler, use the pull-down menu ’Compile’ and select ’Built Project’ or press the key **<F9>**. This occurence will compile the whole project.

The ’Compile’ pull-down menu is shown in figure 2. If you wish to compile the currently edited file then either press **<Alt-C>** followed by the carriage return key or simply enter **<Alt-F9>** from within the editor. This will automatically start the compiler using the active file as its input filename.



12518

Figure 1. Set-up of project file to be selected first



12519

Figure 2. Selection of the compiler within the environment

4.2.1 Compiler Generated Messages

If the compiler detects a code which can not compile correctly, a warning or an error message will be displayed. The occurence of a warning is an indication to you that your program will still compile and is executable, however, it may not produce a code with the desired kind of execution. If an error is found, the compiler will terminate since it is unable to generate executable code. A complete list of all warning and error messages can be found in the Appendix.

The information given during any compilation is the following:

- The qFORTH compiler version and the date of creation with the qFORTH system library used with their date of creation
- The name, drive and directory path of the compiled source file
- The optimizer passes, because a '.' is written to the screen for each step during optimization and a ',' when macro expansion takes place.
- The compilation result:

  If no errors were found, the amount of ROM (in bytes) required and the calculated CRC check-sum stored in the last two bytes of the ROM is displayed.

  If errors occur during the compilation, the error and/or warning messages will be reported instead. They will be attached at the end of your source code within the list file.

Note: A complete list of all warnings and error messages can be found in chapter 4.6 "Error and Warning Messages".

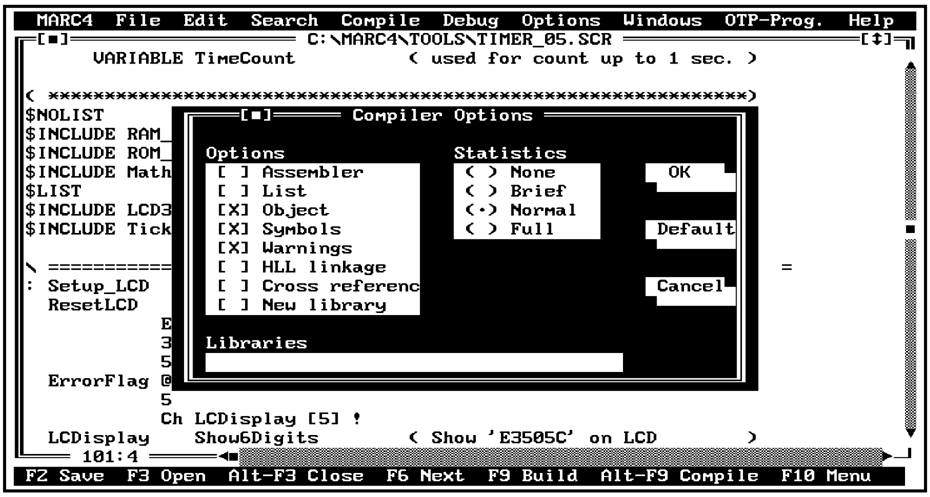
4.2.2 Compiler Generated Files

The compiler generates various files which are normally directed to the same filename, drive and directory path as the project's source file (see table 1).

Table 1. List of all compiler generated files

Extension	File Type & Contents	Format
HEX	Object code	Binary
SYM	Symbol table	Internal
LST	Complete list and statistics	Text
CRF	Cross reference file	Text
ASS	Assembly code list file of compiler generated object code	Text
HLL	Line number reference file for high level language orientated debugging	Internal
LIB	User generated library for often used routines	Internal
RPT	Compilation success/error report file within SDS	Internal

4.2.3 Compiler Switches



12520

Figure 3. Default setting for compiler switches within the option menu

To set the compiler switches for different compiler options, use the pull-down menu ”Options” and select ”Compiler” within the SDS2 environment.

Compiler Options

Assembler

This controls, whether an assembler list file is be generated. The default extension is “**ASS**”, the default filename is that of the source file. This output file may be used to check the efficiency of the generated object code.

List

This controls, whether a source listing has to be generated. The default extension is “**LST**”, the default filename and path is that of the source file. This generated file contains all events during compilation, depending on additional compiler switches.

Object **Default setting**

This controls whether a binary object code file has to be generated. The default ectension is “**HEX**” and default filename is that of the source file. By default, an object and symbol file with full optimization is created.

Symbols **Default setting**

This controls whether a symbol file has to be generated. The default extension is “**SYM**” and the default filename is that of the source code. This file is necessary if you want to check your code with all defined symbols (subroutines and variables). By pressing the function key **<F7>** at the software simulator or emulator, you can view the symbol table data.

Warnings **Default setting**

This controls, whether warnings are written onto the screen and with the setting of the additional switch “**List**” into the list file too.

HLL linkage

This controls whether a high-level-language debugger link file has to be generated. The default extension is “**HLL**”, the default filename and path is that of the source file. This generated file enables source level debugging (see chapter 5 ”Software Simulator”).

Cross reference

This controls, whether a cross reference file has to be generated. The default extension is **“CRF**”, the default filename and path is that of the source file. The cross reference file shows the correlations of all used symbols (subroutines, variables and constants) with regard to their definition and their use for different source files.

New Library

This controls whether a new user library has to be generated. The default extension is **“LIB”**, the default filename is that of the first source file. When a user library is generated, no object, symbol and assemblerfile will be created. A user library will contains all code generated during this compilation or all code read in form in other user libraries (see input line “LIBRARIES”).

Compiler Statistics

None

By setting this switch, all statistical information is suppressed in the list file.

Brief

Generates a summary of all errors, all defined words and all defined variables at the end of the list file.

Normal **Default setting**

Additionally lists the return and expression stack usage of all routines, the addresses of all words placed in ROM, a summary of left ROM holes, unused RAM nibbles and unused short call address entries, an overview of bytes saved during the optimization steps and information about the compiler's memory usage.

Full

Additional information about the subroutine placement algorithm, the CPU time for the different compilation steps, statistics on the usage of the internal symbol table data base, summary of created files and used compiler switch settings.

Libraries

This input line controls whether one or more user libraries have to be read after the system library has been read. By default, no additional user library is read. The list may consist of up to 7 user libraries, their names must be separated by a comma. The default extension is "**LIB**".

4.3 Compiler Directives

A compiler directive may occur anywhere in the source file(s), the first character of a compiler directive is always an "$". In general, a directive is used to control the compilers behavior when processing the source code. Compiler directives can not be abbreviated.

4.3.1 Conditional Compilation

To make your job easier, qForth offers conditional compilation. This means that you can decide what portions of your program to compile based on defined symbols.

The conditional directives are similar in format to the compiler directives you are accustomed to. In other words, they have the format.

$directive <arg>

Where **directive** is the directive (such as **DEFINE**, **IFDEF**, and so on), and **<arg>** is the argument, if any.

Note: There must be a blank as seperator between directive and **<arg>** .

List of conditional compilation directives:

$DEFINE <symbol>	Defines symbol for other directives

To define a symbol, insert this directive into your program. **<symbol>** follows the usual rules for identifiers as far as length, characters allowed, and other specifications are concerned.

Example: **$DEFINE Debug**

This defines the symbol 'Debug' used for the remainder of your program which is to be compiled.

$IFDEF <symbol>	Compiles the following code if **<symbol>** is defined
$ELSE	Compiles the following code if the previous **$IFDEF** is not true, i.e., the **<symbol>** is not defined.
$ENDIF	Marks the end of **$IFDEF** and/or **$ELSE** section.

Example:
```
$IFDEF    <symbol>
          <source code A>
$ELSE
          <source code B>
$ENDIF
```

Where **$IFDEF** is followed by the appropriate argument, and **<source code>** is any amount of qFORTH statements. If the **<symbol>** is not defined, the **<source code A>** is ignored as if it had been commented out of your program.

Within a skipped conditional block only **$IFDEF**, **$ELSE** and **$ENDIF** are processed. All other words (including directives) are ignored. Skipped conditional blocks are marked with a hash sign '#' in the listing file.

Often you have alternate chunks of source code. If the symbol is defined, you need to compile one chunk, and if it's false, you need to compile the other chunk. The qFORTH compiler enables you to do this with the **$ELSE** directive.

Note: All $IFDEF directives must be completed within the same source file, which means they cannot start in one source file and end in another. However, an $IFDEF directive can encompass an include file.

Example:
```
$IFDEF MUX4–LCD
$INCLUDE LCD–MUX4.SCR
$ELSE                \ otherwise 3:1 MUX
$INCLUDE LCD–MUX3.SCR
$ENDIF
```

In this way, you can select alternate include files based on the same condition. You can nest **$IFDEF .. $ENDIF** constructs to achieve the following results:

```
$IFDEF Version–2
              <source code A>
$IFDEF Debug
              <additional code>
$ENDIF        \ end of debugging output
              <source code B>
$ENDIF        \ Version–2
```

4.3.2 Compilation Control

Index Checking

$I+ Default setting
$I–

Normally the index checking for the array indices is on. When the default setting **$I+** is active during compilation, the constant array indices must be kept in the range 0 to <length-1>.

By using **$I–** for a section of code, the index which is-checking is switched off, i.e., any constant array index may be specified. For example, specifying the **DataArray [-1]** could be useful while writing to the array using **[+Y]!** or **[+X]!** instructions within a loop.

Macro Expansion Control

$EXPAND Default setting
$NOEXPAND

To modify the time of the macro expansion and thereby the amount of optimization done by the compiler, the $EXPAND and $NOEXPAND directives may be used. The use of the directives $EXPAND or $NOEXPAND on the outside of a CODE definition sets this directive globally. This means that the macro expansion mode influences all following CODE definitions.

By default all macros are expanded before the optimization process is started. The directive $NOEXPAND means that CODE definitions are expanded after the optimization process has finished.

Branch Stripping Algorithm

$BRA_STRIP NOTALL Default setting

Unconditional branches are stripped so that short branches will stay short branches. i.e., if a short branch leads to a second unconditional short or long branch, the first short branch could be stripped. If this results in a long branch stripping is suppressed.

$BRA_STRIP ALL

All branches are stripped, regardless of wether short branches could become long branches. This kind of branch stripping may result in an increase in code length, but will minimize the execution speed.

ROM CRC-Algorithm

$CRC <arg>

The **$CRC** directive (**C**yclic **R**edundance **C**heck) checks the contents of ROM. The check sum will be stored after compilation at the last two ROM bytes of the last physical ROM bank.

The following arguments are available:

DEFAULT	16-bit software CRC
SIMPLE	8-bit software CRC (optimized code size)
HARDWARE	16-bit hardware CRC (for MARC4 variants with built-in selftest)

4.3.3 List-File Directives

The list file directives will only have an effect, if **/LIST** was specified in the command line or as one of the compiler options in the integrated environment.

$NOLIST Default setting
$LIST

The source listing is suspended by **$NOLIST** until **$LIST** is found again in the source code.

$PAGE

$PAGE will force a form feed in the print output file, if the list output is active.

$DEBUG_STACKS

The compiler directive **$DEBUG_STACKS,** when included in one of the source files, writes the calculated expression and return stack effects of all code and colon definitions into the print file. The four columns following the source line number contain stack depth values that are relative to the beginning of this source line.

The sequence of the columns is as follows :

– current number of nibbles on the expression stack,
– current number of used return stack entries,
– maximum expression stack depth reached within this routine (nibbles),
– maximum return stack depth reached within this routine.

```
     $DEBUG_STACKS
                                : INT7
 5 |   1 |   5 |   1 |                 PortData @
 6 |   1 |   7 |   1 |                 Port0 OUT
 5 |   1 |   7 |   1 |      ;


                                $NOEXPAND
                                $OPTIMIZE    –XYTRACE
                                CODE X–
 0 |   0 |   0 |   0 |                 [X–]@   DROP
 0 |   0 |   1 |   0 |      END–CODE
                                $OPTIMIZE    +XYTRACE


 0 |   0 |   0 |   0 |                 >SP  S0
 0 |   0 |   0 |   0 |                 >RP  NoRAM
 0 |   0 |   0 |   0 |
 0 |   0 |   0 |   0 |      Port0   IN   0 =
 0 |   0 |   2 |   0 |      IF         RAM_TEST
 0 |   0 |   0 |   0 |                 ROM_TEST
 0 |   0 |   7 |   3 |      THEN
 0 |   0 |   7 |   3 |
 0 |   0 |   7 |   3 |      0   0   Timer_A   2 !
 0 |   0 |   7 |   3 |
 0 |   0 |   7 |   3 |      PortData   X!   X–
 0 |   0 |   7 |   3 |      8  #DO
 0 |   1 |   0 |   1 |                 0  [+X] !
 0 |   1 |   7 |   1 |      #LOOP
 0 |   0 |   7 |   3 |
 0 |   0 |   7 |   3 |      2_Hz  Prescaler  OUT
 0 |   0 |   7 |   3 |      ;
```

All values related to the return stack are counted as 16-bit or 4 nibbles entries. The MARC4 core uses 12-bit words on each return stack entry. The address space of the fourth nibble, not used by the return stack, will be assigned by the compiler for single 4-bit variables.

All calculated values are relative to the start of the CODE or colon definition. The expression stack values always start with 0. The return stack value starts with 0 in CODE definitions. In colon definitions it starts with 1 because of the return address which is already saved on the return stack.

4.3.4 Stack Effect Directives

The following directives have no effect if the compiler switch **WARNINGS** is turned **OFF**. The warning messages of the compiler are very helpful when looking for an unexpected expression stack under-/overflows or i.e., different stack effects of **IF .. ELSE .. THEN** parts.

On the other hand, the programmer may be aware of the fact that i.e. a LOOP block eats up a specified number of elements from the stack. Therefore, if the programmer is sure that this particular code works perfectly, the compiler warnings can be turned **OFF**. These compiler directives will be placed at the end of each ‘block’ of qFORTH words (i.e., a **DO .. LOOP**).

They always start with ‘**[** ’ and end with the symbol ‘ **]**’. In between those two symbols each combination of the following directives are allowed:

E <number>	Define expected expression stack effect,
R <number>	Define expected return stack effect,
Ex <number>	Define maximum expression stack effect,
Rx <number>	Define maximum return stack effect,
?	Return and expression stack effects of the previous block are unknown, the corresponding **WARNING** message will be turned **OFF**.

Example:

```
$I-                         \ Turn index checking OFF
: BCD@                      \ Push a number of
                            \ digits onto the stack
    Multiplier [ −1 ] Y!    \ Setup array pointer
    8 #DO
        [ +Y ]@             \ Push an array
                            \ element onto the stack
        [ E 0 ]             \ Turn the compiler
                            \ warning message OFF
    #LOOP
    [ E 8 ]                 \ Set correct number
                            \ pushed onto stack
;
$I+
```

4.3.5 Optimization Control

The amount of optimization done during the compilation process can be controlled by the **$OPTIMIZE** control switch. By default all optimization steps will be performed.

$OPTIMIZE <switch1>, <switch2>

The ABSOLUTE range of optimizations to be performed is set by qualifying the control switch **$OPTIMIZE**. The only types of optimization performed furthermore are those, that are listed after **$OPTIMIZE**.

$OPTIMIZE {+ |–}<switch1>, {+ |–}<switch2>

The optimization qualifiers can also be used in conjunction with the **$OPTIMIZE** control switch for a

RELATIVE setting in the source files. The kind of optimizations performed is determined by adding (+) or removing (–) the listed types from the current optimization set.

$OPTIMIZE ?

The current optimization control settings are written into the print output file.

$NOOPTIMIZE Default setting
$OPTIMIZE

All kinds of optimization are inhibited when the **$NOOPTIMIZE** is specified. Whereas **$OPTIMIZE** will cancel a previous **$NOOPTIMIZE** directive, i.e., the optimization set is the same as before the **$NOOPTIMIZE** directive.

It is possible to control the optimization process in such a way that some specific subroutines or macros will not be optimized. For example, no register tracing in a hand-optimized memory block MOVE routine.

$OPTIMIZE Qualifieres

The user may parameterize the **$OPTIMIZE** directive with the following qualifieres:

CALL	CALL optimizer pass [CALL →SCALL],
BRANCH	Branch optimizer pass [BRA → SBRA],
CMP	Comparison optimizer,
DROP	DUP .. DROP optimizer,
SWAP	SWAP .. SWAP optimizer,
XYLOAD	Register load optimizer,
XY@!	Register load with memory fetch/store operation,
XYTRACE	Register scoreboarding, preincrement /postdecrement

XYLOAD

Sequences like **LIT_p LIT_q .. X!** will be optimized to a **>X $pq** instruction.

XY@!

Sequences like **>X $pq .. [X]!** will be optimized to a **[>X]! $pq** instruction.

XYTRACE

By reloading the **X** or **Y** register sequences like **[>X]@** or **[>Y]! $pq** will be replaced by **[+X]@** or **[Y-]!** operations, whenever possible.

CMP

Sequences like **CMP_cc** .. **TOG_BF .. BRA** are optimized to the sequence **CMP_$\overline{cc}$** .. **BRA**, where $\overline{cc}$ is the opposite condition of **cc**. Also **TOG_BF .. TOG_BF** sequences are omitted which may result from macro expansions.

CALL

A **CALL** instruction is replaced by a **SCALL**, whenever possible.

SAVECONTXT

The **INTx** prefix and postfix register save macro (**X@ Y@ CCR@ .. CCR! Y! X!**) is reduced, whenever possible. If **INT5** does not change the register, **X@** and **X!** are removed from the routine's prefix and postfix sequence.

The lowest priority interrupt routine may be compiled with:

```
$OPTIMIZE −SAVECONTXT
: INT0      Calculate_On_Off
            Update_LCD
;
$OPTIMIZE +SAVECONTXT
```

BRANCH

A long branch instruction is optimized to a short branch instruction within a code page whenever possible.

BRA_EXIT

Unconditional branches to an **EXIT** instruction are replaced by an **EXIT**, also unconditional branches to an instruction that is placed directly before an **EXIT** are replaced by this instruction followed by an **EXIT**.

BRA_STRIP

A branch to a second unconditional branch will be changed so that the first branch goes directly to the target of the second branch. This will not save any code, but result in a faster execution speed. See also the compiler directive **$BRA_STRIP**, which allows you to control the amount of branch stripping being performed.

DROP

Any sequence **<Push nibble onto stack>** .. **DROP** will be removed from the code if this nibble is not used anywhere else and results in no side effects.

Note: Because **[+Y]@ DROP** will change the Y register, it is not optimizable.

SWAP

Any sequence **SWAP .. SWAP** will be removed whenever possible. Furthermore, any sequence **LIT_x .. LIT_y .. SWAP** will be optimized to **LIT_y .. LIT_x**.

4.4 Compiler Optimization Steps

The previous section described how to use the compilers optimization directives. The code optimizations implemented are reviewed in this section.

4.4.1 Branch Optimizer

Short branches are used whenever the address is achieveable within the present 64-byte page, otherwise full branches are used. The programmer does not need to be aware of any page boundaries.

4.4.2 Call Optimizer

Short calls can only be used for colon definitions in the Zero Page (the first 512 bytes). These definitions are automatically selected to be placed in the Zero Page as a result of their size and static usage. The programmer can force a Zero Page placement by appending either **AT <Address>** or **'[Z]'** compiler directives at the end of a colon definition.

4.4.3 Peephole Optimizer

The peephole optimizer replaces a sequence of instructions with a shorter, more efficient sequence. In general, a stack architecture allows a much wider peephole than normal, as stack effects within a 'basic block' may be evaluated at compile time. This means that a given code sequence does not need to be consecutive. Currently eight separate peephole sequences are checked. The following example shows the two sequences which were found to occur most frequently.

Example 1: Compile time constant folding

Source	Assembly code	Optimized code
FRED @	Lit_3 Lit_4 X! [X]@	[>X]@ $FRED

Example 2: DUP DROP optimizing resulting from the MARC4 implementation of the compare instructions, where only one of the top two elements is dropped.

Source	Assembly code	Optimized code
DUP 3 =	DUP	Lit_3
IF	Lit_3	CMP_NE
..	CMP_EQ	SBRA $THEN
THEN	DROP TOG_BF BRA $THEN	

4.4.4 Register Tracking

While a good assembly code programmer may never write code with redundant **DUP** and **DROP** instructions, it is often the case that he may forget exactly which variables and addresses are cached in registers. A good compiler however, can keep track of which register contains are variable. This is especially true in qFORTH since the programmer's model of the machine has no additional registers.

Example 1: Variables **FRED** and **BERT** are in consecutive RAM locations

Source	Assembly code	Optimized	Final code
FRED@	Lit_3	[>X]@ $FRED	[>X]@ $FRED
BERT +!	Lit_4	[>Y]@ $BERT	[+X]@
	X!	ADD	ADD
	[X]@	[Y]!	[X]!
	Lit_3		
	Lit_5		
	Y!	(+! macro)	
	[Y]@		
	ADD		
	[Y]!		

Sometimes register tracking may also eliminate redundant address register loads across an IF statement.

Example 2:

```
FRED @ DUP 5 < >
IF          BERT !
ELSE        DROP
            0 FRED !
THEN
```

4.5 The Command-Line Compiler

Compiling qFORTH programs can also be done by using the command-line approach common to most computers where each step in program generation occurs from the command line. On your PC this means from the DOS command line indicated by the prompt, such as the drive indicator.

C:\MARC4 > qFORTH2 [/<switch>]
<filename>[/<switch>]

To invoke the compiler, enter the program name **qFORTH2** followed by the filename to be compiled. Normally, a file extension is not required since **'SCR'** is default when compiling a main program.

As an example, to compile a file called 'MYFILE.SCR' with the generation of a list and object code file, the following command-line sequences would be accepted as valid by the compiler:

QFORTH2/LIST/NOSTAT MYFILE
QFORTH2 MYFILE/LIST/STAT=NO
QFORTH2/LIST/SYM MYFILE/STAT=FULL

An overview of the various compiler switches, options and directives accepted by the command-line compiler is listed in the subsequent sections of this chapter.

After the compilation of your program is completed, the DOS drive indicator will appear on the screen permitting you to either enter the simulator or emulator (in command-line mode) or to go back to your program editor to correct any possible errors which may have occured.

4.5.1 Compiler Generated Messages

The generated messages of the command-line compiler version are the same as the MARC4 environment integrated version.

4.5.2 Compiler Generated Files

A listing of all generated files is shown in table 1.

4.5.3 Setting the Compiler Switches

The compiler switches of the command-line version are the same as the integrated version. Default switch settings do not have to be called in the command line. For more detailed information, see the section 'Compiler Switches' of the integrated version.

Object Code Generation

/NOOBJECT

/OBJECT[=<object file>] **Default setting**

This controls whether a binary object code file has to be generated. The default extension is **".HEX"** and the default filename is that of the source file.

/NOSYMBOLS

/SYMBOLS[=<symbol file>] **Default setting**

This switch controls whether a symbol file has to be generated. The default extension is **".SYM"** and the default filename is that of the source code. This file is necessary if you want to check your code with all defined symbols (subroutines and variables). By pressing the function key **<F7>** in the software simulator or emulator, you can take a view the symbol table data.

List File Generation

/LIST[=<list file>]

/NOLIST **Default setting**

This switch controls whether a source listing has to be generated. The default extension is **".LST"** and the default filename and path are that of the source file. This generated file contains all events during compilation, depending on additional compiler switches.

/NOWARNING

/WARNING **Default setting**

This controls whether warnings will be written onto the screen and – with the setting of the additional switch "$List" – in the list file, too.

/NOSTATISTICS

/STATISTICS[=<statistics qualifier>]

<**statistics qualifier**>:

/STATISTICS=NO (is identical to NOSTATISTICS)

/STATISTICS=BRIEF

/STATISTICS=NORMAL **Default setting**

/STATISTICS=FULL

No

By setting the switch, all statistical information is suppressed in the list file.

Brief

Generates a summary of all errors, all defined words and all defined variables at the end of the list file.

Normal

Lists additionally the return and expression stack usage of all routines, the addresses of all words placed in ROM, a summery of left ROM holes, unused RAM nibbles and unused short-call address entries, an overview of bytes saved during the optimazition steps and information about the compiler's memory usage.

Full

Additional information about the subroutine placement algorithm, the CPU time for the different compilation steps, statistics on the usage of the internal symbol table data base, summary of created files and used compiler switch settings.

Debugging Support File Generation

/ASSEMBLER[=<assembler file>]

/NOASSEMBLER **Default setting**

This controls whether an assembler list file will be generated. The default extension is **".ASS"**, the default filename is that of the source file. This output file may be used to check the efficiency of the generated object code.

/CRF[=<crossreference file>]

/NOCRF **Default setting**

This controls whether a cross reference file has been generated. The default extension is **”.CRF”**, the default filename and path is that of the source file. The cross reference file shows the correlations of all used symbols (subroutines, variables and constants) with regard to their definition and their use in the different source files.

/LOG[=<HLL file>]

/NOLOG **Default setting**

This switch controls whether a high-level language debugger link file has to be generated. The default extension is ”.HLL”, the default filename and path is that of the source file. This generated file enables source-level debugging (see chapter 5 ”Software Simulator”).

Library Management

/NEWLIB[=<library file>]

/LIBRARY[=<library file>[,<library file>]]

This command controls whether one or more user libraries have to be read after the system library has been read.

/SYSLIB

Controls whether a new system library has to be generated or not. The default filename is ‘qFORTH2.LIB’, generated from the input source file. The source files to be compiled into a system library must have a certain format, otherwise the compilation will fail.

Note: This compiler switch is reserved for TEMIC’s internal use only.

4.6 Error and Warning Messages

Notes:

All errors marked with (****) are severe errors which indicate that the compiler does not work properly. In this case, you should send your source code which caused the error, together with your system library and a brief description, to

TEMIC Semiconductors
MARC4 Applications Department
Erfurter Str. 31
D-85386 Eching

Fax: +49–89–3194621

DOS errors, which are preceeded by the word DOS, are not explained in this manual. Refer to your DOS manual.
Turbo (Pascal) runtime (RT) errors are caused by an incorrect compiler source code.

4.6.1 Coded Error and Warning Messages

001 File not found

When including a file with the $INCLUDE directive, this file was not found. All files to be included are expected in the same directory as the source file, as long as there is no directory path preceeding the filename.

005 Turbo RT: Object not initialized (****)

TURBO runtime error caused by an incorrect compiler code.

006 Turbo RT: Call to abstract method (****)

TURBO runtime error caused by an incorrect compiler code.

050 WARNING — Source line too long. Truncated after 120 characters

A source line is always processed up to 120 characters only. Additional characters are ignored.

051 WARNING — End of file reached while scanning comment

When scanning comment, the end of file was reached prior to the end of comment. The closing parenthesis ‘)’ seems to be missing.

052 Too many nested INCLUDE’s. INCLUDE will be ignored.

Includes may be nested only 4 levels deep. Additional nested include files are ignored. Nevertheless, including can be done sequentially without limitations.

053 Numeric value out of range

The numeric value read was either out of the machin’s integer number range, or an array index was out of its range. Arrays always start with index 0. This message is also issued if you force an object via ’AT’ to a location outside of the current RAM or ROM address range.

054 Internal stack overflow (****)

The compiler’s internal number stack has overflowed.

055 Internal stack empty (****)

The compiler’s internal number stack was empty when the compiler tried to get a number from the stack.

056 Number expected

The compiler expected a number or constant as next item within the source file. This message is often seen on constant or array definitions following a look-up table. Please rearrange the sequence of definition so that a variable or colon/macro definition follows a look-up table.

057 Assembler definitions expected when creating library

When compiling a system library source, the compiler always expects a section with assembler definitions at the beginning. This section was not found.

058 Additional characters ignored

There are characters after the end of a program in your source file. They will be ignored by the compiler.

059 Only CODE, ’:’ or CONSTANT definitions are permitted

In a system library source, only CODE, COLON and CONSTANT definitions may occur.

060 END–ASSEMBLER expected

The end of the assembler section has to be marked with this word. The compiler did not find it in the system library source code.

061 QFORTH–LIBRARY expected

The COLON and MACRO definition in a system library source have to be enclosed by the words QFORTH–LIBRARY ... END–LIBRARY. This error occurs if there are no COLON or MACRO definitions at whatsoever.

062 Reserved word QFORTH–LIBRARY not found, will be added

When compiling a system library source, a COLON or MACRO defintion was found before the word QFORTH–LIBRARY.

063 Unable to handle. Skipped to next

When looking for the beginning of an object definition, an unusable object definition was found. The compiler skips to the beginning of the next object definition.

064 ’:’ added

Whenever an undefined name is found, and the compiler looks for the beginning of a new definition, this name is regarded to be the name of a COLON definition, where the user forgot to write the colon.

065 WARNING — Undefined Word

An undefined word was found within a COLON or MACRO definition .

066 Undefined label or label referenced outside of definition

All labels used within a COLON or MACRO definition have to be defined in this definition, unless the labels are maked as ’special labels’ which begin with the two characters ’_$’.If this error occurs, one or more labels within a definition were not defined.

067 END–CODE expected

When compiling a macro, the beginning of the next definition was found while the macro was not compiled completely. In this case, an END–CODE is added by the compiler which causes the compilation of the macro to be completed properly.

068 ’;’ expected

When compiling a COLON definition, the beginning of the next definition was found while the colon definition was not compiled completely. In this case, a ’;’ is added by the compiler which causes the compilation of the colon definition to be completed properly.

069 WARNING — There is no special handling of negative numbers

The MARC4 is not able to process signed numbers in binary format. All negative numbers used in your source file will be treated as positive values.

070 $VERSION expected

The system library source code must begin with a $VERSION statement. The word $VERSION must be followed by a string that will identify this library version. The compiler also uses the version string to check the validity of user libraries.

071 Only ’CALL’ and ’BRA’ instructions permitted

When using assembler instructions in COLON or MACRO definitions you are not allowed to use the short-call (SCALL) or short branch (SBRA) instruction. The compiler will optimize the long branch (BRA) and long-call (CALL) instruction to SBRA and SCALL instructions whenever possible.

072 Insufficient space for intermediate code (****)

When compiling a program, the code is stored in an intermediate array before the object code is assembled. This error does not occur, if the space reserved within the compiler for intermediate code is defined, to be large enough.

073 ’%’ or ’$’ not permitted in label names

These two characters may not occur in label names, for they are reserved to the compiler’s use when substituting macros. Furthermore care should be taken when using labels beginning with an underscore, for most labels in the qFORTH library begin with an underscore. This might cause duplicated label names.

074 WARNING — Label too long. Truncated to 16 characters

The length of a label is limited to 16 characters.

075 Duplicate label names

Within your program, two duplicate label names were found. You have to rename one of them.

Note: Avoid label names beginning with an underscore, as this might cause interferences with label names already used within the qFORTH library.

076 “]” expected

The option list or an array index must always be enclosed in square brackets. In an option list, these brackets must be preceeded and followed by at least one blank. When supplying an index, the opening bracket must be preceeded by at least one blank, the closing bracket must be followed by at least one blank. The index may be preceeded or followed by one or more blanks optional. An index may only occur after an array name in the source code.

077 WARNING — Stack effect of word not computable

Normally, the compiler computes the EXP and return stack effects of every COLON and MACRO definition. This is impossible if

– BRA assembler instructions are used in the definition,

– you use a COLON or MACRO definition whose stack effects are un-computable,

– this COLON definition is recursive,

– this COLON or MACRO definition contains an IF–ELSE–THEN statement where THEN and ELSE part have different stack effects,

– this COLON or MACRO definition contains any loop (DO .. LOOP or #DO .. #LOOP or ... or BEGIN ... AGAIN or BEGIN ... UNTIL or ...) in whose block the RET or EXP stack effect is <> 0

– this COLON or MACRO definition contains DO ... +LOOP statement wherein the RET stack effect is not 0 or the EXP stack effect is not 1.

–this COLON or MACRO definition contains a CASE statement wherein the effects of all selections are not the same. You can suppress this warning, by classifying the COLON/MACRO as one, which stack effect do not has to be computed by supplying the option ’?’ or by explicitly typing the stack effects in the option brackets, e.g. [E 0 R 0].

078 Only ’:’ definitions can be forced to ZERO PAGE

Only COLON defintions can be forced to the zero page by the ’Z’ option. A COLON definition forced to the zero page will be placed there, regardless whether it is called by any routine or not.

079 Nesting of object definitions not permitted

Object definitions may not be nested, i.e., you can not write a COLON or MACRO definition within an other COLON or MACRO definition.

080 ELSE or THEN expected; THEN will be added

When processing the THEN part of an IF statement, the beginning of another object definition or the end of the current definition was found. In this case, a THEN is added to correct the block structure.

081 THEN added

When processing the ELSE part of an IF statement, the beginning of another object definition or the end of the current definition was found. In this case, a THEN is added to correct the block structure.

082 #LOOP added

When processing an #DO ... #LOOP statement, the beginning of another object definition or the end of the current definition was found. In this case, a #LOOP is added to correct the block structure.

083 Last numeric entry omitted

When scanning the source code the compiler has to do a look-ahead of one word to process CONSTANT or ARRAY definitions. Therefore, when a number is found at the beginning of an object definition, the compiler has to read the next word to decide whether the number is valid or not. If a number is invalid, this is flagged at the word following the number with this message.

084 Predefined value is already initialized

Predefined constants like $RAMSIZE or $ROMSIZE can only be set once in a program’s source code.

085 WARNING — Return stack doesn’t start at address 0

The return stack does not start at adress 0, because it was forced to another location by ’AT’. This means, when an RET stack underflow occurs, NO SLEEP mode is entered and program exectution will continue at a random location. You should ensure, that this mode is impossible when forcing the RET stack to a specific address.

086 $RAMSIZE value is insufficient

By declaring too large stacks or too much arrays or variables, there is insufficient space in the internal RAM to place all objects into it. The compiler has to be told the RAM size in the predefined constant $RAMSIZE, or a default value of now 111 nibbles is used.

087 AT not permitted here

The AT part of an array or a variable definition has to stand in front of the ALLOT part.

088 A label became too long when macroing

Calling macros in other macros over several levels may cause the label name length to overrun the limit. You should use shorter names or less excessive macro-in-macro-calls.

089 LOOP added

When processing a ?DO/DO ... LOOP statement, the beginning of another object definition or the end of the current definition was found. In this case, a LOOP is added to correct the block structure.

090 WARNING — THEN and ELSE block with different stack effects

When processing a COLON or MACRO definition, an IF–THEN–ELSE statement with different stack effects in the THEN and ELSE part was found. This causes the stack effects of the current COLON/MACRO definition to be uncomputable. An IF–THEN–ELSE statement with an absent ELSE part is regarded as an IF–THEN–ELSE statement with an RET and EXP stack effect of 0 in the ELSE part.

091 WARNING — RET stack effect in LOOP block is <> 0

The current COLON/MACRO definition contains any kind of loop in which the RET stack effect is not 0. This causes the stack effects of the whole COLON/MACRO definition to be uncomputable.

092 WARNING — EXP stack effect in LOOP block is <> 0

The current COLON/MACRO definition contains any kind of loop in which the EXP stack effect is not 0. This causes the stack effects of the whole COLON/MACRO definition to be uncomputable.

093 WARNING — EXP stack effect in final +LOOP block is <> 1

The current COLON/MACRO definition contains a DO ... +LOOP statement in which the EXP stack effect of the last block in front of the +LOOP is not 1. This causes the stack effects of the whole COLON/MACRO definition to be uncomputable.

094 UNTIL, WHILE or AGAIN expected. UNTIL added

When processing a BEGIN statement, the beginning of another object definition or the end of the current definition was found. In this case, an UNTIL is added to correct the block structure.

095 REPEAT added

When processing a BEGIN ... WHILE ... statement, the beginning of another object definition or the end of the current definition was found. In this case, an UNTIL is added to correct the block structure.

096 ENDCASE added

When processing a CASE .. OF .. ENDOF .. statement, the beginning of another object definition or the end of the current definition was found. In this case, an ENDCASE is added to correct the block structure.

097 ENDOF added

When processing a CASE .. OF .. statement, the beginning of another object definition or the end of the current definiton was found. In this case, an ENDCASE is added to correct the block structure.

098 System library incomplete! Contact TEMIC for immediate support (****)

One basic part of the system libray QFORTH.LIB was not found. This error only occurs if your system library has been damaged by hard-disk errors. For first aid, delete qFORTH2.LIB and dearchive this file from MARC4.ARJ on the installation disk.

099 Internal compiler error ! Contact TEMIC for immediate support (****)

Supply your source code for failure analysis.

100 $AUTOSLEEP and $RESET have fixed ROM addresses, AT ignored

The $AUTOSLEEP routine is always placed at ROM address 000h, while $RESET is placed at ROM address 008h. Trying to force these routines to different start addresses will result in this warning.

101 Symbol longer than 20 characters – truncated

Whenever a symbol name with more than 20 characters is defined, the name is truncated to 20 characters.

102 WARNING — Dirty programming style – Stack effects uncomputable

This warning occurs if you are using BRA instructions and lables within a MACRO/COLON definition. The compiler is not able to calculate the correct stack effects.

103 WARNING — Redefining a Number with a qFORTH word

By creating a new vocabulary entry and linking it in the word-list, this warning will occur if you have defined a new object with the name which already exists.

104 Undefined ROM–Segment

The name of the defined ROM segment is unknown.

105 Expected $ENDSEG – $ENDSEG inserted

If routines or ROM constants are to be placed into specific ROM segment blocks, they have to be enclosed by the compiler directives $BEGINSEG .... $ENDSEG. During compilation, the ROM segment directive $ENDSEG was expected because the compiler has found the beginning of an additional ROM segment definition. The directive $ENDSEG was inserted automatically.

106 No previous $BEGINSEG

The compiler found the directive $ENDSEG of a ROM segment definition without a ROM segment block being introduced by the directive $BEGINSEG.

107 Overriding previous Segment assignment

A routine within a $BEGINSEG .... $ENDSEG block overlapped with a SEGMENT directive for a single placement of a ROM constant or subroutine.

108 Expected SEGMENT – SEGMENT added

During compilation, the word SEGMENT was expected into the $DEFSEG directive. The compiler inserted the word SEGMENT automatically.

109 Defined Segment does not fit into parent segment

The defined ROM space area is greater than the ROM space area of the superior ROM segment block.

110 ROM–Segment of $AUTOSLEEP, $RESET, and INTx routines cannot be changed

The $AUTOSLEEP, $RESET and INTx routines have fixed addresses in the ROM base bank and can not move into another ROM segment.

153 Unexpected end of source file

The end of a source file was found before a definition in the source code was completed. Maybe a <CR> following a 'j'or ENDCODE statement is missing in the last line of a source file.

160 Only "@" or "!" operations are allowed with external objects

If you use external storage, the only operations on external objects, which have been declared as EXTERNAL before, are store and fetch operations. This means, for example, you can not push the address of an external object onto the stack and manipulate it.

161 Global labels in macros are not allowed

The use of global labels (beginning with _$) is forbidden, as this label would cause multiple definition problems when this macro is called.

163 $EXTMEMSIZE value is insufficient

The size of the external memory is too small, i.e., not all external objects can be placed.

164 You can not call that

The only thing that can be called by CALL-assembler-instructions are subroutines, defined by COLON definitions or labels within assembler subroutines.

165 Do not redefine assembler words

Assembler words can not be redefined.

166 Optimize XYTRACE : Not enough memory in partition list (****)

An internal list of the qFORTH compiler is too small.

167 Fatal error during XYTRACE (****)

A fatal internal compiler error occured during XYTRACE optimization. Please submit your source code to TEMIC for failure analysis.

168 Partition–pointer–list too small (****)

An internal list of the qFORTH compiler is too small.

169 Invalid Context–Save/Context–Restore Macros

An internal compiler error occured during the compilation of a new system library.

170 Found operator where operand was expected

171 Found operand where operator was expected

172 String constant truncated to 80 characters

If a string constant with more than 80 characters is defined, the string constant is truncated after 80 characters.

197 Use $Bank_Switch FULL

Replace the default argument RESTRICTED of the compiler directive $BANK_SWITCH <arg> by the argument FULL.

198 Panic: Could not place subtree of SCALL–Routine in BB

To organize the base bank efficient, the compiler has to place all routines with fixed ROM addresses into the zero page (INTx, $RESET, ...). The next step of the compiler is to place all routines and subroutines into the base bank which will be forced to a SCALL address. After that, the zero page is filled up with SCALL routines. This error occurs, if there is not enough memory space to place the corresponding routines into the base bank. You have to rearrange your source code .

199 TBL not actualized

Stack operation error caused by SWAP and DROP instruction. This failure may occur during compilation of a new system library.

200 Internal consistency check failed!

Internal compiler error! Please contact your TEMIC sales person for immediate support.

201 WARNING — Different RET stack effect than in previous block

When processing a CASE .. OF .. ENDOF .. ENCASE statement, the current block has a different return stack effect than the previous block. This causes the stack effects of the current COLON/MACRO definition to be uncomputable.

202 WARNING — Different EXP–stack effect than in previous block

When processing a CASE .. OF .. ENDOF .. ENCASE statement, the current block has a different EXP stack effect than the previous block. This causes the stack effects of the current COLON/MACRO definition to be uncomputable.

203 Illegal word will be ignored

When processing a COLON/MACRO definition, an improper keyword was found (such as THEN without a prior IF or an UNTIL without a prior BEGIN ...) or within a ROMCONST definition an unusable word was found. Within ROMCONST definitions, only numbers, constants, strings, and names of arrays, variables are allowed.

204 End of file reached while reading a string

When processing a string, the end of the source file was found before the end of the string was reached.

205 ',' expected

When defining a look-up table with ROMCONST, each item has to be followed by a blank and a comma, the last item too.

206 Index out of range

When indexing an array, the index was less than 0 or greater than the maximum index.

207 WARNING — Index expected

Everytime an opening square bracket after an array name is found, the compiler assumes to read an array index which has to be a number or a constant of the appropriate range.

208 WARNING — Value not in range 1 .. 16

The maximum index when defining a short byte or short nibble array has to be less than or equal to 15 and greater than or equal to 0. The index for any short array has to be a nibble. When defining an array, the bracketed number is the number of array elements whose index starts at 0 !

209 Value not in range 1 .. 256

The maximum index when defining a long byte or nibble array has to be less than or equal to 255 and greater than or equal to 0. The same range is valid when indexing a short array. The index for a long array has to be a byte.

210 WARNING — Unknown or invalid option

An unknown option was found when specifying an option list after a block within a COLON or MACRO definition.

211 WARNING — No INTERRUPT routine has been defined

Each program has to contain at least one definition of an interrupt service routine. The interrupt service routines are named INT0 to INT7. The compiler uses the defined interrupt routines to compute the set of used subroutines by following the execution path for every interrupt routine.

212 Recursion in CODE definitions not permitted

A macro definition can not be used in its own definition.

213 WARNING — Recursive stack effects unpredictable

The stack effects of a recursive COLON definition can not be computed. Nevertheless you can specify them in the option list using '[ExRy]'.

214 Inconsistent assembler definitions in system library (****)

This error indicates inconsistency in the definitions of assembler words in the system library.

215 WARNING — Forcing to an address overrides forcing to ZERO PAGE

If a COLON definition is forced to the zero page by the ’Z’ option and forced to a specific address by AT, the AT overrides the ’Z’.

216 DPMI: General Protection Fault (****)

Internal compiler error! Please contact your TEMIC sales person for immediate support.

217 Unable to place $RESET routine in ROM

The compiler was unable to place the $RESET routine at address 008h or there is not enough space to place the whole $RESET routine. Reduce the size of the $RESET routine.

218 Unable to place INTERRUPT routine in ROM

The compiler was unable to place an interrupt routine at a predefined address as there was not enough space to place the routine. The reason and the steps to avoid this are the same as described in error 217.

219 FATAL ERROR – Routine became longer when optimizing (****)

This means that a subroutine increased in length when it was optimized.

220 Insufficient ROM for placing ROM constant values

There was not enough space left in ROM when the compiler tried to place the ROM constant look-up table definition. The ROM constants are placed after all subroutines have been placed. If this error occurs, increase the size of the on-chip ROM (using $ROMSIZE) or break ROM constants and subroutines into smaller parts, so they can fit into smaller ROM holes not used.

221 FATAL ERROR during optimization (****)

A fatal error occured when optimizing the intermediate object code.

222 FATAL ERROR during ROM placement (****)

A fatal error occured when carrying out the ROM placement. Typical error if the ROM size is too small.

223 RET stack does not fit into RAM

The size of the return stack is greater than on-chip RAM. Increase the on-chip RAM or decrease the size of the return stack allocation.

224 EXP stack does not fit into RAM

The size of the EXP stack is greater than the size of on-chip RAM. Increase the on-chip RAM, decrease the size of the EXP stack or return stack allocation.

225 RET and EXP stack will overlap

You forced the EXP stack to a specific address so that both stacks do not use disjointed parts of RAM. Do not force the EXP stack to a specific address.

226 Not the name of a colon definition

You tried to use an assembler mnemonic or another reserved word of a colon definition.

227 WARNING — Unkown interrupt source of current routine – Stack effects 0,0 assumed

You have defined a SWI-instruction in the current routine, but the compiler cannot find out the interrupt which will activate this routine.

228 Insufficient $ROMSIZE for placing subroutines

Not all necessary subroutines could be placed in the MARC4's ROM. The actual ROM size is given to the compiler by the predefined constant $ROMSIZE. If this directive is not found in the source, a default ROM size is taken.

229 WARNING — RET stack size not set – Using default value

The return stack size was not set by the sequence VARIABLE R0 <xx> ALLOT where <xx> is the size in nibbles. The default value is 48 nibbles. The number of return stack entries is calculated by (<xx> /4) + 2.

230 WARNING — EXP–stack size not set – Using default value

The EXP stack size was not set by the sequence VARIABLE S0 <xx> ALLOT where <xx> is the size of the expression stack –1. The default value is 16 nibbles.

231 WARNING — Cannot determine target of SWI instruction

You have called an interrupt before the interrupt service routine is defined. Please try to rearrange your source code.

232 WARNING — Not all Z–optioned routines could be placed

There is not enough space in the zero page thus, not all subroutines, which are forced to a short-call address by the Z-option, could be placed on a short-call address. Decrease the length of interrupt or $RESET routines or use less forcing to a zero–page address by AT or Z-option.

233 WARNING — Using default value for $ROMSIZE

You did not set the predefined constant $ROMSIZE. The default value of 1.5K is taken.

234 WARNING — Using default value for $RAMSIZE

You did not set the predefined constant $RAMSIZE. So the default value of 111 nibbles is taken.

235 WARNING — Using default value for $EXTMEMSIZE

You defined external memory objects, but did not specify the size of the external memory. So a default value is taken. The default value is set to 255 nibbles. This warning is issued only if external objects have been defined.

236 WARNING — Using default value for $EXTMEMPORT

The predefined constant $EXTMEMPORT needs the port address for external memory accesses. The default value is Fh.

237 WARNING — Using default value for $EXTMEMTYPE

If external memory is used, the types RAM or EEPROM are valid parameters. RAM is the default parameter.

238 Unkown CRC–Algorithm. Valid are DEFAULT, SIMPLE and HARDWARE

The compiler supports three CRC algorithm which influence the checksum generation differently. Only DEFAULT, SIMPLE and HARDWARE are valid parameters. If there is no $CRC-directive defined in your source-code, the CRC algorithm is DEFAULT.

239 Unknown Bank switch method. Valid are Restricted and FULL

The compiler has found an invalid argument for the directive of the ROM bank switch. Only RESTRICTED and FULL are valid arguments. If there is no $BANK_SWITCH <arg> directive defined in your source-code, the $BANK_SWITCH argument is RESTRICTED.

241 No previous $IFDEF

When using conditional compilation, the compiler found a $ELSE or a $ENDIF but no previous $IFDEF.

242 Unmatched $IFDEF(s), possibly $ENDIF missing

When using conditional compilation, there were unmatched $IFDEF(s) left at the end of the source, i.e., there were more $IFDEFs than $ENDIFs.

244 Program aborted via <CTRL+C>

This error message is issued when the compiler is terminated via <CTRL+C> keyboard break.

245 – 247 TURBO–RT: TURBO runtime error (****)

This TURBO-runtime errors are caused by an incorrect compiler code. Please contact your TEMIC sales person for immediate support.

248 WARNING — Protect only assembler words in Colon or Code Definitions

The compiler directive $PROTECT must be used only to protect assembler words in colon or code definitions by creating a new library.

250 WARNING — $(NO)EXPAND meaningless in Colon definition

This compiler directive is only a macro expansion control.

251 TURBO–RT: TURBO runtime error (****)

This TURBO runtime error is caused by an incorrect compiler code. Please contact your TEMIC sales person for immediate support.

252 WARNING — Expansion mode has already been set globally

If the compiler directive $(NO)EXPAND is set on the outside of a code definition, the expansion mode is set globally. This global expansion mode can be set once in your source code only.

4.6.2 Uncoded Error and Warning Messages

Error message file <path> qFORTH2.MSG not found

The compiler's error message file is not available in the compiler's directory.

User library <name> not found

An user library was not found. Check, whether you supplied the correct extension, or if your user libraries have the default extension .LIB, whether you supplied the correct path or, if no path was supplied, whether they are stored in the same directory as the source file.

<name> is never called

You forced a routine to an address or to the zero page, which is never called directly by any other routine. You may omit this routine, or, if you do not force it, the compiler will not place it in ROM area.

ERROR — Illegal environment found by the compiler (****)

The compiler reads the program's environment to determine the path where the compiler overlay can be found. Use an MS-DOS operating system release 3.3 or above.

<list of files> : List is ignored

Do not pass more than one source file to be compiled. All except the first will be ignored.

<list of qualifiers> : List of qualifiers not allowed

The list of qualifiers <qualifix> is not allowed here and will be ignored.

Contradicting switch <switch> is ignored

You supplied two contradicting switches such as LIST and NOLIST. Omit one of them.

<qualified switch> : qualifying not allowed

You qualified a switch which does not allow qualifying.

<qualifix> : unknown qualifier

You specified an invalid qualifier after the /STATISTICS switch. Valid are only: NO, BRIEF, NORMAL and FULL (or abbreviations of these)

XY@!–optimizing requires XYLOAD–optimizing

You cannot optimize indirect–X/Y-fetch/store unless the loading of the X and Y register is optimized. XY@! optimizing is not completed.

XYTRACE optimizing requires XY@!–optimizing

You cannot carry out a register trace optimizing unless 'XY@!' is optimized. XYTRACE optimizing is not done.

Unable to place <name> (<length> Bytes) (****)

The remaining space in ROM is too small to place the flagged object. Use less forcing to zero page or to an address or break large routines into smaller pieces. This warning will also produce a severe error.

This word is redefined (see <place of first definition>)

You defined the same object twice. This might cause problems when referenced by other routines. Rename or remove one object from your source code.

X@/Y@ in <name> : content could be changed by the optimizer

Within <name> you used the Y register, although XYTRACE optimization is done. This might change the normal content of the X or Y register. Don not use these registers by direct assembler instructions.

Call <routine> in <name> : Don't pass parameters in registers

<routine> uses the X or Y register. The compiler assumes that parameters to <routine> are passed in the registers. If XYTRACE optimization is carried out in <name>, the original content of these registers might change.

<name> can't be forced, because ROM address is fixed

Interrupt routines have fixed ROM addresses:

$AUTOSLEEP	000h	INT3	100h
$RESET	008h	INT4	140h
INT0	040h	INT5	180h
INT1	080h	INT6	1C0h
INT2	0C0h	INT7	1E0h

Unexpected end of file in library

When reading the system or a user library, an end of file was found before the library was read completely. Copy the qFORTH system library from the installation disk to your working directory. If the error still occurs, recompile your user library/libraries.

Illegal <record–type> record in library

The library file is destroyed by any reason or you are trying to read an old library with a new compiler version. Use the new system library, if you have a new compiler version, re-install the system library from the installation disk to your working directory. If the error still occurs, recompile your user-library/libraries.

<predefined value> is already set

A predefined value is set in more than one user library. This is not allowed. Recompile, so that the value is set in one user library only.

Duplicate label <labelname> in User Library

Two user libraries contain the same label name. Rename one of them.

<name> : name not found when reading cdrflist (****)

A part of your user library is lost by damaging the user library. Recompile it, if the error still occurs, send your files to TEMIC for failure analysis.

No more room for intermediate code (****)

You have read in too many user libraries. Reduce the number of user libraries or split them up into a larger number and omit all libraries you do not need in this application program.

Your user library is inconsistent

You are trying to read an old user library in conjunction with a new version of the qFORTH system library. Recompile all your user libraries.

Switch <switch> is ambiguous

The switch <switch> is ambiguous, i.e., it is not possible to determine exactly which qualifier is meant (e.g., \LI is ambiguous (LIST or LIBRARY)).

Switch <switch> does not care me

You supplied an unknown switch.

Source file not found

The compiler did not find the specified source file. Maybe you specified an invalid path or the extension of your source file is not the default extension .SCR or .INC for an include file and you did not specify it.

System library QFORTH2.LIB not found

The compiler is looking for the system library in the same directory the compiler is stored in. Make sure that the system library is available as QFORTH2.LIB in that directory.

Qualifier <qualifix> is ambiguous

The qualifier <qualifix> in a qualifier list is ambiguous, i.e., it is not possible to determine exactly which qualifier you mean.

Qualifier <qualifix> does not care me

You supplied an unknown qualifier <qualifix> in the qualifier list. Indirect recursion not allowed. By re-definition of one or more objects you got an indirect recursion. This is not allowed.

<name> does not fit into ROM–hole

You forced a ROM item into a too small ROM hole, by forcing another item to near to this item in the ROM. Use less forcing or supply larger distances between the items.

<name> there is already something in ROM

You tried to force two items in the ROM. As result, they are overlapping. Move one of them to another location or use less forcing with AT.

<name> – defining occurence not found

An external labels was not defined. When an external label is used, its name has to be preceeded by '_$'. The name at the defining occurence must not be preceeded by there characters.

<predefined variable> — Value not set (****)

Indicates that a predefined variable was not set, not even with its default value.

<object> has not been defined

An external object (arguments of assembler instructions with operands or parts of a ROM constant) was not defined until the end of the source code.

<name> – out of address space (****)

A call of <name> or a branch to <name> exceeds of the maximum 4K address space

<array><index> : out of RAM space

An array fits only partially into RAM. Use less forcing with AT.

Conflict between RET–stack and <ram–object>

The RET stack and another RAM object do not occupy disjoint RAM. Use less forcing via AT or move one object.

Conflict between EXP–stack and <ram–object>

See above

Conflict between <ram–object> and an other RAM object

See above

<external object> is not in available external storage

An external object was forced via AT to a location outside the available external memory area.

Conflict between <name> and another external object

Two external objects do not occupy disjoint memory areas. Use less forcing via AT or move one object.

5 Software Simulator

5.1 Introduction

A software simulator is a computer program used to imitate the operational function of another system (implemented either in the hardware or software).

The simulator accepts the same input data, executes the application program in the same way as the target environment and accomplishes the same results as the targeted system which it imitates. A software simulator operates as a rule much slower than the targeted system and does not have any physical resemblance to the hardware system.

Simulators are used to develop and debug application programs written for target hardware which may not be available.

5.1.1 Simulation as an Instrument for Program Verification

The software simulator has at its core a register-transfer-level model of the MARC4 processor. Figure 1 shows the various interfaces between the software simulator and the input, output and program data files. By using this total system, it is possible to verify that your MARC4 application program is functionally correct. Stack effects as well as RAM values can be checked by using the simulator. The status of the ports and interrupt registers can be observed. By using an input polling file, it is possible to simulate with the expected real-world data under simulated real-time conditions. To support the programmer in finding errors in his program, a breakpoint feature is integrated into the simulator. The trace memory data enables easy examination of the instructions that have been executed before a breakpoint occurred. To help the programmer in analyzing the simulation results, they can be written to an I/O capture file.

5.2 Getting Started

Before starting the simulator, make sure that the following files are available in the MARC4 base directory:

SIM05.EXE, SIM05.DAT, SIM05.HLP, SIM05.CFG

- To start the simulator with the SDS2 environment: Check that the correct directory path for the simulator has been entered in the setting window “Directories” (see Installation Guide). Press the function key **<F7>**.

Note: On starting the simulator, the program file which has been defined as the project file in the SDS2–IDE will be loaded by default.

- To start the simulator as an independent program: If the MARC4 directory has not been included in the AUTOEXEC.BAT search path, change into the MARC4 system directory and enter the following command:

  C:\MARC4>SIM05 PROGRAM\TIMER

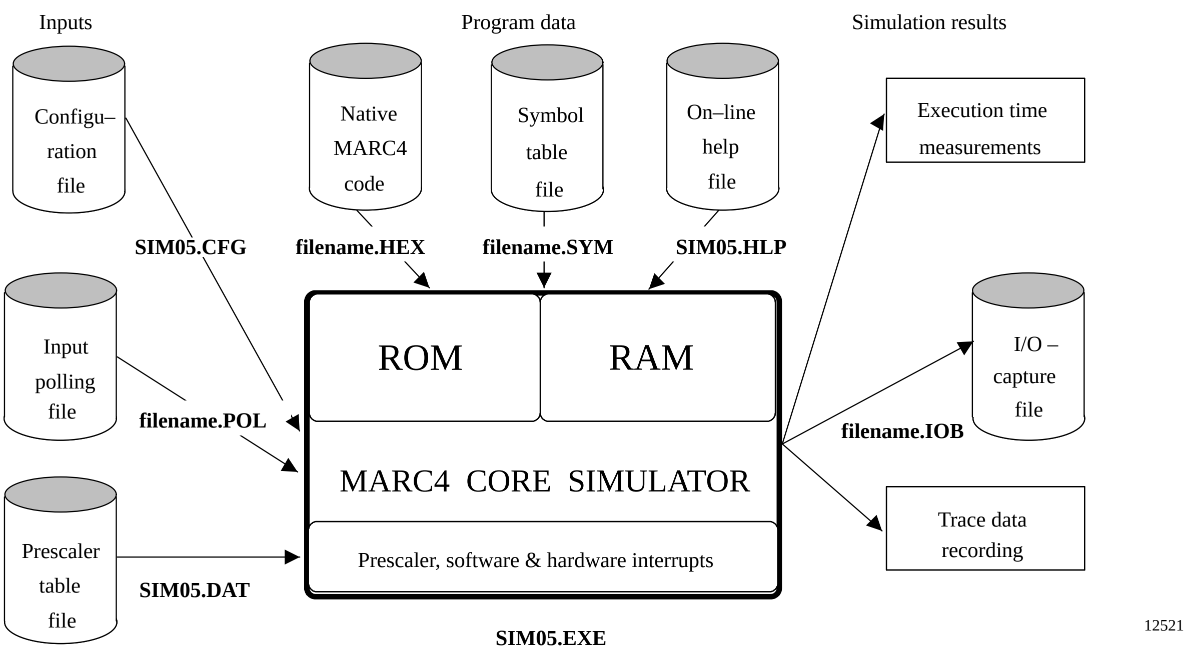


Figure 1. Software simulator input/output interfaces

The argument PROGRAM\TIMER is the path and file name of an example project which has to be simulated.

When exiting, the name of the simulated file will be saved on the file "SIM05.CFG". In the case of a new or changed file which has to be simulated, a warning message will be displayed in the message line.

5.3 Using the Simulator

5.3.1 Simulator Screen

The screen is subdivided into windows which show the current status of the MARC4 microcontroller as the program code is being simulated (see figure 2).

The different standard windows provide the following information:

ROM disassembly displays the code to be executed

Expression stack lists the contents of the data stack

MARC4 registers display the status of the 3 internal flags, the contents of the program counter and the RAM address registers

Return stack lists the addresses pushed on to the return stack

RAM data displays a portion of the RAM locations

Port status displays the direction of data at the ports

Time displays the total instruction execution time and the percentage of time the processor was active

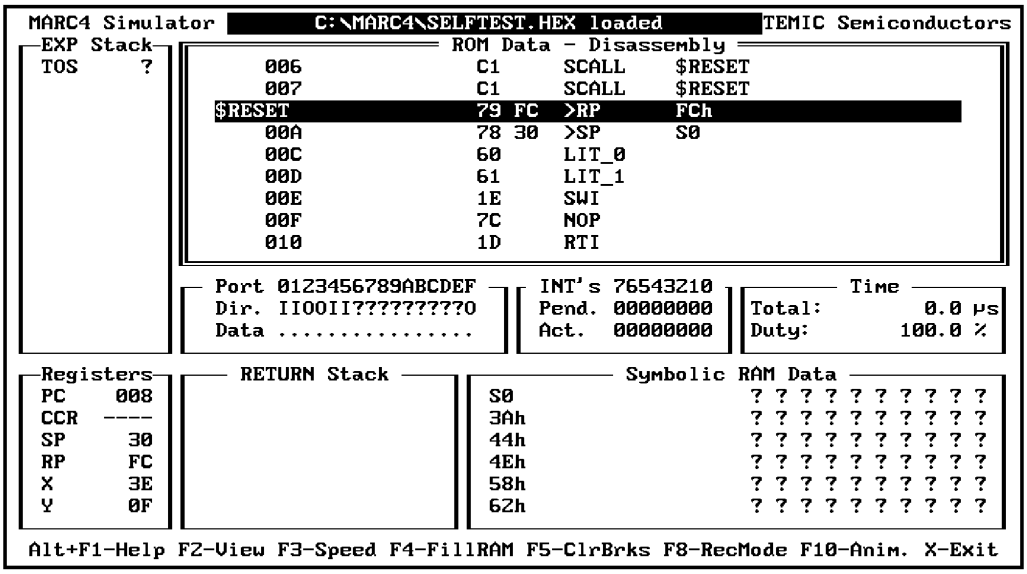
The message line at the top of screen contains the display status, error or help information. The active line at the bottom of the screen describes the function keys used by the simulator. This line can be toggled by pressing the **<Alt>**–key. Furthermore, there are some additional windows which will be displayed by pressing a specific function key. They will disappear again after pressing the **<ESC>**–key. These windows and their function keys will be described later

Select a window:

Keyboard: To change the active window, please use the arrow keys <←>**,** <→>**.**

Mouse: Move the mouse cursor over the window and press the left mouse button.

By pressing the **<Pos1>/<Home>**– key on your keyboard, the display of the currently active window will move to the top address location, i.e. the ROM disassembly window to the $AUTOSLEEP address. The **<End>** key is the corresponding counterpart. The scroll page commands also work in the active window.



12522

Figure 2. MARC4 software simulator screen

5.3.2 Simulator Window Description

ROM Disassembly

When tracing the program execution with the simulator, the ROM disassembly window will show the MARC4 native code instructions, their opcodes and symbolic addresses. The line that is currently highlighted has an arrow pointing to the ROM address which is the same as the PC. The $AUTOSLEEP and $RESET symbols are always located at address 000h and 008h in the ROM. These are predefined in both the MARC4 processor and the **qFORTH** compiler.

MARC4 Registers

The MARC4's RAM can be indirectly addressed via one of the four 8-bit wide RAM address registers:

- **SP** - Expression stack pointer,
- **RP** - Return stack pointer,
- **X** - RAM pointer register X,
- **Y** - RAM pointer register Y.

Your MARC4 application program written in **qFORTH** is disassembled into the MARC4's native code language and stored in the ROM. In order to access the 8-bit wide ROM instructions (or look-up tables), a 12-bit wide program counter (**PC**) is used.

- **CCR** - condition code register

The **CCR** is a 4-bit wide register containing ALU status information. The simulator has an area within the register window which displays the following information:

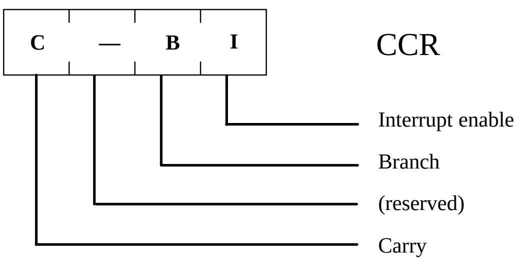


Figure 3. The MARC4 condition code register

RAM Data

The RAM data window displays the contents of the random access memory. The RAM addresses are usually displayed as symbolic names, the data values are shown in hexadecimal notation. Locations which are not initialized are shown as '?'.

The two stacks within the RAM have a user-definable depth and are displayed in different colors. They are refered to as the expression and the return stack.

Expression Stack

The expression stack is used to store parameters as 4-bit wide data elements. The expression stack window displays those parameters beginning with the top of stack element (**TOS**) as the first nibble followed by the TOS-1, the TOS-2 and so on which are stored in the RAM.

Return Stack

The return stack, which is displayed in the return stack window, is usually used to store 12-bit wide subroutine return addresses. A return address is generated whenever a **CALL** to a subroutine or an interrupt acknowledge is performed from the executed program.

The return stack is also used to store the loop index of **DO .. LOOP** and **#DO .. #LOOP** sequences. By using the **>R**, **2>R** or **3>R** assembler instructions, it is possible to move data from the expression to the return stack. The **R>**, **2R>** and **3R>** instructions are the counterparts.

Interrupt Status

The MARC4 is able to handle up to 8 prioritized interrupts which can be generated asynchronously from either on-chip modules, external sources or synchronously from the CPU itself (software interrupts).

The transmission of the interrupts occurs via the MARC4's internal I/O bus. The software simulator enables viewing of the 'pending' as well as the 'active' interrupts. The interrupt section of the simulator screen is presented below. The top line of the window shows the priority of the interrupts.

INTs	7 6 5 4 3 2 1 0
Pending	0 0 1 0 0 0 1 0
Active	0 0 1 0 0 0 0 0

Time

The execution time window displays the time needed by the simulated MARC4 program up to the point it is stopped either by the user or at a breakpoint match. Additionally, it will be updated periodically. The time shown is the actual time required by the MARC4 processor when running your program at a specific instruction cycle time. The simulated instruction cycle time can be changed using the 'Speed' command **<Alt-F3>**.

Whenever the program has been stopped, you are able to reset the time to zero by pressing **<Alt-F6>**. This enables examination of the time that passes until the next breakpoint match occurs.

Port Status

See special section on I/O simulation.

5.4 Simulator Commands

5.4.1 Command Keys Summary

Key	Function	Short Description
F1	Step	Single step
F2	Step call	Step over call
F3	Run	Execute simulation until breakpoint or user break occurs
F4	Reset	Simulator (re-) initialization
F5	BrkPts	Breakpoints set-up function
F6	Load	Read in a binary object file
F7	Symb	Display symbol table information
F8	Trace	Display recorded trace data
F9	Edit window	Window data editor
F10	Source code	Display source code
Alt–F1	Help	Pop–up help window
Alt–F2	View	View ROM data from start address or symbol
Alt–F3	Speed	Change MARC4 processor speed
Alt–F4	FillRAM	Fill a section of RAM with specific data
Alt–F5	ClrBrks	Reset all earlier defined breakpoints
Alt–F6	ClearTime	Reset elapsed time in time window to zero
Alt–F7	Exit	Exit simulator, return to enviroment or DOS
Alt–F8	RecMode	Select mode for trace data recording
Alt–F9	Print	Print the contents of RAM, ROM or trace memory
Alt–F10	Animation	Animation = continous single step execution
Alt–X	Exit	Exit simulator, return to enviroment or DOS
Alt–0 ... 7		Entering hardware interrupts during simulation
Shift–F1		Show version number and date of creation
Shift–F2		Change the prescaler address
←,→		Change the current window
Pos 1/Home		Set current window of its top address
End		Set current window of its last address
Page ↑ Page ↓		Scroll page

5.4.2 Simulator Commands Description

Single Step <F1>

The simulator walks through the program code displayed in the ROM disassembly window instruction by instruction for each **<F1>** selection. The message ’1 instruction executed’ is then displayed on the screen’s message line. The internal RAM and register content will be updated and displayed on the screen.

Step over CALL <F2>

This executes all deeper-nested routines until the **EXIT** or **RTI** instruction of a following subroutine is found. The ROM disassembly will be highlighted at the location following the [**S**]**CALL** invocation. The message line will display ’Step over CALL executed’.

Program Execution <F3>

The simulator will execute the application program until a break occurs or the processor enters the sleep mode and no interrupt is pending. While executing the program, the message line will display ’Running’. When the program has been completed, the message line will display ’Halted -No interrupt pending’. The program execution may be stopped by one of the following events:

- a user keyboard break by pressing any key during execution
- a PC or I/O breakpoint match
- an expression stack under-/overflow
- a return stack overflow
- an interrupt request was lost
- an indefinite sleep mode occurred.

To continue the execution of the program, press the **<F3>** key again.

Animation <Alt–F10>

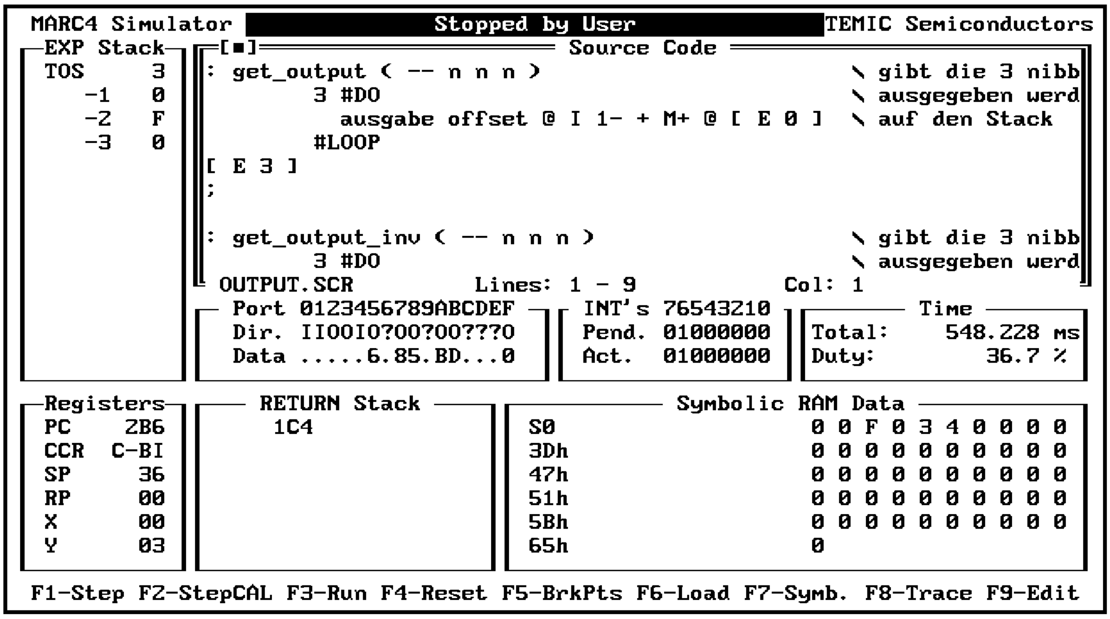
The simulation will walk through the program code displayed in the ROM disassembly window instruction by instruction and the screen will be updated after every step. To stop the animation, you may press any key. The message line will display the number of instructions that have been executed.

Animation Speed <Shift–F1>

Will slow down the animation mode. This may be of use if you are running this software on a fast PC. To run the animation mode with faster speed, press the function key again. Additionally, the version number and date of creation will be displayed in the message line.

Source Code Window <F10>

The source code window enables source level debugging without the need to leave the simulator. It displays the qForth source file and the present simulator execution point.



12523

Figure 4. MARC4 software simulator source code window

The following function keys support source code scrolling: **<Page up>, <Page down>, <End>, <Pos1>/<Home>.**

Note: This option requires a project HLL file to be generated during compilation with the "hll linkage" compiler switch set.

Simulator Initialization <F4>

This command resets the simulator and all screen display windows. The **PC** is set to ROM address 008h similar to a real power-on-reset. The RAM cells are set to undefined (displayed as '?'), the software preset **RP** to FCh and **SP** to **S0** to ensure that the stack pointers are initialized on the display. The message line will then display 'MARC4 processor has been reset'.

Note: The breakpoints are not reset to their default values.

Setting Breakpoints <F5>

The breakpoint window appears where the ROM disassembly window is usually displayed. Pressing the **<Esc>** key will result in the breakpoint window being closed and the reappearance of the ROM disassembly window.

The software simulator enables the programmer to test the program by setting breakpoints in his code or data area. Breakpoints enable you to stop the execution of your program at any (**PC**) address. You may enter up to four different PC breakpoints. These breakpoints are classified as passpoints, since the number of times the program counter can pass the specified address location can be set to a maximum of 255. Besides this, there are two RAM breakpoints to stop execution whenever there is an expression stack over-/underflow or a return stack overflow, and two I/O breakpoints to stop whenever specific data is read from or written to a given port. An additional breakpoint stops the program execution whenever an interrupt request is lost, caused by a processor overload or an RC oscillator speed which is too low.

If a breakpoint occurs during simulation, all windows are updated. By using either the **<F1>**, **<F2>** or **<F3>** function key, it is possible to continue execution either until the next breakpoint match or a user break (keyboard entry) has been encountered.

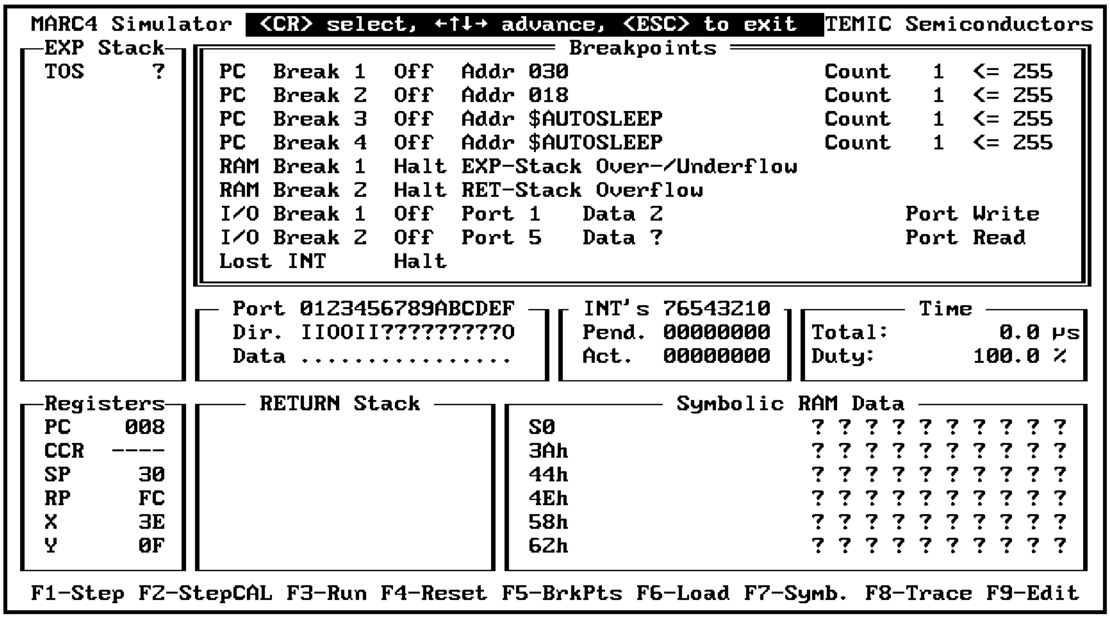
Three different breakpoint options are available:

Off Turns off a specified breakpoint or snapshot. The counter is not of interest in this state.

Halt Will stop the simulation at the defined address when the counter reaches zero (only for PC breakpoints). The message line will for example display 'Stopped at PC Break 1'.

Snap Snapshot causes a screen update every time the specified address is passed during the program execution. For example, the message line will display 'Snapshot at PC Break 1'. The count field is not of interest in this state.

If you want to modify the attributes of a breakpoint, please follow the instructions given in the message line .



12524

Figure 5. Breakpoint window

Reset all Breakpoints <Alt–F5>

All breakpoints are set to the default status, i.e., ’Halt’ for the RAM breakpoints and the breakpoint on lost interrupts, and ’Off’ for all other breakpoints. The message line will display ’All breakpoints cleared’.

Loading a New Program File <F6>

To load a new program file, a window is displayed in which you are prompted to enter the binary object file to be simulated. The file extension is assumed as **HEX**.

If you cannot remember the correct name and directory, just press **<Enter>**. A separate file pick window will be opened and you may browse through the directory tree to find the new MARC4 object file. This file will be read into the simulated ROM.

Note: All breakpoints will be reset to their default values similar to the command **<Alt-F5>**.

Select Mode for Trace Recording <Alt-F8>

In this case, a pop-up window allows you to choose between different trace modes. These are:

- Instruction cycle trace
- Subroutine entry histogram AND/OR
- I/O cycle trace.

The selection of one of the first two items enables the displaying of the trace window after the execution of parts of the program. If **I/O cycle trace** is selected, a file with the extension **IOB** will be opened which has the same name as the project file. It contains all the information about the types of interrupts that occurred, and the port read/write operations that the program has performed during the simulation. See also the section on I/O capture files.

Displaying of Trace Data <F8>

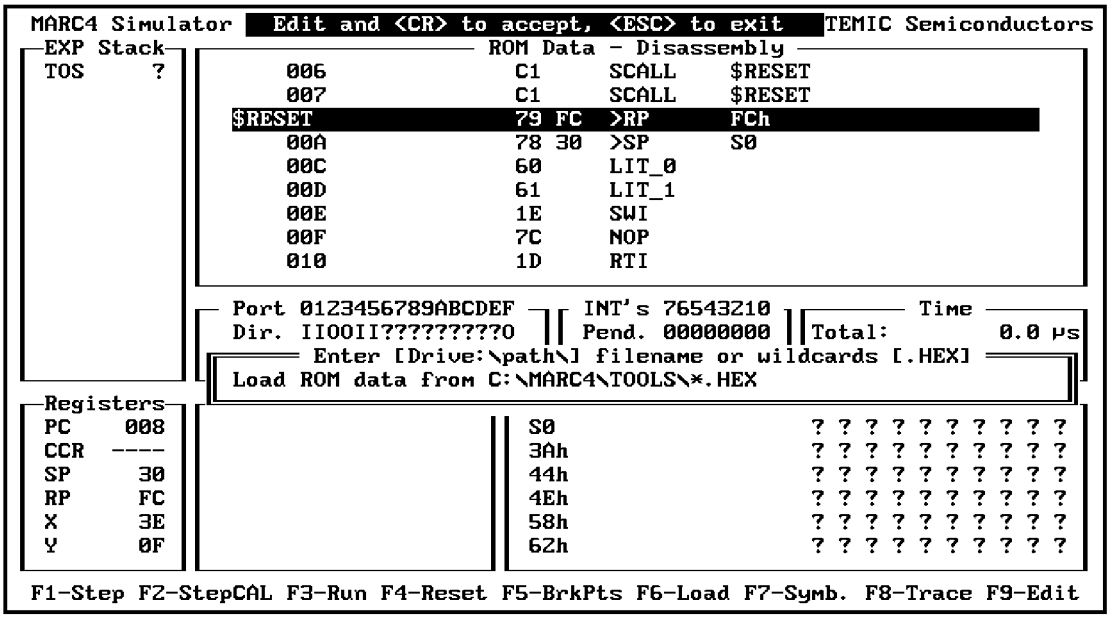
By pressing this key, a window displays the recorded trace data for the instruction cycles, or the subroutine entries are opened. The window can only be opened if the following conditions have been met:

- **<Alt-F8>** has been selected before AND
- either instruction cycle or subroutine entry recording has been chosen.

If this conditions have not met, an error message will be displayed on the message line.

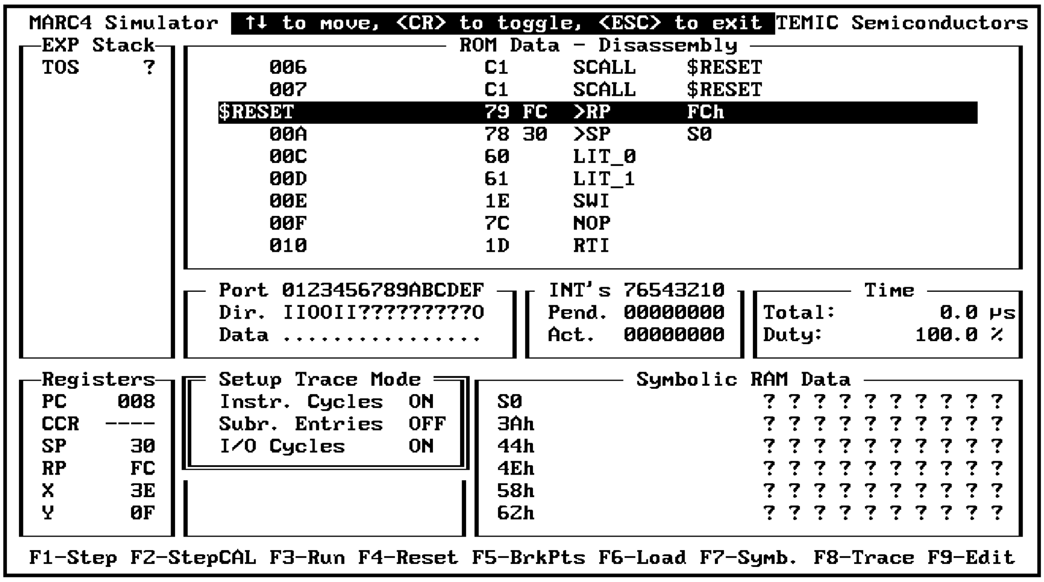
In the trace data window you can examine up to 1023 executed instructions (see figure 8), or look at the subroutine entry histogram (figure 9) of your program which depends on the chosen topic in the **<Alt-F8>** pop-up window.

For example, if the instruction cycles have been recorded, the first column in the trace window shows the time that has passed between the instruction in the line you are looking at and the last executed instruction. The last line shown in the screen’s window is the last executed instruction which was executed before the breakpoint occurred. The second column displays the program counter for that line, the next columns show the instruction followed by the mnemonic of that instruction.



12525

Figure 6. Loading a new program into the ROM



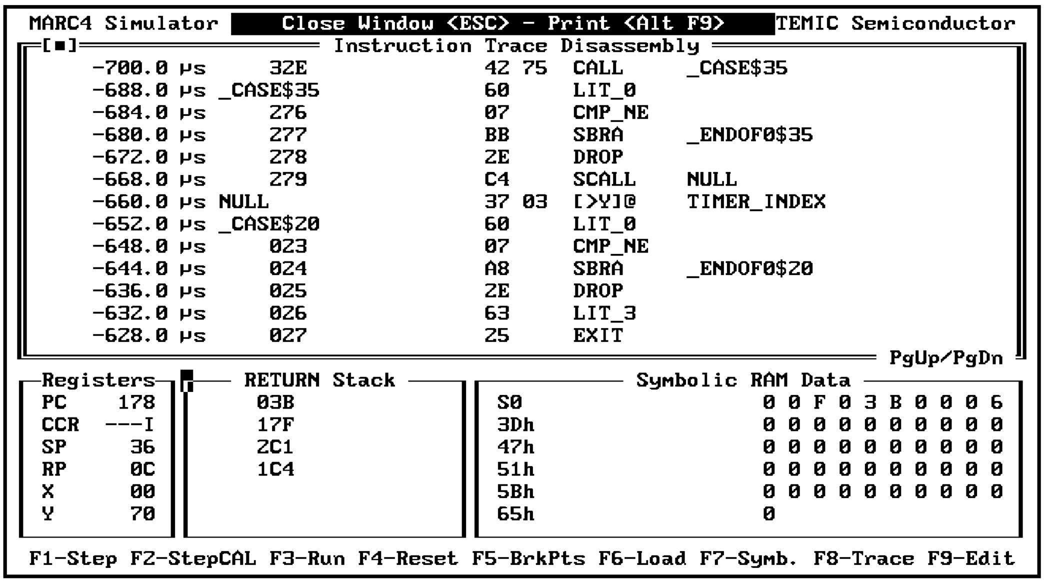
12526

Figure 7. Selection of different trace data recording mode

On the right hand of the trace window, you can see the transferred I/O data if there was either an **IN** or **OUT** instruction executed or an interrupt occurred. On color monitors, it is displayed on a blue background with the address and data of the port or the priority of the interrupt. If there was an interrupt request, this will be displayed by an ’INT !’ first. The decoding of that interrupt will occur some cycles later.

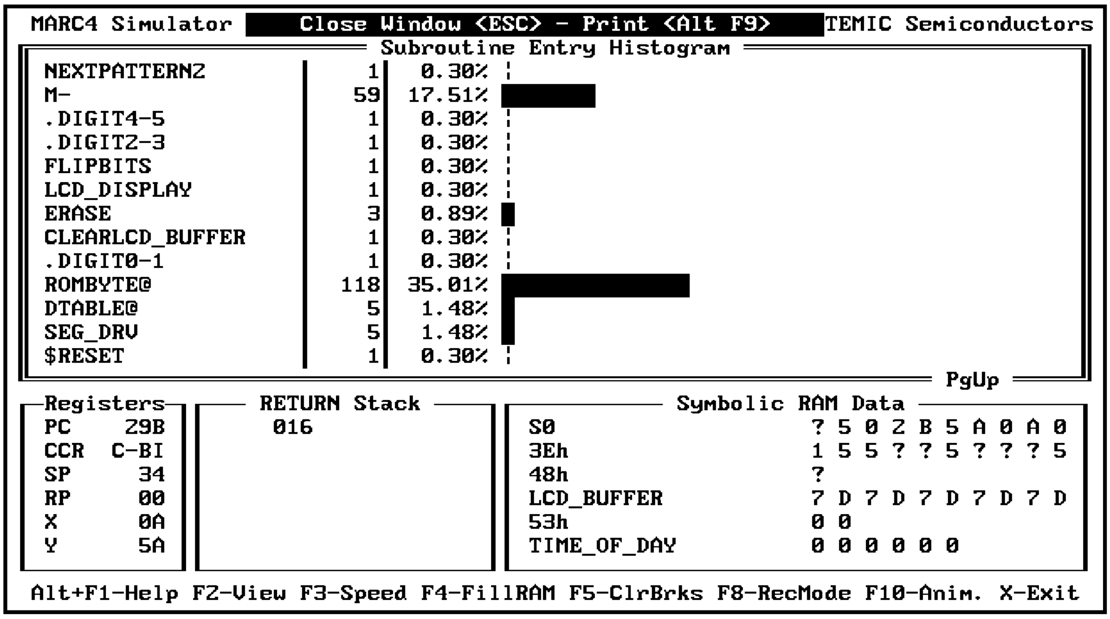
If you have recorded the subroutine entries of your program, the trace data window will display the following items. The first column shows the symbolic addresses the program jumped to at every interrupt acknowledge **CALL** or **SCALL** instruction. The second column displays the number of times each subroutine was called. The third column shows the percentage and the fourth column is a bar graph of the percentages.

You may use the **<PgUp>**, **<PgDn>**, **<Up>**, **<Down>**, **<Pos 1>/<Home>** and **<End>** keys to browse through the trace window. If you want to print the contents of the trace data window, you may press the **<Alt-F9>** function key. Pressing the **<Esc>** key will close the window.



12527

Figure 8. Instruction trace disassembly



12528

Figure 9. Subroutine entry histogram

Editing Data in a Window <F9>

Pressing **<F9>** enables you to edit data in the currently selected window (the one with the bold frame around it). When you have finished, you can leave the edit mode by pressing the **<Esc>** key. To change the currently selected window, press the **<Left>** or **<Right>** arrow key. Only selectable windows can be edited using the **<F9>** key.

These are listed below:

- MARC4 register window

  The **X**, **Y**, **SP**, **RP** and **PC** registers can be edited using symbolic RAM and ROM addresses.

  The **CCR** flags can be set by entering an '1' or the corresponding character ('C', 'B', 'I') at the correct bit position. To reset a CCR flag you may type either '0' or '–'.

- ROM disassembly window

  Instructions can be changed in the opcode column using the opcode bytes of the MARC4's instruction set. Please refer to the '**MARC4 Programmer's Guide**' for the opcodes.

- RAM data window

  RAM locations can be overwritten nibble-wise using hexadecimal values.

  Refer to the 'Fill RAM' command to initialize partitions of the RAM.

- Expression stack window

  The **TOS** and the values of all other items on the stack can be modified using hexadecimal numbers.

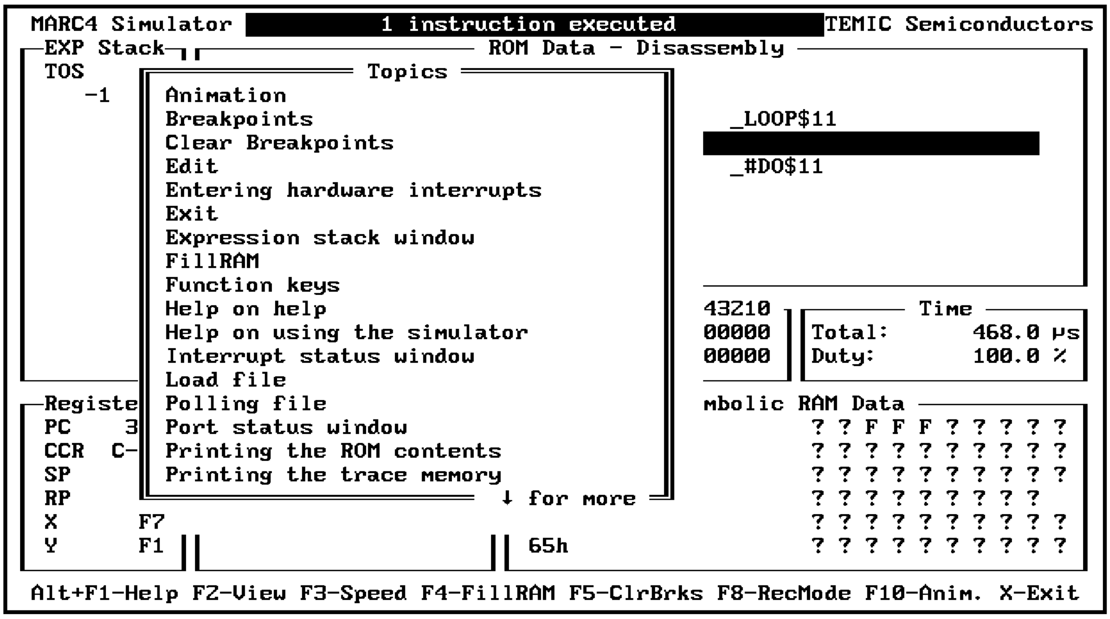
Pop-up Help Function <Alt–F1>

By pressing **<ALT–F1>,** a pop-up help window will open which displays information about the currently selected window. If the currently selected window is the ROM disassembly window, general information about the simulator will be displayed. In the help window, you can get information about the highlighted and blinking topic by just pressing **<Enter>**. You can move the highlighted bar among the cross-reference topics (written in yellow letters) by pressing the arrow keys **<Up>**, **<Down>**, **<Left>** or **<Right>**.

If a 'PgUp/PgDn' appears at the lower right hand corner of the window, you can display the previous or next help page by pressing **<PgUp>** or **<PgDn>**.

Press **<Alt-F1>** if you want to display the help topic most recently selected.

Press **<F1>** if you want a list of all the topics described in the help menu. To exit any help window press **<Esc>**.



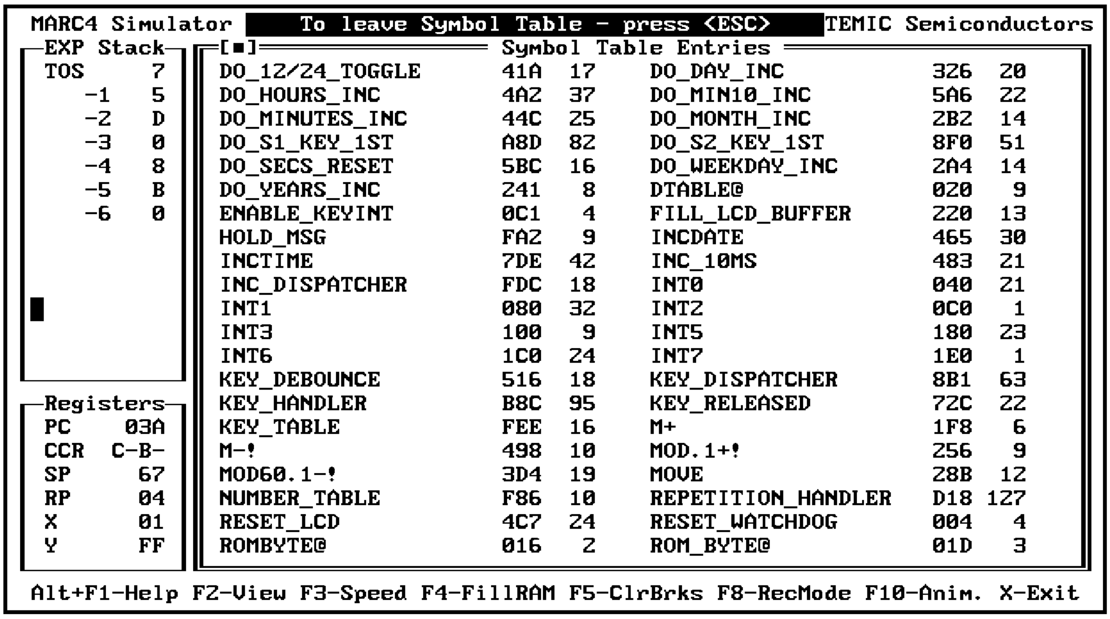
12529

Figure 10. Pop-up help window

Display of Symbol Table Information <F7>

A small pop-up window will be displayed in which you can choose between symbols of subroutines, variables or constants. If you press **<Enter>,** the chosen symbol types together with their addresses and lengths or constant data will be displayed in the symbol table window.

A RAM address followed by an 'x' marks this variable as EXTERNAL to the MARC4's internal RAM area (i.e., in a RAM module addressed over a port).



12530

Figure 11. Symbol table display for variables

Search for a ROM Symbol <Alt–F2>

Pressing **<Alt–F2>** enables you to disassemble the ROM starting at a particular location. The screen for the ’View’ command shows a pop-up dialog box prompting you to enter either the **qFORTH** word’s name or a hexadecimal ROM address. The corresponding ROM location will be displayed in the first line of the window. The pop-up dialog box will disappear after you have entered the desired name and pressed the **<Enter>** key. If, however, you wish to cancel the command after having invoked it, press the **<Esc>** key and the box will disappear without any changes to the ROM display. If the symbol or the address that you entered was not found, an error message will be displayed.

If you are in the trace data window and have recorded the instruction cycles, pressing **<Alt-F2>** opens a similar window as described above. It will not only searche for ROM addresses that match the entered string, but also for instructions in the trace data window.

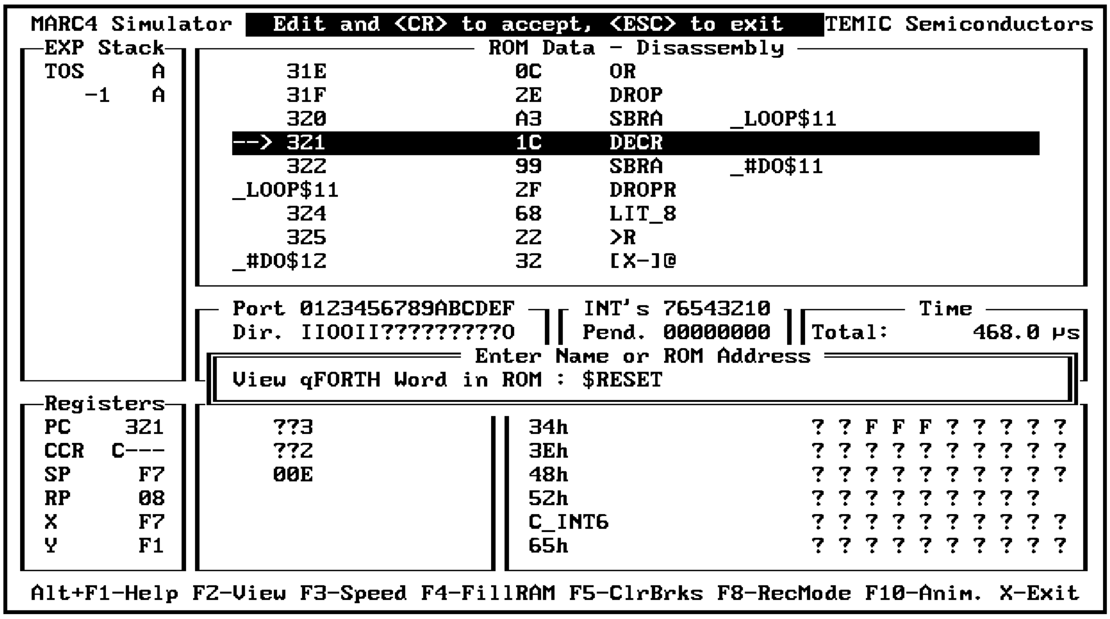
To repeat the search, just enter **<Ctrl-L>**. The next location that matches the entered string will be displayed, and you can repeat this procedure until the string can no longer be found.

Changing the Processor Speed <Alt–F3>

This command opens a pop-up window in the upper left corner of the screen that enables you to change the MARC4’s internal RC oscillator frequency. Follow the instructions given in the message line to change the frequency.

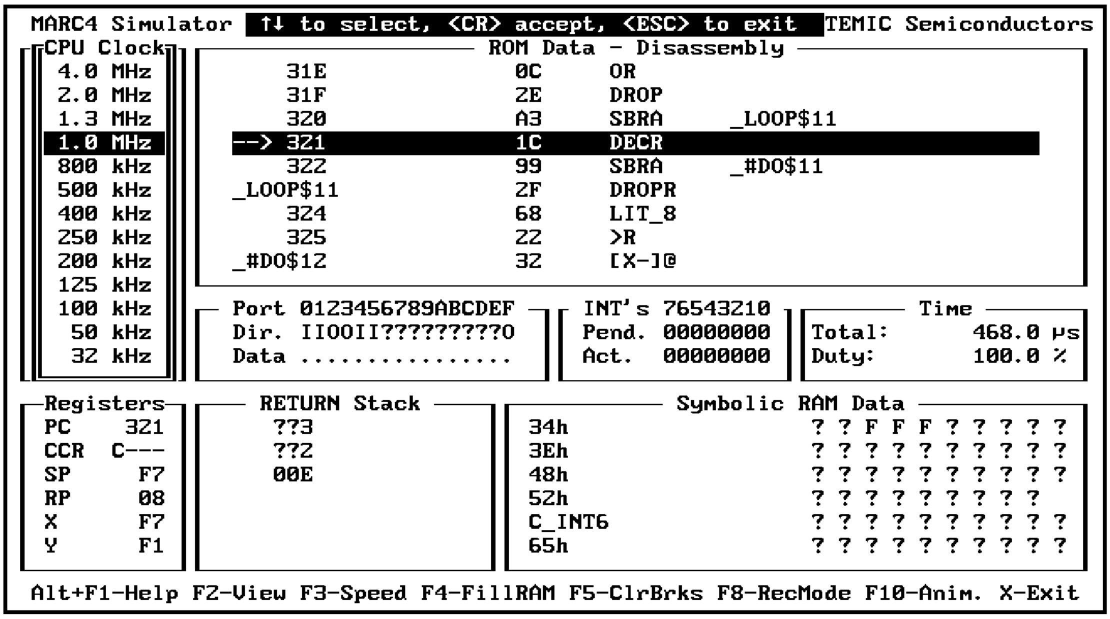
The default value is set to 500 kHz which corresponds to an instruction cycle time of 4 µs.

The modified frequency value will be stored in the configuration file when leaving the simulator.



12531

Figure 12. Pop-up box caused by the ‘View’ command



12532

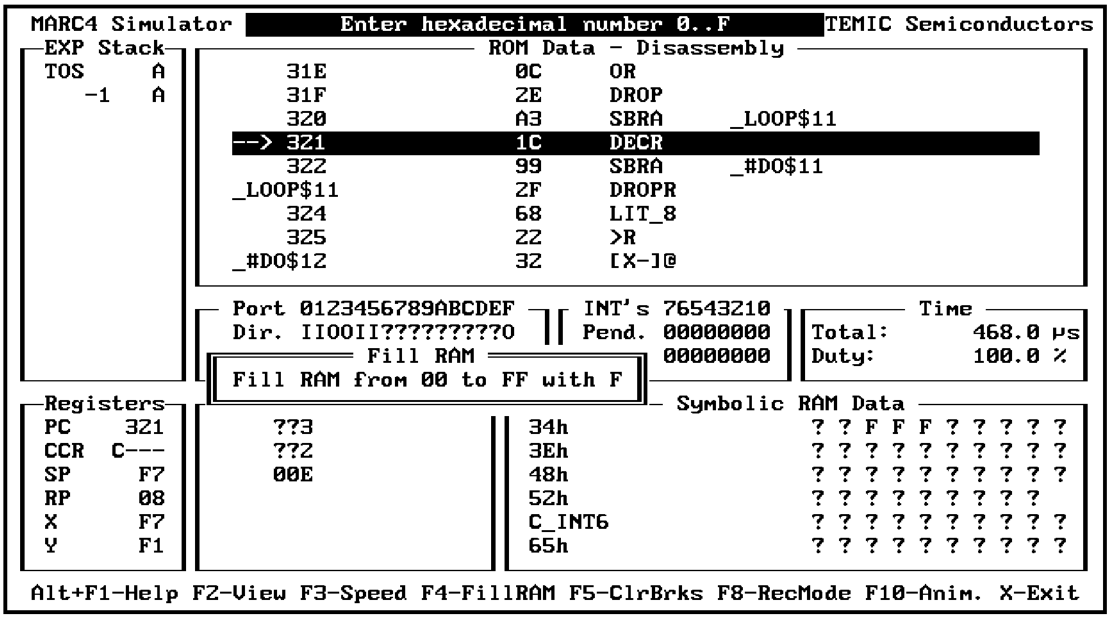
Figure 13. Changing the instruction cycle time

Reset Elapsed Time <Alt–F6>

Pressing **<Alt–F6>** enables you to reset the time to zero whenever you have stopped the execution of the program. This makes it easier to examine the time that passes until the next breakpoint occurs.

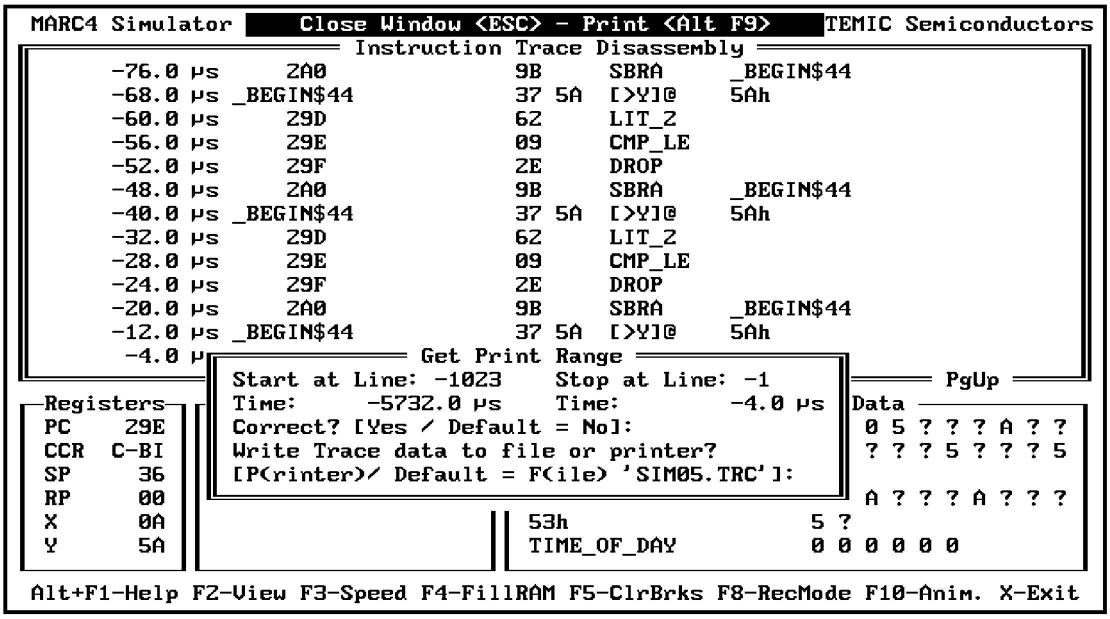
Fill a Section of RAM <Alt–F4>

This command opens a window that allows you to fill the RAM from one address to another with a hexadecimal value. To change single values, use the ’Edit’ command. To set the complete RAM area to the value ’undefined’, use the ’Reset’ command.



12533

Figure 14. Initialization of a RAM area



12534

Figure 15. Printing the trace memory data

Printing the Contents of RAM, ROM or Trace Memory <Alt–F9>

If the trace data window has been opened and you have recorded the instruction cycles, a window will be displayed where you can enter the first and the last line of the trace data window that will be written to the printer or to the file ’SIM05.TRC’. The time that corresponds to the entered lines is also displayed in that window. Enter ’Y’ if the line numbers you have entered are correct. Afterwards, select whether the data should be written to the printer or to the file ’SIM05.TRC’ which is the default. The pop-up window will be closed and the trace data will be printed or written to the file. If the line numbers are not correct, type ’N’ or press the **<Enter>** key and open the dialog box again.

If you have recorded the subroutine entries, all the lines in the trace data window will be printed.

Note: Whenever you are using this function, be sure that your printer is switched on and set to ’ON LINE’.

Leaving the MARC4 Software Simulator <Alt–X>, <Alt–F7>

Entering **<Alt-X>** or **<Alt-F7>** will result in all windows being closed. In the integrated development system, exiting the software simulator will return you to the SDS main menu while in the command line version, it will return you to DOS.

While exiting all the entries you have made in the breakpoint, the processor speed and the trace mode window saved on the file ’SIM05.CFG’ together with the name of the simulated file. The next time the simulator is invoked, this setup will be used as default configuration. In the case of a new or changed file which is to be simulated, a warning message will be displayed in the message line.

5.5 First Steps

5.5.1 Moving About Within the Simulator Screen

The arrow keys enable movement to windows and within the ’current’ window. The **<Left>** arrow key moves clockwise around the windows, **<Right>** moves counterclockwise to select a window. Once selected, the window frame is highlighted to indicate that it is the ’current’ window. The contents within the active window can be altered using the ’Edit’ function.

You may use the arrow keys **<PgUp>**, **<PgDn>**, **<Up>**, **<Down>**, **<Home>** and **<End>** to browse through the ’current’ window.
If you want to print the contents of the RAM, ROM or trace data window, press the **<Alt-F9>** key.

5.5.2 Modes of Operation

The software simulator enables you to test your MARC4 program in four operation modes:

- Single step
- Step over CALL
- Run
- Animation

Step Mode <F1>

The step mode operates one instruction each time the **<F1>** key is pressed. You will be able to view any changes to the MARC4’s internal registers, data, ports and interrupt registers following the execution of a single instruction.

Step over CALL <F2>

Stepping over subroutines is used to execute a known instruction sequence (including all nested subroutines) until the current return stack level is reached again.

Run Mode <F3>

When using the run mode, the program will operate at full speed until a breakpoint is located. The user can stop the simulation by pressing any key or the processor enters the sleep mode without any interrupt pending.

Animation Mode <F10>

The animation mode executes your program from the beginning to the end but at a much slower speed than the run mode. This offer you more time to view and understand MARC4’s internal workings.

Working with Breakpoints <F5>

By using the simulator’s breakpoint facilities, you can set up to nine software breakpoints. These program supervisors enable you to trace the application program’s execution at a defined address, detecting stack over-/underflows or halting on a specific I/O event.

To reset all defined breakpoints to their initial state (’**Off**’ for PC and I/O breakpoints, ’**Halt**’ for the others), use the **<Alt-F5>** function key.

5.5.3 Simulation of Real-time Events

Hardware Interrupts

The external hardware interrupts can be activated by entering them into the input polling file. See the example in the section on input polling files (4.5.4).

To enter external hardware interrupts during simulation, you can use the keys **<Alt-0>** ... **<Alt-7>** which correspond to hardware interrupts with the priorities 0 to 7. After pressing one of the keys, the simulation will stop, one more step will be performed and the resulting interrupt will be pending and can be seen in the interrupt status window.

Software Interrupts

A software interrupt occurs when the MARC4 instruction **SWI** is found within the program module being executed by the simulator. The **SWI0** .. **SWI7** instructions tell the MARC4 CPU to transfer the program control to the interrupt service routine known as **INT0** .. **INT7** (where 0 .. 7 are the priority numbers from a low priority of 0 to the highest priority of level 7). Any higher priority level can interrupt a lower level as long as the interrupt enable flag in the CCR is set.

Prescaler Module Programming

Usually, the integrated programmable prescaler is driven by an external 32.768-kHz quartz crystal. The prescaler module powers up in the reset condition and offers two time interval interrupt sources. The table below illustrates the selectable interrupt frequencies for the **M43C505** or **M44C636**. A similiar table is available for the M43C200/201.

```
\             Prescaler selectable interrupts
\
\ Control codes    Int. priority      Interrupt time
\                  (8: masked)        (0: disabled)
\
      F               8                   0
      E               6                   0
      D               4                 244.14
      C               6                 976.56
      B               6                3906.25
      A               6               15625
      9               6               62500
      8               8                   0
      7               5                7812.50
      6               5               15625
      5               5               31250
      4               5               62500
      3               5               125E3
      2               5               250E3
      1               5               500E3
      0               5                1.0E6
```

Figure 16. Prescaler control file ’SIM05.DAT’

Table 2. Prescaler interrupt frequency programming (M43C505/M44C636)

Control Code	Priority	Interval Time	Interrupt Frequency
F	none	Reset & hold complete prescaler	
E	(INT5)	INT6 disabled, INT5 still active	
D	INT6	244.14 sec	4096 Hz
C	INT6	976.56 sec	1024 Hz
B	INT6	3.906 msec	256 Hz
A	INT6	15.625 msec	64 Hz
9	INT6	62.5 msec	16 Hz
8	(INT5)	Reserved, production test mode	
7	INT5	7.81 msec	128 Hz
6	INT5	15.625 msec	64 Hz
5	INT5	31.25 msec	32 Hz
4	INT5	62.50 msec	16 Hz
3	INT5	125 msec	8 Hz
2	INT5	250 msec	4 Hz
1	INT5	500 msec	2 Hz
0	INT5	1 sec	1 Hz

Table 3. MARC4 interrupt priority list (M43C505)

Priority Level	Functional Description
$RESET	Software and hardware initialization after POR
INT7	External hardware interrupt, negative edge triggered
INT6	Prescaler interrupt #2
INT5	Prescaler interrupt #1 (for real–time clock applications)
INT4	Port 5 interrupt
INT3	Software interrupt **(**SWI3**)**
INT2	External hardware interrupt, negative edge triggered
INT1	Software interrupt **(**SWI1**)**
INT0	Software interrupt **(**SWI0**)**

After writing a control code to the prescaler port at address 15, the corresponding interrupt will occur periodically until the interrupt is disabled either by the program itself or by a hardware reset (simulated by the 'Reset' command).

The software simulator receives its information about the priorities and the times of the programmed interrupts from a file named **SIM05.DAT**. If this file is missing in your MARC4 system directory, an error message will be displayed.

An example of the file **SIM05.DAT** as used for the simulation of the **M43C505/M44C636** is shown in figure 16.

Notes: Every comment line in the file has to start with a '\' in the first column.

The file contains the control codes that can be written to Port 15, their interrupt priorities and times (in µs), separated by at least one blank.

Priority '8' means: 'No interrupt' or 'Reserved'.

Time interval '0' corresponds to 'Interrupt disabled'.

The **M43C505** requires a '1' at the end of control code 7. This is to indicate that this prescaler interrupt will also occur if an interrupt of a higher priority (i.e. INT6) was programmed first. For correct operation of the prescaler, the INT5 tap should be programmed before the INT6 tap.

If you want to simulate a MARC4 version other than the **M43C505** (with an external 32.768 kHz quartz crystal attached), just create a new file containing the prescaler table of that version and copy it to the file **SIM05.DAT**.

```
\                Prescaler selectable interrupts
\
\ Control codes    Int. priority     Interrupt time
\                  (8: masked)       (0: disabled)
\
     F                 8                   0
     E                 4                  64
     D                 4                 128
     C                 4                 256
     B                 4                 512
     A                 4                1024
     9                 4                2048
     8                 4                4096
     7                 4                8192
     6                 4               16384
     5                 4               32769
     4                 4               65536
     3                 4              131072
     2                 4              262144
     1                 4              524288
     0                 4             1.048E6
```

Figure 17. Prescaler control file 'SIM05.DAT' for M43C200/M43C201

5.5.4 I/O Simulation

I/O through the Port Status Window

During program execution, the port status window is used to show the state of the MARC4 input and output ports.

The port addresses are shown in the top line of the window. If the direction of a port is set to input, this is represented by an 'I' in the direction line of the window in the column of the addressed port. If the direction is set to output, is represented by an 'O'. If any of the given 16 I/O ports does not exist on the simulated MARC4 or has not been addressed yet, a '?' will be displayed.

The data line of the window shows the hexadecimal data that is written to or read from the ports. If there is no data at the port, a dot is displayed in the data line. If the simulator receives an **IN** instruction, it tries to read the requested data from an input polling file. If there is no such file or no data for that port at the given time step, it prompts you to enter the data as a hexadecimal value. Just follow the instructions given in the top line of the screen.

Input Polling Files

The input polling file enables you to introduce real-world data into your program. Whenever there is an **IN** instruction in your code, the simulator tries to read the requested data from the polling file. It is also possible to simulate hardware interrupts using this kind of 'script' file. The input polling file uses the file extension **POL** and has the same filename as the project file.

The input polling file contains three columns

- The absolute time (in µs) from the start of the program at which the event occurs
- The port address or interrupt priority
- The corresponding data value

The following example of an input polling file explains how to use it:

```
\  Input polling file 'EXAMPLE.POL' for project
\ 'EXAMPLE.HEX'
\
\        Time        Address       Data
          100        A             7
          300        0             3
         2200        2
         3400        5             5
         5700        1             4
\  END
```

After 100 µs from the start of your program, a 7 occurs at Port A, after 300 µs, a 3 appears at Port 0, after 2200 µs, a hardware interrupt of the priority 2 should be simulated, after 3400 µs, a 5 at Port 5 and after 5700 µs, a 4 at Port 1 should be set.

Notes: Every comment line must begin with a '\' in the first column of the line.

Normal data lines contain a time column in µs (the time must advance for each new line) followed by the port address and data as hexadecimal values separated by at least one blank.

Lines that do not contain a data column will be interpreted as hardware interrupts with the priority given in the address column.

Do not forget the '\ **END**' in the last line of the file.

In case an **IN** instruction occurs while the program is running with port address A after 50 µs, the simulator will prompt you for the data because it cannot find any valid data at Port A at this time. In this case you must enter it in the port status window. If there is however, a read from Port A again at 4000 µs, the simulator will still read the 7 at Port A which it already received at 100 µs. Unless you entered a line with new data for Port A into the input polling file at a time < 4000 µs, this data is still valid.

Restrictions:

The MARC4 software simulator does not support the additional interrupt features of the **M43C505/M44C636** family (maskable external interrupts, interrupt driven keyboard at Port 5) nor any timer/counter functions of other derivates.

I/O Capture Files

An I/O capture file is used to record all I/O activities to and from the peripheral modules. The I/O event number is recorded along with the time and type of activity.

In figure 18, the first registered activity on the MARC4's peripheral bus was when the data value F was written to Port F which resets the prescaler. The next activity was when the data value 6 was read from Port 0. The eighth event that caused I/O bus activity was a **SWI** followed by an interrupt acknowledge at the time 964 µs. The next activity is a 7 written to Port 0, followed by a **RTI** after 1000 µs. Whenever there is an interrupt acknowledge or **RTI,** you can examine the content of the active (AT) and pending 'task' (PT) interrupt registers of the MARC4. The I/O capture file is named with the same name as the project file which is being simulated, the created file will have the extension **IOB**. To generate an I/O capture file, you must press the **<Alt-F8>** key and select the **I/O cycle trace** in the pop-up window before program execution.

5.5.5 Simulation Restrictions

The MARC4 software simulator is the ideal design and test platform for applications where target hardware is not available until programming has been completed. However, for testing applications where real-time factors are important, the simulator will quickly prove to be insufficient since a speed factor difference between the MARC4 simulator and the emulator can be as much as a factor 1000 slower than the software simulator. Another area in which the simulator will prove restrictive is in testing keyboard controlled applications where a large polling file is required. For common 4 x 4 matrix keypads, the input polling file may still be of a manageable size. However, for larger interrupt driven keyboards, the simulation times will be greatly lengthened due to the time it takes to handle the larger input polling file.

IO Nr.	Time	PC	Port	Data	Instr	Port 0123456789ABCDEF
1	28.0 µs	00E	F	1111	OUT	...............F
2	36.0 µs	010	0	0110	IN	6..............F
3	84.0 µs	239	F	1111	OUT	6..............F
4	116.0 µs	241	0	0000	OUT	0..............F
5	124.0 µs	243	1	0000	OUT	00.............F
6	164.0 µs	241	0	0001	OUT	10.............F
7	932.0 µs	205	0	1111	OUT	F0.............F
8	960.0 µs	24E	1111	1111	SWI	
	964.0 µs	250	Acknowledge		INT7	AT:10000000 PT:11111111
9	988.0 µs	1E3	0	0111	OUT	70.............F
10	1000.0 µs	1E5			RTI	AT:00000000 PT:01111111
	1004.0 µs	251	Acknowledge		INT6	AT:01000000 PT:01111111
11	1036.0 µs	1C4	0	0110	OUT	60.............F

12535

Figure 18. Example of an I/O capture file

6 Emulator

6.1 Introduction

The PC-bus compatible MARC4 emulator board enables real-time emulation of customer programs and highlighting any timing problems which would not be visible through simulation alone. Time-critical interrupt procedures may be measured using real external hardware events.

Another function of the emulator is to enable the ASIC designer and/or customer to prototype unavailable peripheral modules with standard CMOS logic or FPGAs.

The PC-resident emulator software controls the loading of the breakpoint conditions and provides the user interface into the MARC4 Software Development System. The MARC4 emulator can stop and restart a program at specified (break-)points during execution, making it possible to examine and modify the memory contents and those of various registers during program execution. It is also useful in analyzing the executed instruction sequences and the corresponding I/O activity. The execution time of the emulated program and its duty cycle can be monitored by the user.

The MARC4 emulator board is a universal development tool because its operation is independent from the peripheral module configuration of any specific or standard MARC4 microcontroller family member and the customer's application hardware.

Note: The information contained herein is provided under the assumption that the software and hardware described here has been tested in combination with the target MARC4 at system-clock frequencies as high as 2 MHz. No warranty or guarantee can be made if you run the devices at system-clock frequencies higher than 3 MHz.

6.2 Features

- Emulator board uses the PC's plug-in, 8-bit wide ISA bus interface
- Multi-window user interface with mouse support, similar to the MARC4 simulator
- Context sensitive on-line help feature
- Expandable emulation RAM for download of customer programs
- 4K × 32 bit deep execution trace memory capturing
  - the ROM address lines [PC11 .. PC0]
  - the ROM bank switch lines
  - the demultiplexed I/O bus [Port Address, Port Data and Interrupt Request]
  - the I/O control lines (Read, Write)
  - the NON_SEQ instruction control signal
  - 4 user-definable, application-specific signal inputs [Trace 0 .. Trace 3].
- Software-supported, unlimited execution time and duty cycle measurement
- Automatic configuration setup store/restore function
- External clock generation, so the frequency can be changed for worst-case evaluations
- Examination of pending and active interrupts
- Examination and editing of all internal registers, the RAM contents as well as the ROM code
- Breakpoints based on the e3400EVC with ROM address, RAM access, I/O and interrupt activity supervisor functions
- Sequential, time dependent and post trigger breakpoint features.

6.3 Getting Started

Before starting the emulator, make sure that the following files are available in MARC4 base directory:

EMU3.EXE, EMU3.CFG, EMU3.HLP, EMU4.HEX

- To start the emulator with the SDS2 environment: Check that the correct directory path for the emulator has been entered in the setting window "Directories" (see Installation Guide). Press the function key **<F8>**.

Note: On starting the emulator, the program file which has been defined as the project file in the SDS2–IDE will be loaded by default.

- To start the emulator as an independent program: If the MARC4 directory has not been included in the AUTOEXEC.BAT search path, switch to the MARC4 system directory and enter the following command:

  C:\MARC4>EMU3 PROGRAM\TIMER

  The argument PROGRAM\TIMER is the path and file name of an example project which has to be emulated.

- First of all you have to set up the correct target device available on the target application board by using **<Shift–F7>**

While exiting, the name of the emulated file will be saved on the file "EMU3.CFG". In the case of a new or changed file which has to be emulated, a warning message will be displayed in the message line.

If the MARC4 emulator hardware is in any way defect, you will get an error message from the built-in test program. The devices that may be faulty will be listed and their locations can be found in figure 4 of the installation guide. If this situation occurs, please contact your local TEMIC sales person. As far as this description is concerned, we presume that the emulator board is functionally correct.

6.4 Using the Emulator

6.4.1 Emulator Screen

The look and feel of the MARC4 emulator is simular to the MARC4 simulator display screen (see figure 1). The screen is subdivided into windows which show the current status of the emulated MARC4 core.

The different standard windows provide the following information:

ROM data disassembly
: displays the ROM code to be executed. The line that is currently highlighted has got an arrow pointing to the ROM address which is the same as the PC.

Expression stack
: lists the contents of the data or parameter stack

MARC4 registers
: displays the status of the 3 internal flags (CCR), the contents of the program counter and the RAM address registers

Return stack
: lists the addresses pushed onto the return stack

RAM data
: displays a portion of the internal RAM locations

Interrupt status
: displays the status of the internal interrupt registers

Time
: displays the total execution time, the active time and the percentage of time the processor was active

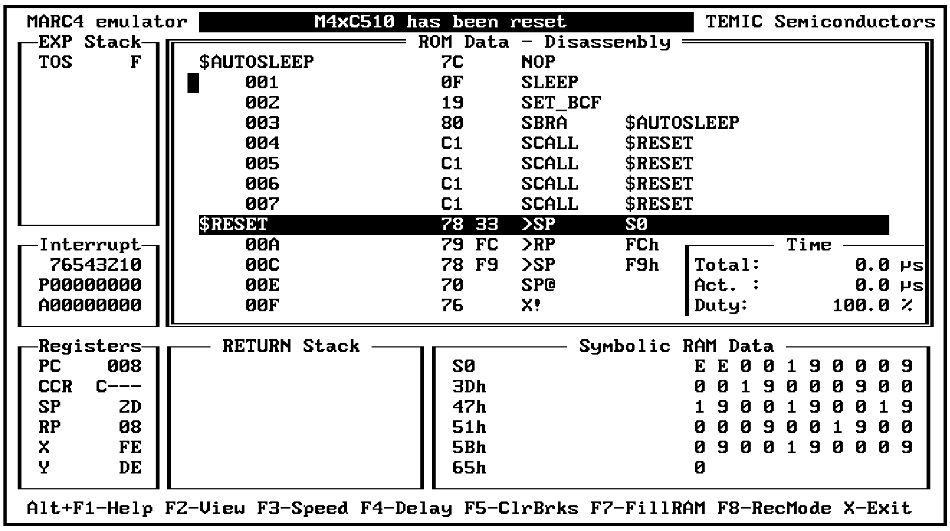
The message line at the top of the screen contains the display status, error or help information. The active line at the bottom of the screen describes the function keys used by the emulator. This line can be toggled by pressing the <**Alt**>-key. Furthermore, there are some additional windows which are displayed by pressing a specific function key. They will disappear again after pressing the <**ESC**>-key. These windows and their function keys will be described later.

Select a window:

Keyboard: To change the active window, please use the arrow keys <←>**,** <→>.

Mouse: Move the mouse cursor over the window and press the left mouse button.

By pressing the <**Pos1**>-key on your keyboard, the display of the currently active window will move to the top address location, i.e., the ROM disassembly window to the $AUTOSLEEP address. The <**End**>-key is the corresponding counterpart. The scroll page commands also work on the active window.



12536

Figure 1. MARC4 software emulator screen

6.4.2 Emulator Window Description

Note: Most of the emulator windows provide the same information as in the MARC4 simulator. This is because only the little differences will be described in the following description.

Interrupt Status Window

The MARC4 can handle up to 8 prioritized interrupts which can be generated asynchronously from either on-chip modules, external sources or synchronously from the CPU itself (software interrupts).

The transmission of the interrupts occurs over the I/O bus. The emulator enables the user to view the PENDING (P) as well as the ACTIVE (A) interrupts.

Time Window

The time window displays the total elapsed and the active time.

The active time is the time the MARC4 is operating and not in Sleep mode. The total elapsed time is the sum of the active and the Sleep mode time based on the PC's internal timekeeping hardware.

The time window is updated every second. The active time shown is the execution time required by the MARC4 processor when running your program at a specific instruction cycle time.

The time window also displays the percentage of time the processor was active, the so called 'duty cycle':

$$\text{Duty cycle} = \frac{\text{Active time}}{\text{Active time} + \text{Sleep time}} \times 100\ \%$$

6.5 Emulator Commands

6.5.1 Command Keys Summary

Key	Function	Short Description
F1	Step	Single step
F2	ROMbreak	Set ROM address break on top line
F3	Run	Execute emulator until breakpoint or user break
F4	Reset	Emulator (re–) initialization
F5	BrkPts	Breakpoint set–up function
F6	Load	Read in a binary object file
F7	Symbols	Display symbol table information
F8	Trace	Display recorded trace data
F9	Edit	Window data editor
F10	Source code	Display source code
Alt–F1	Help	Pop-up help window
Alt–F2	View	View ROM data from start address
Alt–F3	Speed	Change MARC4 processor speed
Alt–F4	Delay	Set clock delay, if VDD < 1,6V
Alt–F5	ClrBrks	Reset all earlier defined breakpoints
Alt–F6	Toggle_IF	Toggles interrupt enable flag status
Alt–F7	FillRam	Fill a section of RAM
Alt–F8	RecMode	Select mode for trace memory recording
Alt–F9	Print	Print the RAM, ROM or trace memory contents
Alt–F10	Animation	Continous single step mode
Alt–X	Exit	Exit emulator, return to environment or DOS
Shift–F1	Show release	Show version number and date of creation
Shift–F3	Setup	Show current emulator configuration
Shift–F4	ROMBreak	Display ROM address break
Shift–F5	RAMBreak	Display RAM access break

Key	Function	Short Description
Shift–F6	IOBreak	Display I/O breaks
Shift–F7	Target	Select target chip
Shift–F8	Ports	Display port window
Shift–F9	Memory	Display memory window
←,→		Change the current window
Pos 1/Home		Set current window of its top address
End		Set current window of its last address
Page ↑ Page ↓		Scroll page

6.5.2 Command Description

Single Step <F1>

This key enables the user to walk through the program code displayed in the ROM data window one instruction for each **<F1>** selection. The message '1 instruction executed' is then displayed on the screen's message line.

The internal RAM and register contents and all other windows will be updated and displayed on the screen. For single step of lower priority interrupt service routines, use the 'Toggle I-Flag' function key **<Alt-F6>**.

Set ROM Address Break <F2>

This key allows the user to set a breakpoint directly in the ROM data window without opening the breakpoint selection window and selecting the required ROM address.

To set a breakpoint, activate the ROM window and use the cursor keys to scroll the selected ROM address up to the top line of the window and then press the **<F2>**–key. The top line of the ROM disassembly window will be highlighted red and the program execution will stop one instruction after the choosen ROM address.

To delete such a breakpoint, scroll the highlighted ROM address to the top line and press the **<F2>**–key again.

Note: If you want to set a ROM address break, it is much easier to use the mouse. Move the mouse cursor over the desired ROM address and press the left mouse button.

Program Execution in Real-Time <F3>

The emulator will execute the application program in real-time until a breakpoint condition is met.

Whenever the processor is active the message line will display 'Active Mode' in power down 'Sleep Mode' will be displayed. You can stop the execution of the program by pressing any key. The message line will then display 'Stopped at User Break' or 'User Break from SLEEP', depending on whether the processor was active or in Sleep mode.

You may continue the execution of the program by pressing **<F3>** again, but it cannot be guaranteed under all circumstances that the program execution will continue correctly, especially if there is an interrupt pending or the processor was in SLEEP at the time the break occurred. So the proper solution is to reset the MARC4, to clear or change the breakpoint conditions, and to start the program execution again.

Emulator Initialization <F4>

This command resets the processor and all the screen display windows. Because of the reset, the program counter (PC) is set to ROM address 008h, the RAM contents will be lost, the RAM address registers are undefined and the I-flag is reset.

Note: The breakpoints will NOT be reset to their default values.

The message line will display 'M4xCxxx has been reset', depending on the selected target chip (see **<Shift–F7>**). If the message line is displaying 'Unable to reset processor', try pressing **<F4>** once more. If that does not help, exit the emulator and enter the emulator menu again.

Set Breakpoints <F5>

The command **<F5>** will display the breakpoint selection window on the screen. The emulator permits the programmer to test the software by setting breakpoints in his code, on accesses to specific I/O ports or data areas. Breakpoints enable the user to stop the execution of his program whenever a user definable condition or sequence is met.

Note: The MARC4 processor uses an internal 3-stage instruction pipeline which therefore avoids breaks of the program execution during an instruction or one instruction after an EXIT, RTI, TABLE, SCALL or SBRA instruction as well as ROM breakpoints on the second byte of a two byte instruction.

In the breakpoint selection window shown in figure 2 you may choose between one of six different breakpoint categories:

- Breaks on ROM addresses
- Breaks on RAM accesses
- Breaks on I/O activities
- Break after a specific time interval
- Sequential trigger
- Posttrigger setup

Depending on the selection of one of these six items a new window will be opened.

ROM Addresses <Shift-F4>

Set up to four different breakpoints on ROM addresses which the program may pass during the code execution.

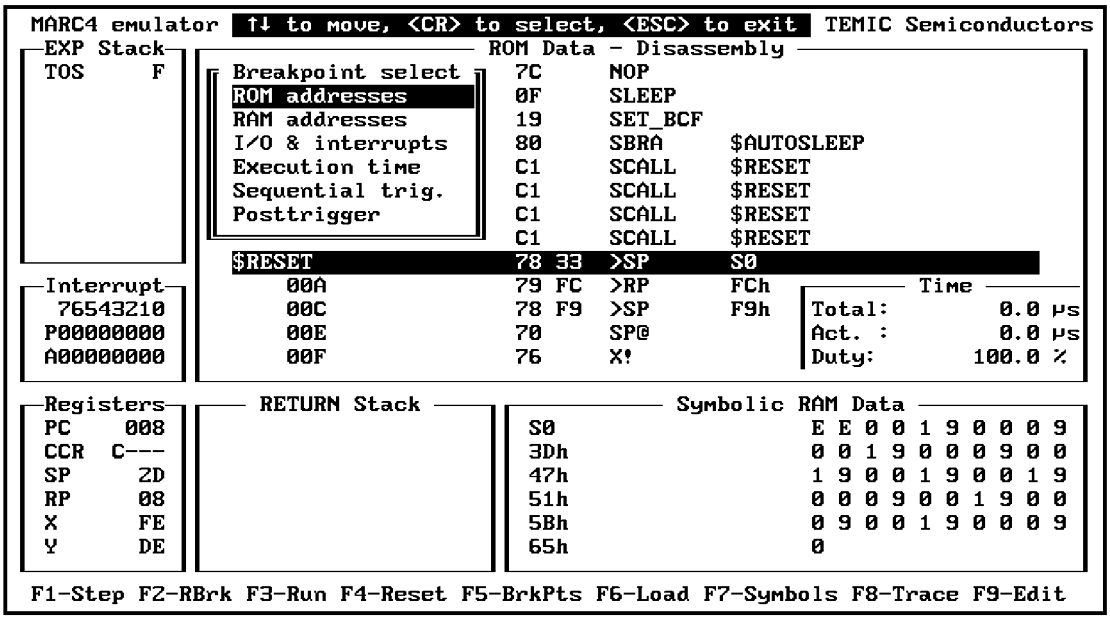
The addresses can either be entered as a hexadecimal number or as a symbol. To activate or to inhibit breakpoints there is a choice between three options:

Do not break
Program execution will not stop if breakpoint condition is met. The count field will not be considered in this state.

Stop program execution
Emulation will stop if the defined breakpoint condition is met and the pass counter (which counts values between 1 and 255) reaches zero.

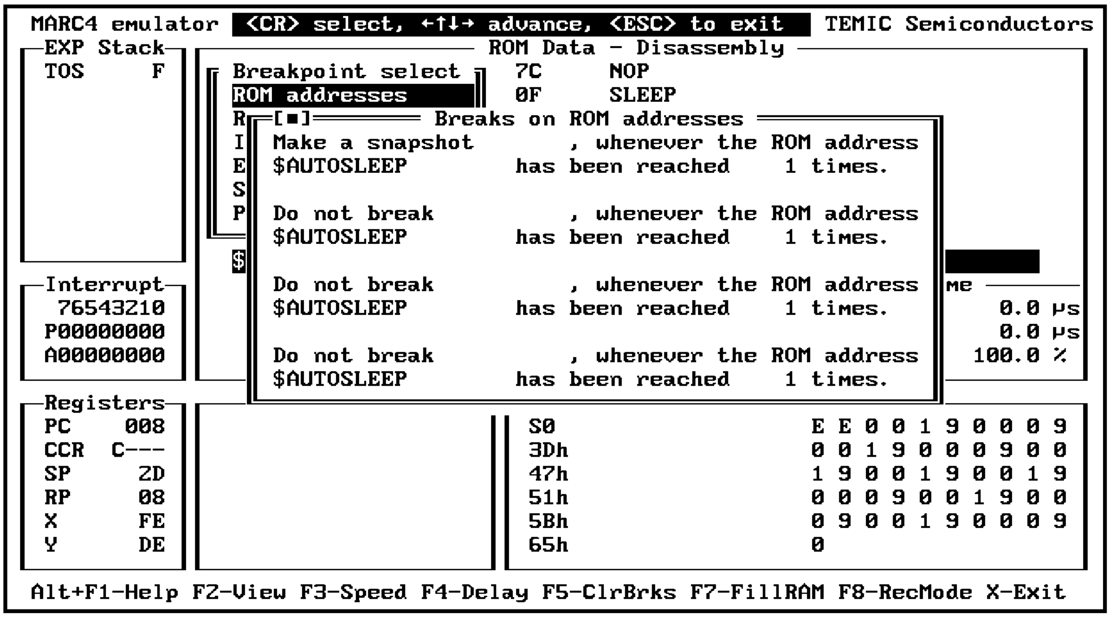
Make a snapshot
Snapshot makes a screen update every time the specified condition is passed during program execution. The count field will be considered in this mode too.

A breakpoint may also be set using **<F2>** or the left mouse button in the ROM window.



12537

Figure 2. The breakpoint selection window



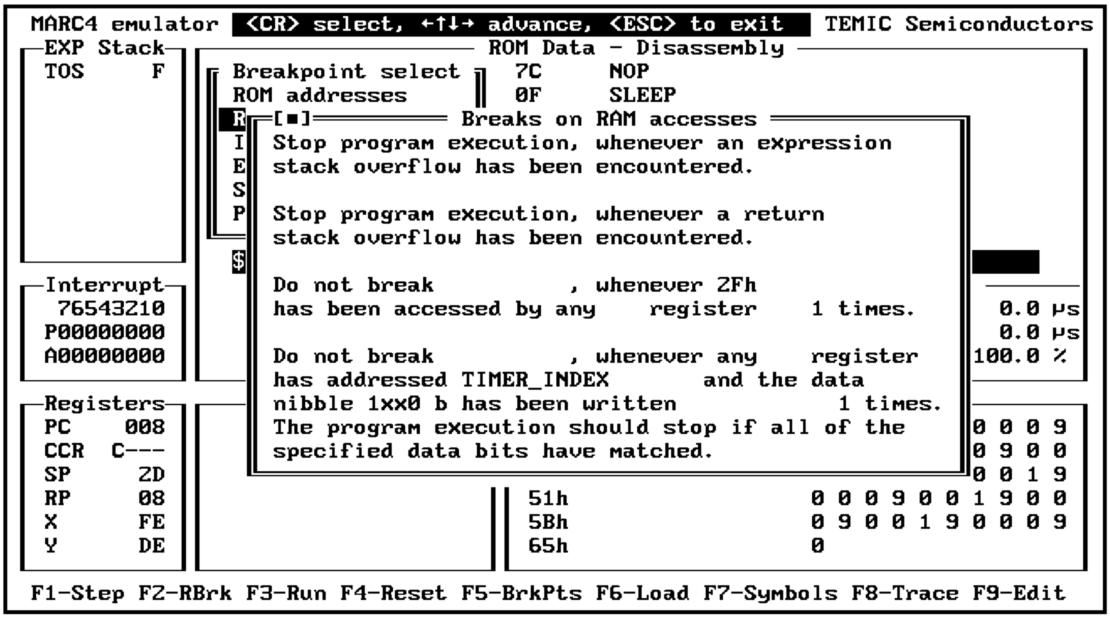
12538

Figure 3. Breakpoints on ROM addresses

Breakpoints on RAM Accesses <Shift-F5>

Set up to four different breakpoints on RAM accesses. These are two RAM breakpoints to stop execution whenever there is an expression stack or a return stack overflow. The third breakpoint is additionally characterized by a specific register which is used to access a RAM nibble. To detect an expression stack overflow this breakpoint has to be set to ’Stop program execution whenever S0–1’ (see hex value in symbol table) has been accessed by SP register 1 times. The fourth breakpoint supplementary allows the user to enter a (maskable) data nibble which should be read from or written to the specified RAM address.

The addresses can either be entered as a hexadecimal number or as a symbol. The toggling of the breakpoint state and the setting of an additional count value between 1 and 255 is identical to the ROM address breakpoints.



12539

Figure 4. Breakpoint on RAM accesses

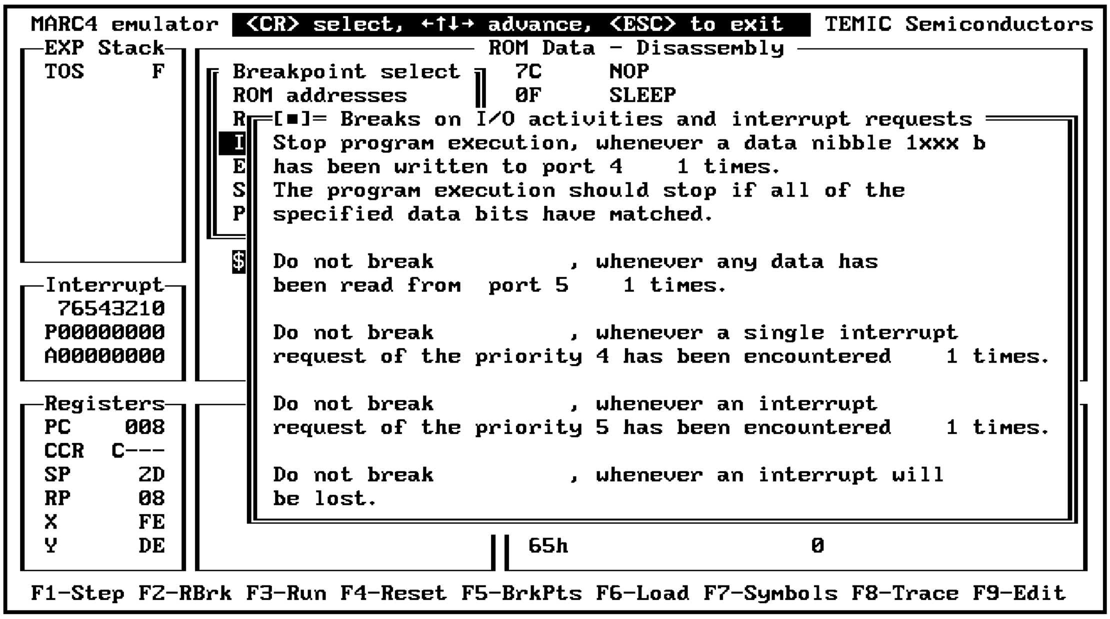


Figure 5. Breakpoint on I/O activities

Breakpoints on I/O Activities <Shift-F6>

Program execution will stop if any of the specified I/O conditions are met. It can be distinguished between reads from or writes to (hexadecimal) ports and interrupts of a given priority. Besides this, the user can enter a (maskable) data nibble which should be read from, or written to a given port address. Furthermore, a breakpoint can be set to examine whether any interrupt will be lost during the program execution, which is caused by a high priority interrupt overload.

The difference between the two interrupt breakpoints is that the first one matches only if the specified interrupt priority is discovered on the I/O bus as a single interrupt event and no other interrupt is transferred in the same cycle. The second one also matches if the specified priority occurs together with other priorities, e. g., when prescaler interrupt 5 and 6 occur in the same time slot.

The toggling of the breakpoint state and the setting of a count value between 1 and 255 is identical to the ROM address breakpoints.

The first four breakpoint conditions in this window are not independent on each other. If you arm the first breakpoint, the third will be turned off automatically and vice versa. This is true for the second and the fourth breakpoint condition too. Additionally, if you want to use the first breakpoint condition with a data nibble other than the don’t care ’xxxxb’, the following three breakpoints will all be turned off. The reason for this is the on-chip hardware implementation of the I/O breakpoint logic.

Masking of RAM and I/O Data Accessses

If you have selected the I/O or the RAM breakpoint window you are then able to mask the data that is transferred.

Example 1:

If you want to stop the program execution on the occurence of a data nibble ‘1xx0b’, that is for all nibbles whose highest bit is one, whose lowest bit is zero and whose other two bits are don’t care, just enter that nibble into the data section of the window and be aware that the program execution only stops if **all** of the specified bits have been matched. If, in this example, you had selected **any** of the specified bits you would get a break for all nibbles that would look like ’1xxx’ or ’xxx0’.

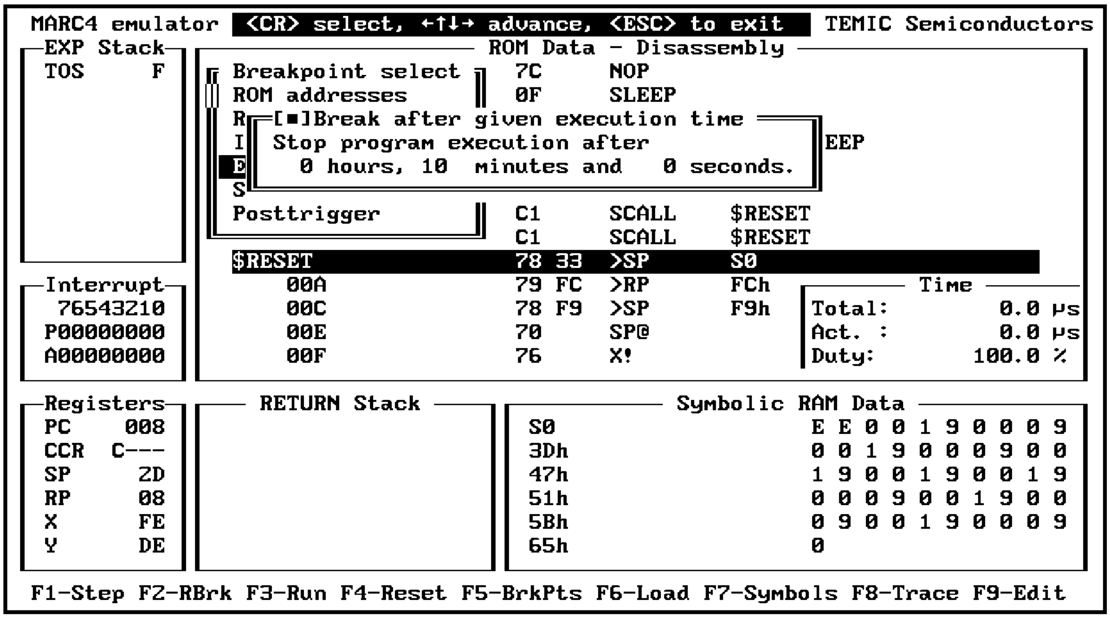
Example 2:

The mask is set to ‘x0x1b’.

If you have selected ’**any**’ you will get breakpoints on the occurence of the data nibble ’0**0**00’, ’0**0**0**1**’, ’0**0**10’, ’0**0**1**1**’, ’010**1**’, ’011**1**’, ’1**0**00’, ’1**0**0**1**’, ’1**0**10’, ’1**0**1**1**’, ’110**1**’, ’111**1**’.
If you have selected ’**all**’ you will get a breakpoint on the data nibbles ’0**0**0**1**’, ’0**0**1**1**’, ’1**0**0**1**’, ’1**0**1**1**’ only.

Break after Execution Time

This is used to stop the execution of the program after a given time between 1 second and 99 hours, 59 minutes and 59 seconds. To activate the time break, you have to toggle its state from ’Do not break’ to ’Stop program execution’. The break occurs after the total time relative to the start of the program and is based on the PC’s internal real-time clock facilities.



12541

Figure 6. Break after execution time

Note: Make sure that you increase the break time or clear the time break after this breakpoint occurred before continuing the emulation.

Sequential Trigger

Sequential triggering allows the user to have a break after the occurrence of two predefined conditions. Only those breakpoint conditions can be sequenced that have been activated in the ROM-, RAM-or I/O breakpoint window before. There are two groups of breakpoints and the member of each group can be combined to each member of the other group to form a sequence.

Table 1. The two breakpoint groups for a sequential trigger

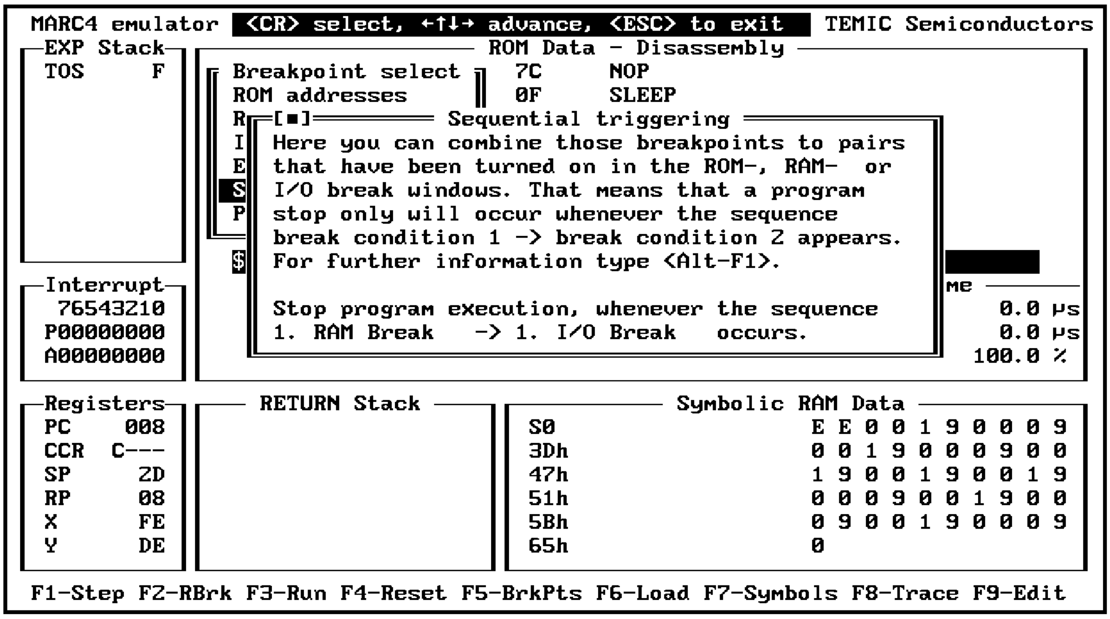
Group 1	Group 2
1. ROM Break	2. ROM Break
4. ROM Break	3. ROM Break
1. RAM Break	2. RAM Break
4. RAM Break	3. RAM Break
Lost INT Break	1. I/O Break
	2. I/O Break

E. g., ’1. ROM Break’ is the first breakpoint condition in the ROM break window, ’3. RAM Break’ is the third breakpoint condition in the RAM break window and ’Lost INT Break’ is the break on lost interrupts in the I/O break window.

’1. I/O Break’ is the first or third breakpoint condition in the I/O break window, depending on which one is turned on, ’2. I/O Break’ is the second or fourth breakpoint condition in that window.

To set up the sequential trigger condition, activate the corresponding breakpoints first, then arm the sequential trigger, move the cursor to the first breakpoint condition and select one of the possible entries by pressing the **<Enter>** key. Then do the same for the second breakpoint condition.

Note: Only one of the two sequential break conditions should have a count value greater than one (set in the corresponding breakpoint window). Otherwise, the sequential trigger may not behave as you expect. Count values greater than one of breakpoints that do not belong to the sequential trigger condition will automatically be reset to one. All other breakpoints will be disabled. Due to internal hardware restrictions, the ’I/O trace mode’ will not work together with a sequential trigger. A message will request you to turn off the I/O trace first if you are trying to turn on the sequential breakpoint facility.



12542

Figure 7. Sequential triggering

Posttrigger Setup

The posttriggering window enables you to stop program execution after a defined number of instructions following the occurrence of a breakpoint. This feature may be helpful if you want to examine the instructions in the trace memory that have been executed after a specified breakpoint condition, e.g., if an external interrupt request, occurs.

Load a New Program File <F6>

This command will display a window in which you are prompted to enter the name of the binary object file to be emulated. The file extension is assumed as *.HEX. If you can not remember the correct name and directory just press <**Enter**>. A separate file pick window will be opened and you may browse through the directory tree to find the new object file. This file and the accompanied symbol table file (if available) will be uploaded into the emulation ROM.

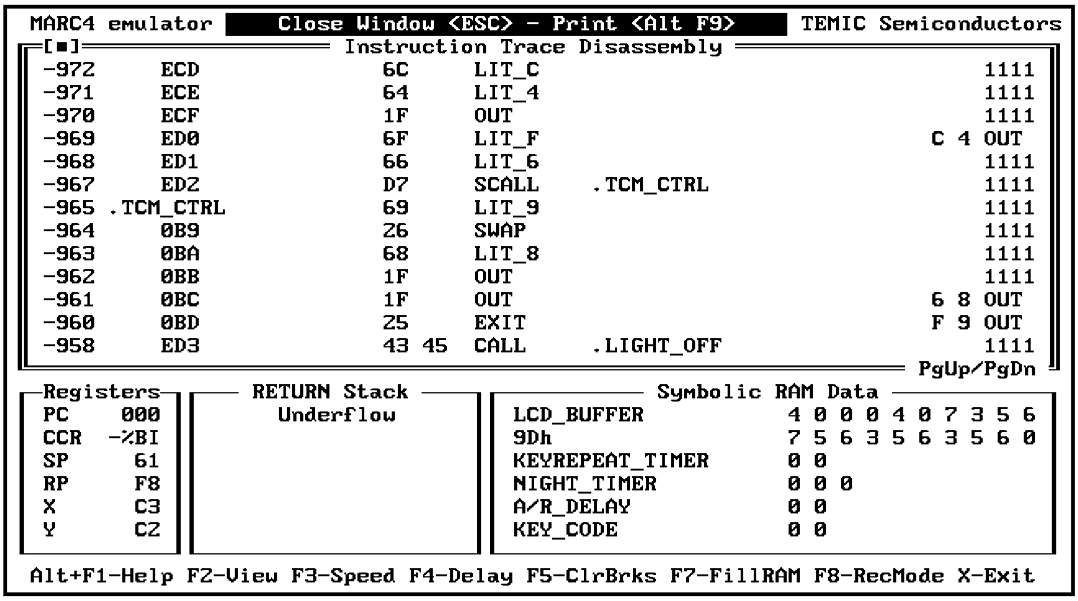
Note: All breakpoints will be turned OFF after a file read operation.

Display of Symbol Table <F7>

If you enter this command, a small pop-up window will be displayed in which you can choose between symbols of subroutines, variables or constants (used in the qFORTH application program) that should be displayed. If you press <**Enter**>, the chosen symbols together with their addresses and length or constant data will be displayed in the symbols window.

Display of Trace Data <F8>

This command will open a window that displays the recorded trace data for the instruction cycles, I/O cycles or the subroutine entries. The window can only be opened if a trace mode using**<Alt-F8>** has been set up before and either the instruction cycle, I/O activity or the activity statistics recording has been chosen and at least one instruction has been executed. Otherwise, an error message will be displayed on the message line. The trace window shown in figure 8, displays the disassembled contents of the 4K×32-bit deep ring buffer. The buffer works on the same principle as found in other emulators or logic analyzers.



12543

Figure 8. Instruction trace disassembly

Therefore, you can examine the last 4095 instructions (if there are so many) or look at the activity statistics of your program (depending on the chosen topic in the **<Alt-F8>** window).

If, e.g., the instruction cycles or the I/O activities have been recorded, the last line shown in the screen's window (offset is usually between -1 and -3) is the last executed instruction which was executed before the breakpoint occured. The negative numeric value shown on the left side is therefore a breakpoint relative memory address in the trace buffer.

By using either the **<Up/Down>** key or the **<PgUp/ PgDn>** key, you can view the disassembled contents of the trace buffer.

The next column shows the ROM address where the processsor has fetched the MARC4 opcode with its corresponding instruction mnemonic in the next two columns.

The following column will contain information about the program Branch/Call addresses, or the contents of ROM read operations (when a TABLE instruction has been executed).

The column on the far right is an I/O bus trace (similar to the I/O capture feature found in the MARC4 simulator). There, you can examine the external trace data lines (Trace 3 .. Trace 0), except when an external interrupt request, an IN or OUT instruction was performed which are also shown with their data and port address values.

Note: Internal interrupt requests like software interrupts will not be recorded in the trace memory.

The difference between instruction cycle and I/O activity recording is that in the latter case, only those instructions will be recorded that are joined to an I/O activity; that is an address, data or interrupt transfer. I/O activity recording will be disabled whenever you are executing single steps.

Note: Due to internal hardware restrictions I/O activity recording and sequential trigger will not work together. A message will warn you and the sequential breakpoint will be turned off.
Breakpoints that belong to group 2 of Table 3 and that have count values greater than 1 will be set to a count value of 1 during I/O activity recording.

If you have selected the activity statistics of your program with the **<Alt-F8>** key, the first column in the trace window shows the symbolic addresses the program jumped to at every CALL and SCALL instruction. The second column displays the number each symbol was called. The third column will show you the percentage the symbol in that line was addressed and the fourth column is a histogram of the percentages (displayed in green on a colour monitor). You can use the **<PgUp>**, **<PgDn>**, **<Up>**, **<Down>**, **<Home>** and **<End>** keys to browse through the trace window. If you want to print the contents of the trace window, you may press the **<Alt-F9>** function key. By pressing **<Esc>**, you will leave the trace window.

Edit Data in a Window <F9>

This command is for editing inside the currently selected window (the one with the bold frame around) which can be chosen by pressing the arrow keys or just clicking the left mouse button onto the window. When finished, you can leave the Edit mode by pressing the **<Esc>** key. To change the currently selected window, press the **<Left>** or **<Right>** key.

The RAM window’s data can be modified nibble-wise using hexadecimal values.

The ROM window’s contents are modified in the opcode column using the opcode bytes of the MARC4’s basic instruction set.

The MARC4 register window allows the usage of symbolic RAM and ROM addresses which can be assigned to the specified registers. In the expression stack window, only the top of stack value can be modified using a hexadecimal number.

Source Code Window <F10>

The source code window enables source-level debugging without the need to leave the emulator. It displays the qForth source file and the present emulator execution point. The following function keys support source-code scrolling: **<Page up>**, **<Page down>**, **<End>**, **<Pos1>/<Home>**

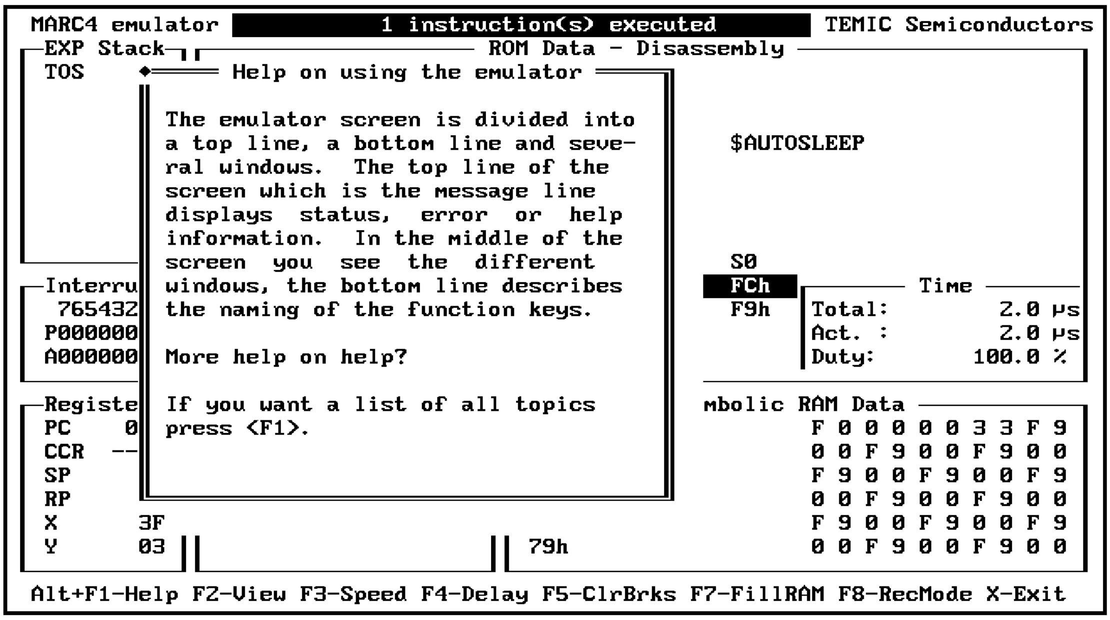
Note: This option requires a project HLL file generated during the compilation with the “hll linkage” compiler switch set.

Pop-up Help Window <Alt–F1>

This command will open a pop-up help window which displays information about the currently selected window. If the currently selected window is the ROM data window, general information about the emulator will be displayed. In the help window, you can get information about the highlighted and blinking topic if you just press **<Enter>**.

You can move the highlight bar among the cross-reference topics (written in yellow letters) by pressing the arrow keys , **<Up>**, **<Down>**, **<Left>** or **<Right>**. If a ‘PgUp/PgDn’ appears at the lower right hand corner of the window, you can display the previous or next help page by pressing **<PgUp>** or **<PgDn>**.

Press **<Alt-F1>** if you want to display the help topic most recently selected. Press **<F1>** if you want a list of all the topics described in the help menu (figure 9). To exit any help window just press **<Esc>**.



12544

Figure 9. The help window after pressing <Alt – F1>

Search for ROM Symbol, Address or Opcode <Alt–F2>

The command 'View' enables you to disassemble the ROM starting at a particular address. The screen for the view command shows a pop-up dialog box prompting you to enter either the qFORTH word's name or a hexadecimal ROM address.

The corresponding ROM location will be displayed in the first line of the window. The pop-up dialog box will disappear after you have entered the desired name and have pressed the **<Enter>** key. If, however, you wish to cancel the command after having invoked it, press the **<Esc>** key and the box will disappear without any changes to the ROM display. If the symbol or the address you have entered was not found, an error message will be displayed. To repeat the search, just enter **<Ctrl-L>** The next location that matches the entered string will be displayed and you can repeat this procedure until the string can no longer be found.

If you are in the trace data window and have recorded the instruction cycles, pressing **<Alt-F2>** opens a similar window as described above. This window will not only allow the user to search for ROM addresses that match the entered string, but also for instructions in the trace data window.

Change Processor Speed <Alt–F3>

The command 'Speed' opens a pop-up window in the upper left hand corner of the screen that enables you to change the MARC4's system clock frequency. The default value is 1 MHz which corresponds to an instruction cycle time of 2 µs.

To enable emulation at frequencies greater than 2 MHz the SLEEP instruction in your ROM code will be replaced with an NOP instruction, so the processor will stay active all the time. Furthermore, every occuring I/O activity will be processed with half of the selected speed. This explains the difference between the total elapsed and the active time in the execution time window.

Set Clock Delay, if V_{DD} < 1.6 V <Alt–F4>

To emulate a low voltage MARC4, it may be necessary to have a delay between the external clock which feeds both the ROM-less version of the MARC4 (e3400EVC) and the target MARC4 to compensate the different internal delays of both chips.

Table 2 indicates which delay is the best for your selected supply voltage range.

Table 2. Clock frequency vs. voltage and delay setup for low voltage MARC4 variants

Supply Voltage V_{DD} [V]	Del. Setting <Alt-F4>	Max. Speed <Alt-F3>
1.6 to 2.4	0	800 kHz to 1.3 MHz
≤ 1.6	250 ns	≤ 800 kHz
1.5	250 ns	≤ 800 kHz
1.3	250 ns	≤ 800 kHz
> 1.2	375 ns	≤ 800 kHz
≤ 1.2	375 ns	≤ 500 kHz

Reset all Breakpoints <Alt–F5>

All breakpoints will be set to the 'OFF' state . The message line will display 'All breakpoints cleared'.

Toggle Interrupt Flag <Alt–F6>

This function will toggle the interrupt enable flag status in the CCR of the MARC4 for single step purposes.

Fill a Section of RAM <Alt–F7>

This command opens a window that allows you to fill the RAM from a hexadecimal address to another hexadecimal address with a hexadecimal value. To change single values please use the 'Edit' (**<F9>**) command.

Select Mode for Trace Recording <Alt–F8>

This command opens a pop-up window that allows you to choose between different trace modes.

These are:

- Trace-recording off,
- Record all instruction cycles,
- Record I/O actions only,
- Activity statistics

The selection of one of the last three items makes it possible to display the trace data window after the execution of parts of the program.

Print the Contents of RAM, ROM or Trace Memory <Alt–F9>

If the currently selected 'active' window is the RAM data window, the function key **<Alt-F9>** enables the user to print the whole RAM with its symbolic and hexadecimal addresses together with the corresponding RAM values.

If the active window is the ROM disassembly window, another window will popp up which will then enable the user to enter the address range that will be printed. Those parts of the ROM that contain only SCALL $RESET (instruction 'C1h') will be printed in a compressed manner. Enter 'Y' if the addresses have been entered correctly. Otherwise, press the **<Enter>** key.

If the trace window has been opened and you have recorded the instruction cycles, a window will be displayed where you can enter the first and the last line of the trace window you want to have printed. Enter 'Y' if the line numbers you have entered are correct. The window will be closed and the trace data will be printed. Otherwise, enter 'N' or press **<Enter>** and open the window with **<Alt-F9>** again.

Note: Whenever you want to use this function, be sure that your line printer is switched ON and is 'ON LINE'!

If you have recorded the activity statistics, all the addresses will be printed.

Animation, Continous Single Step <Alt–F10>

On entering this command, the emulation will walk through the program code displayed in the ROM data disassembly instruction by instruction, and after every step the screen will be updated. To stop the animation, you may press any key. The message line will then display the number of instructions that have been executed.

If the trace recording of instruction cycles or activity statistics is turned on, the animation mode will slow down. The recording of I/O activities is inhibited during animation mode.

Leaving the Emulator <Alt–X>

Entering this command will close all windows. In the integrated development system, leaving the emulator will return you to the SDS main menu, while in the command line version the exit command will return you to DOS.

All the entries you have made in the breakpoint, the processor type, the speed, the clock delay and the trace mode window will be saved on the file 'EMU3.CFG' together with the name of the emulated file. The next time the emulator is invoked, this setup will be used as the default configuration.

If a new or changed file is to be emulated, a warning message will be displayed in the message line.

Show Version Number <Shift–F1>

After pressing this function key, the version number and creation date of the emulator software will be displayed in the message line.

The animation mode will also slow down. This may be of use if you are running this software on a fast PC. To run the animation in the 'flickering fast' mode, press **<Shift-F1>** again.

Show Current Emulator Setup <Shift–F3>

This window allows you to have a quick look at the current setup configuration. It displays the name of the file you have loaded into the ROM, the breakpoints you have activated, the record mode, if trace recording has been selected, the chosen processor type, the speed and the clock delay, if it is greater than 0. Press **<Esc>** to close the window.

Select a Target Chip <Shift–F7>

This function has to be used every time, the type of target device on the target application board has been changed or the emulator is starting for the first time. By pressing **<Shift–F7>** a selection window will be displayed in which you have to select the target chip you are working with.

Note: The selected type of target chip enable or disable the displaying of the port window **<Shift–F8>** or the memory window **<Shift–F9>.**

Display Port Window <Shift–F8>

By using this command, a peripheral window will be displayed. This window shows the names of ports and subports (when available), the corresponding port symbols, the contents of the port in hexedecimal manner and the binary display.

Note: This function is only available, if the access to ports and other peripherals is enabled. That means, the selected and used target device has to support this function.

Display Memory Window <Shift–F9>

By using this command, a memory window will be displayed. The meaning of the different columns in this window corresponds to the port window.

Note: This function is only available, if the access to memories over the MARC4 I/O bus is enabled. That means the selected and used target device has to support this function.

7 Target Application Boards

7.1 Introduction

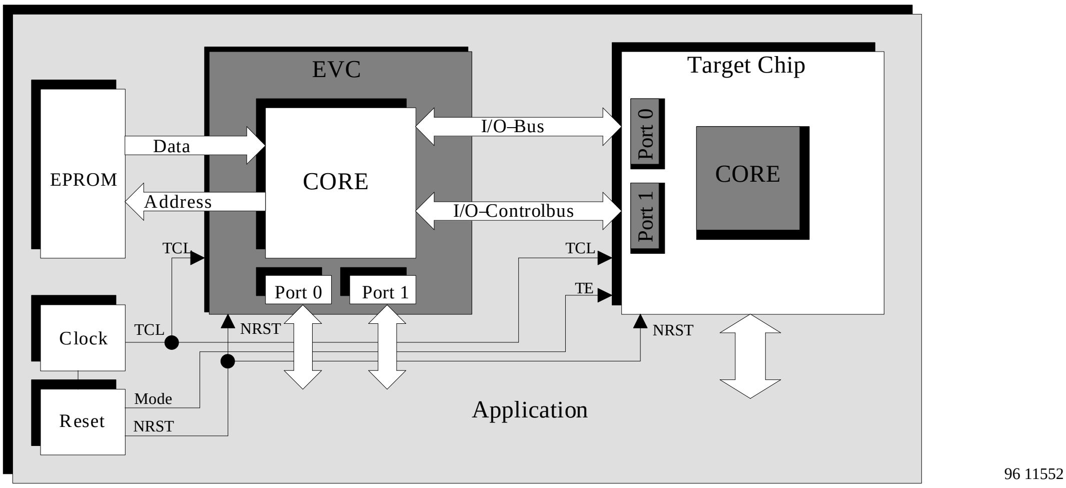


Figure 1. Functional block diagram of TAB505 in stand-alone emulation mode

All MARC4 controllers have a special emulation mode. It is activated by setting the TE pin to logic HIGH level or the TST2 pin to logic LOW level after reset, depending on the used target chip. In this mode, the internal CPU core is inactive and the I/O bus is available via Port 0 and Port 1 to allow the emulator to access the on-chip peripherals. The emulator contains a special emulation CPU (e3400 EVC) with a MARC4 core and additional breakpoint logic which takes over the core function. The basic function of the emulator is to evaluate the customer's program and hardware in real time.

The **T**arget **A**pplication Interface **B**oard (TABxxx) is useful as a ready-to-use interface between the MARC4 emulator (inside the PC), the target device on the TAB and the target application.

The MARC4 development system contains two different types of target application boards. The TAB505 supports two operation modes, the emulation and the stand-alone mode as shown in figure 1. The TAB260 supports the emulation mode only.

7.2 Target Application Board TAB505

Features

In emulation mode:

- Zero-force 64-pin DIL socket for different target MARC4 µC's
- Power supply from the PC (+5 V)
- Adjustable target supply voltage (1.2 to 5 V) with level shift logic
- Standard connectors (DB37, VG96) and shielded emulator interface cable
- Additional LCD interface board for standardized 3:1 or 4:1 multiplex displays

Additional features in stand-alone operation (figure 1):

- Single external supply voltage input (7 to 9 V),
- On-board e3400EVC surrounded by the necessary interface logic,
- 28-pin DIL socket for the customer's EPROM (27C64 - 150 ns),
- Separate RESET button,
- Variable µC operation frequency up to 2.5 MHz,
- Clock delay setting according to the target supply voltage (V_{DD} < 1.6 V).

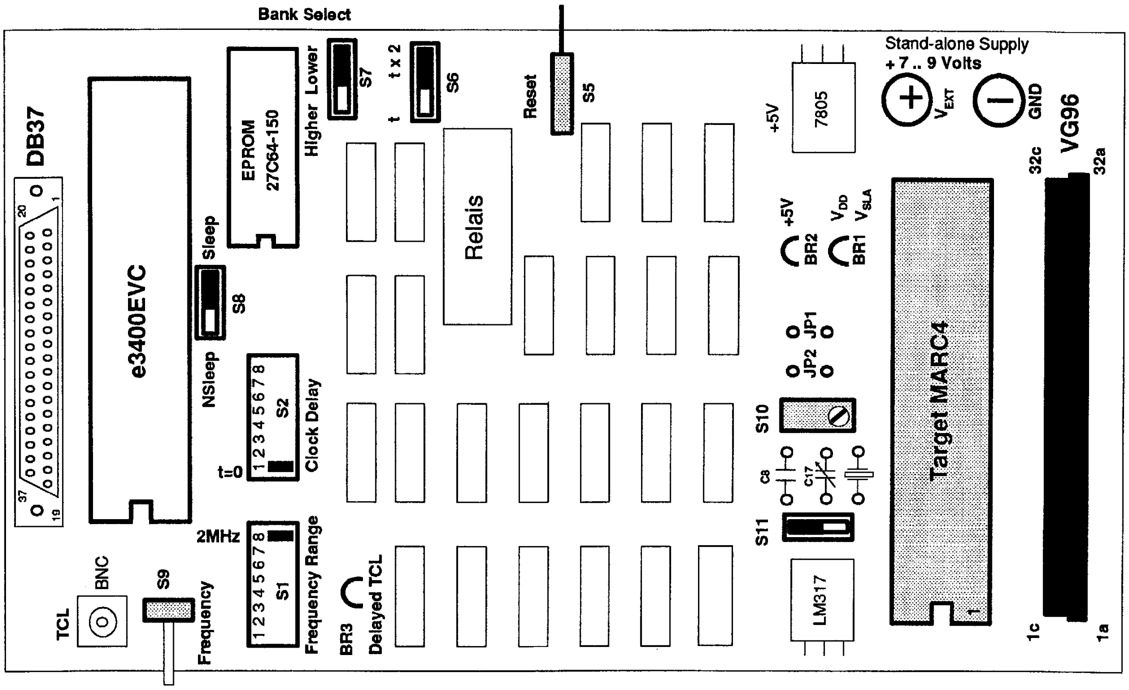


Figure 2. Target application board TAB505 – switch positions

7.2.1 Supply Voltages

The TAB505 has the unique feature of an adjustable target supply voltage by using the trim resistor **S10**. This allows the operation of the target hardware within a wide supply voltage range of 1.2 to 5 Volts. To measure the target supply voltage V_{SLA} = V_{DD}, attach your voltmeter to BR1 and GND, then use a screw driver on **S10** to adjust the voltage output of the LM317 regulator.

The on-board relais is powered by the PC and switches between the external or PC internal supply voltage automatically.

Note: In emulation mode all voltages are derived from the PC's internal +5 V power supply.

In stand-alone operation mode, an external power supply input of 7 to 9 Volts is recommended at $V_{EXT.}$ This input voltage is regulated down to V_{CC} = +5 V.

If the target application operates at +5 V, **JP1** and **JP2** have to be inserted to bypass the voltage drop of the LM317.

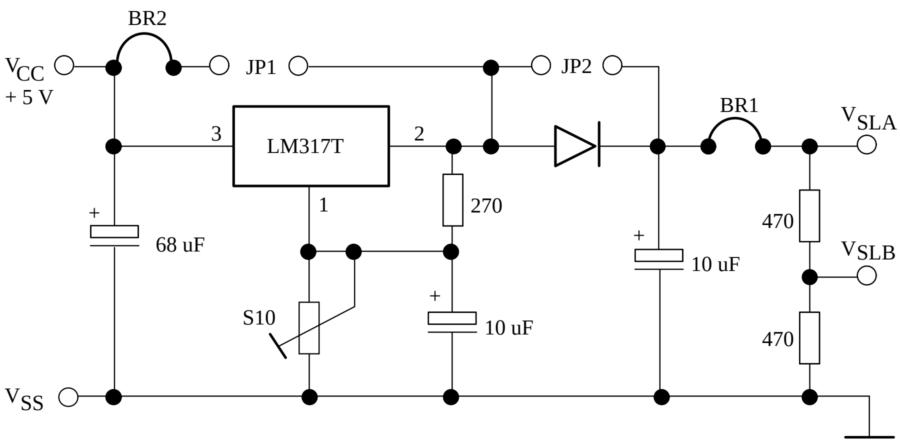


Figure 3. Target supply-voltage generation

7.2.2 Periphery Connector

The VG96 connector is the interface to your application hardware which may be built on a separate board. It is also possible to use a 64-wire ribbon cable as an interface link between the connector and the hardware. The VG96 numbering scheme depends on the used connector. To verify the specified signal assignment use an Ohm-meter between pin 1 of the target MARC4 and the bottom left pin of the VG96 connector which corresponds to VG96-1a in table 1.

Note: By applying the M44C636, M40C092 or M44C510 as target MARC4, it is necessary to use the corresponding target board adapter.

Table 1. Signal assignments on the VG96 connector for M43C505, M45C535, M44C510 and M44C636

Pin	M43C505 / M45C535			M44C510			M44C636		
Nr.	Row a	Row b	Row c	Row a	Row b	Row c	Row a	Row b	Row c
1	BP42	VCC	BP43	BP42	VCC	BP43	BP42	VCC	BP43
2	BP41	——	Trace0_E	BP41	——	Trace0_E	BP41	——	Trace0_E
3	BP40	VSLA	Trace1_E	BP40	VSLA	Trace1_E	BP40	VSLA	Trace1_E
4	S20	——	Trace2_E	BP70	——	Trace2_E	S20	——	Trace2_E
5	S19	VSLB	Trace3_E	BP71	VSLB	Trace3_E	S19	VSLB	Trace3_E
6	S18	——	——	BP72	——	——	S18	——	——
7	S17	BP43	Sleep_E	BP73	BP43	Sleep_E	S17	BP43	Sleep_E
8	S16	IBUS0_S	——	BP61	IBUS0_S	——	S16	IBUS0_S	——
9	S15	IBUS1_S	——	BP60	IBUS1_S	——	S15	IBUS1_S	——
10	S14	IBUS2_S	OD_S	BPB3	IBUS2_S	OD_S	S14	IBUS2_S	OD_S
11	S13	IBUS3_S	NST_E	BPB2	IBUS3_S	NST_E	S13	IBUS3_S	NST_E
12	S12	SL_Dir_S	BP13_S	BPB1	SL_Dir_S	BP13_S	S12	SL_Dir_S	BP13_S
13	S11	NHOLD_S	BP12_S	BPB0	NHOLD_S	BP12_S	S11	NHOLD_S	BP12_S
14	S10	NREAD_S	BP11_S	BPC1 *	NREAD_S	BP11_S	S10	NREAD_S	BP11_S
15	S09	NCYCLE_S	BP10_S	BPC0 *	NCYCLE_S	BP10_S	S09	NCYCLE_S	BP10_S
16	S08	NWRITE_S	BP03_S	——	NWRITE_S	BP03_S	S08	NWRITE_S	BP03_S
17	S07	——	BP02_S	——	——	BP02_S	S07	——	BP02_S
18	S06	SYSCL	BP01_S	BPA0	CLKSL	BP01_S	S06	SYSCL	BP01_S
19	S05	——	BP00_S	BPA1	——	BP00_S	S05	WDEN	BP00_S
20	S04	VSLA	INT7	BPA2	VSLA	——	S04	VSLA	INT7
21	S03	——	VSLA	BPA3	——	VSLA	S03	——	VSLA
22	S02	——	INT2	TIM1	——	——	S02	——	INT2
23	S01	——	IP53	——	——	BP53	S01	——	IP53
24	COM3	——	IP52	——	——	BP52	COM3	——	IP52
25	COM2	——	IP51	——	——	BP51	COM2	——	IP51
26	COM1	——	IP50	——	——	BP50	COM1	——	IP50
27	COM0	TST2_S	——	——	TE_S	——	COM0	TST2_S	——
28	V_{EE1}	TCL_S	——	——	TCL_S	——	V_{EE1}	TCL_S	——
29	C1	——	VINT	——	——	——	C1	——	VINT
30	C2	——	——	——	——	——	C2	——	TIM1
31	V_{EE2}	NRST_S	——	——	NRST_S	NWD_OUT	V_{EE2}	NRST_S	NWD_OUT
32	V_{REG}	GND	GND	——	GND	GND	V_{REG}	GND	GND

* BPC1 and BPC0 are only available if you have a special version of the M44C510 target board adapter. Please contact your TEMIC sales person for more detailed information on this adapter.

7.2.3 Settings

The clock frequency of the internal RC oscillator found on some of the MARC4 cores varies with the operating voltage. The oscillator tracks the supply and temperature to ensure optimum operation of the microcontroller under all conditions. Select the appropriate clock speed either in an emulator pop-up menu (by pressing **<ALT-F3>**) or by setting switch **S1** and **S9** corresponding to the data given in the data sheets, when operated in stand-alone mode.

When using a low voltage MARC4 microcontroller with a target supply voltage of V_{DD} < 1.6 Volts on the target application interface board, the TCL clock supplied to the e3400EVC must be delayed externally.

Table 2 supports provides the possible supply voltages, clock frequencies and clock delay combinations for the M44C636 target µC.

The clock delay is modified either in an emulator pop-up menu (by pressing **<ALT-F4>**) or by setting switch **S2** in the stand-alone operation.

Table 2. Programmable TCL clock delay for M44C636

Target Voltage [V]	Delay Setting	Max. Emulation Speed
V_{DD} [V]	<Alt-F4>	<Alt-F3>
> 1.6	0	800 kHz
≤ 1.6	250 ns	≤ 800 kHz
1.5	250 ns	≤ 800 kHz
1.3	250 ns	≤ 800 kHz
> 1.2	375 ns	≤ 800 kHz
≤ 1.2	375 ns	≤ 500 kHz

To adjust and measure the current TCL clock frequency in stand-alone operation, you may use either the BNC connector or **BR3** (delayed TCL) to attach an oscilloscope or a frequency counter.

The linear DIP switch **S1** gives you a rough frequency select option between maximum values of 20 kHz and 2.5 MHz corresponding to the switch positions 1 and 8.

To calculate the maximum frequency values 'Max_Freq' depending on the position of **S1**, the following formula may be used:

$$\text{Max\_Freq (S1\_pos)} = \frac{2.5\ \text{MHz}}{2^{(8-\text{S1\_pos})}} \text{ with S1\_pos} = 1 \,..\, 8$$

To fine tune the frequency between the calculated maximum values, the trim resistor **S9** should be adjusted.

To get a continous processor clock output, independent from the executed application program, the switch **S8** has to be set to the 'NSleep' position. Then the trim resistor **S9** will allow you to fine tune the µC's clock frequency very easily.

After adjusting the clock frequency the switch **S8** has to be set to the 'Sleep' position to get correct emulation results. In this mode the BNC connector enables you to observe the µC's activity bursts when executing the application program.

The setup of the clock delay in stand-alone operation with V_{DD} < 1.6 V may be done by using table 2 as a guideline for the correct adjustment.

First of all you need a 2-channel oscilloscope which has to be attached to:

- the 'Slave TCL' available on the BNC connector and
- the delayed 'Master TCL' for the e3400EVC available at BR3.

Secondly, the following formulas may help you to adjust the clock delay.

'Delay_Freq' is the frequency that can be measured at the BNC connector with **S1** in position 8.

Then the clock delay is calculated as follows:

$$\text{Delay (S2\_pos, s6} = \text{'t} \times 2\text{')} = \frac{\text{S2\_pos}-1}{\text{Delay\_Freq} * 2}$$

$$\text{Delay (S2\_pos, s6} = \text{'t')} = \frac{\text{S2\_pos}-1}{\text{Delay\_Freq} * 4}$$ with S2_pos = 1 .. 8

Note: The resulting clock skew on the target application interface board does not only depend on the position of the delay switch **S2** but also on the variation of the clock frequency by **S9**. Therefore, it is important to check the delay between the two on-board clocks after major changes in the frequency using **S9**.
To halve the clock delay alter the position of switch **S6** from **'t x 2'** (which should be the default) to **'t'**.

For target supply voltages above 1.6 V, the clock delay switch has to be set to the farthest position on the left (no delay) which is also the default position.

Optional External Trim Capacitors

In some applications where the need for a very accurate time base arises, an additional external trim capacitor C17 could be attached to the OSCIN pin of the MARC4. This capacitance, if used, has to be tied with switch **S11** to V_{SS}.

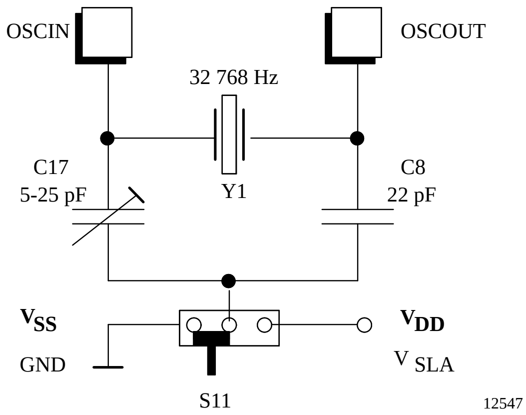


Figure 4. Optional oscillator trimming

7.2.4 Target Board Adapters

M44C510

Trim Resistor R8 – Trigger Level Setup for Internal RESET Pulse

The trimming resistor **R8** (see figure 5) defines the trigger level of the NWD_OUT signal (default $V_{DD}/2$). Figure 5 also shows the trimming resistor **R7** which defines the emulation control signal (TE) trigger level (default 2/3 V_{DD}).

Switch S/E

For stand-alone operation mode, the switch must be in position **S** and for emulation mode in position **E**. Switch **S/E** inverts the emulation control signal (TST2).

Switch TC/CK – Periphery Clock Mode

The positon of the switch **TC/CK** depends on the target µC's SUBCL option. If the periphery clock of the target µC is generated by the SYSCL (SUBCL = SYSCL/64), the switch must be in position **CK**. If the periphery clock is based on the 32-kHz oscillator and SYSCL is stopped in sleep mode the switch has to set in position **TC**. Apart from this, by using switch **TC/CK** it is possible to emulate the mask option SYSCL running or SYSCL stopped during CPU in sleep mode.

Note: All default trimmer values on the target board adapter have been preset to provide optimal adapter performance and should not be changed.

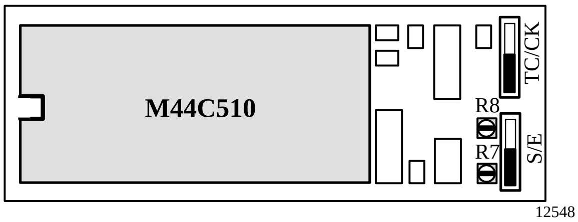


Figure 5. M44C510 target board adapter

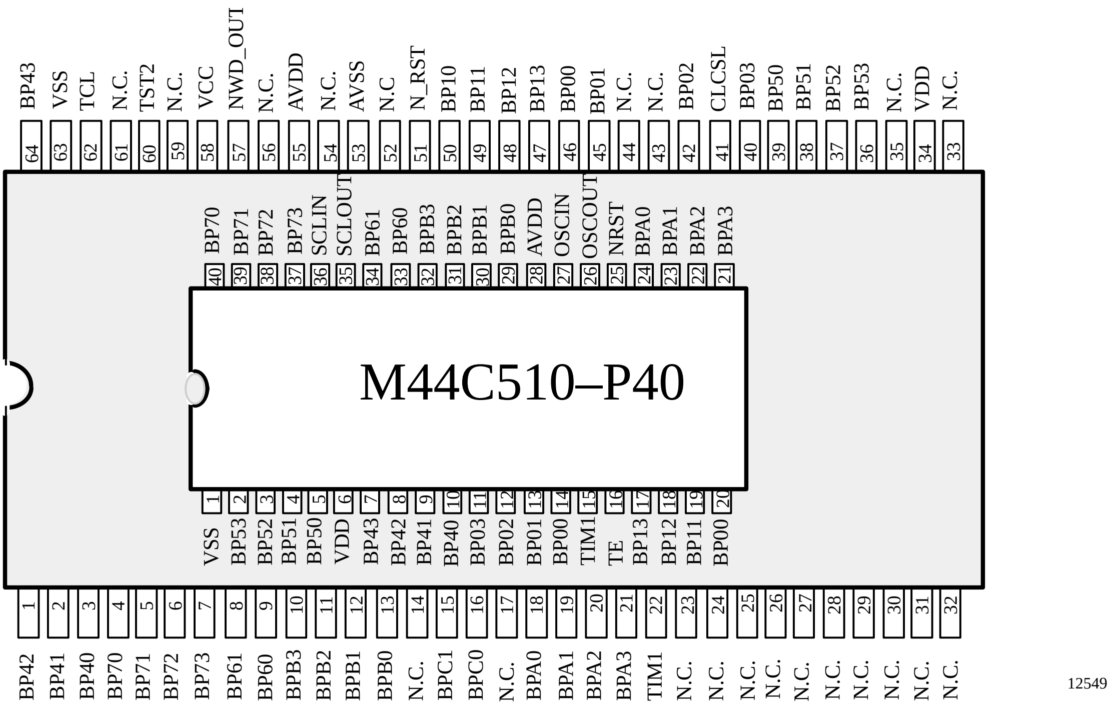


Figure 6. Pin configuration TAB adapter C510 and M44C510

M44C636

32-kHz On-Board Oscillator Q

The watch crystal (Q) is connected to the pins OSCIN and OSCOUT on target board adapter M44C636.

Note: The M44C636 target board adapter converts the pin-out of the M44C636 emulation device into the pin-out of the DIL64 on the TAB505 (see pin configuration in figure 8).

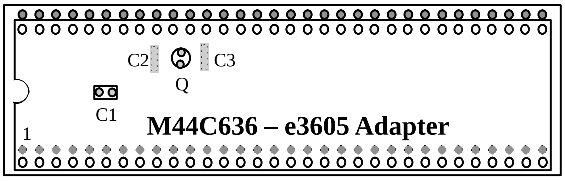


Figure 7. M44C636 target board adapter

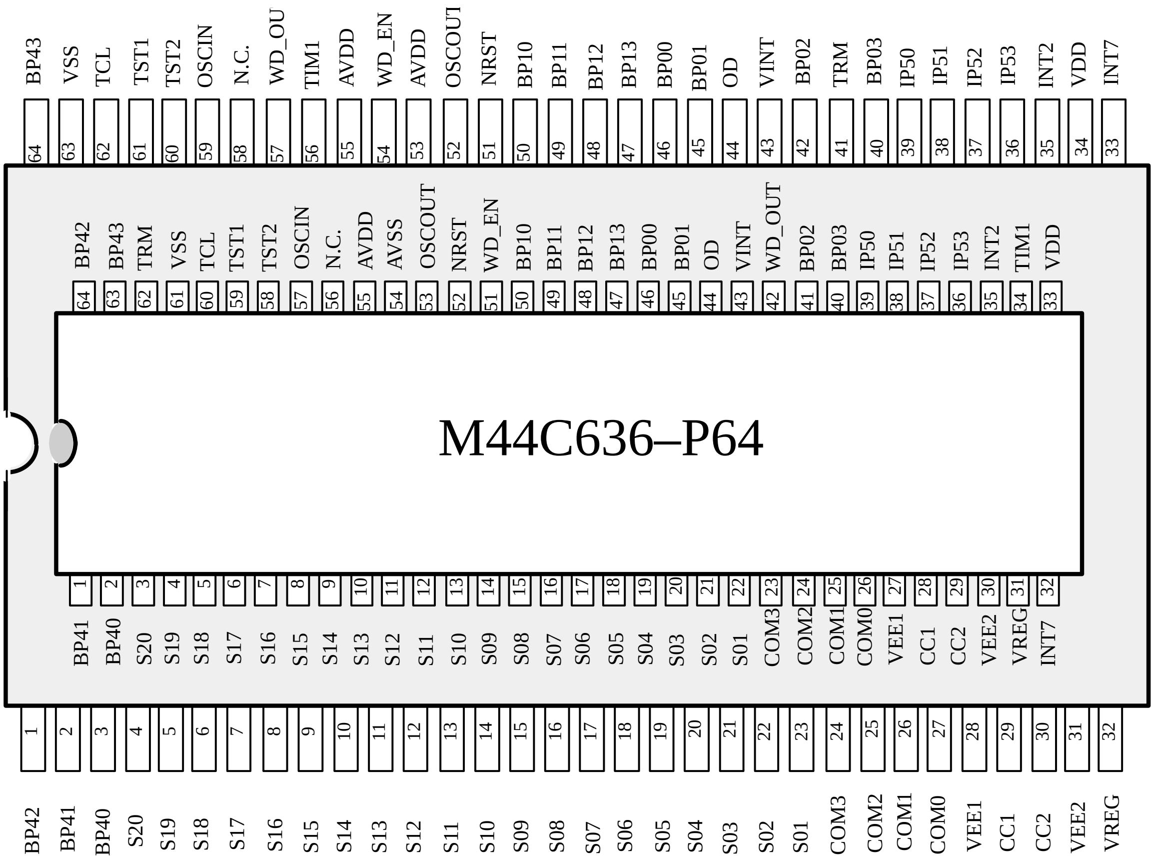
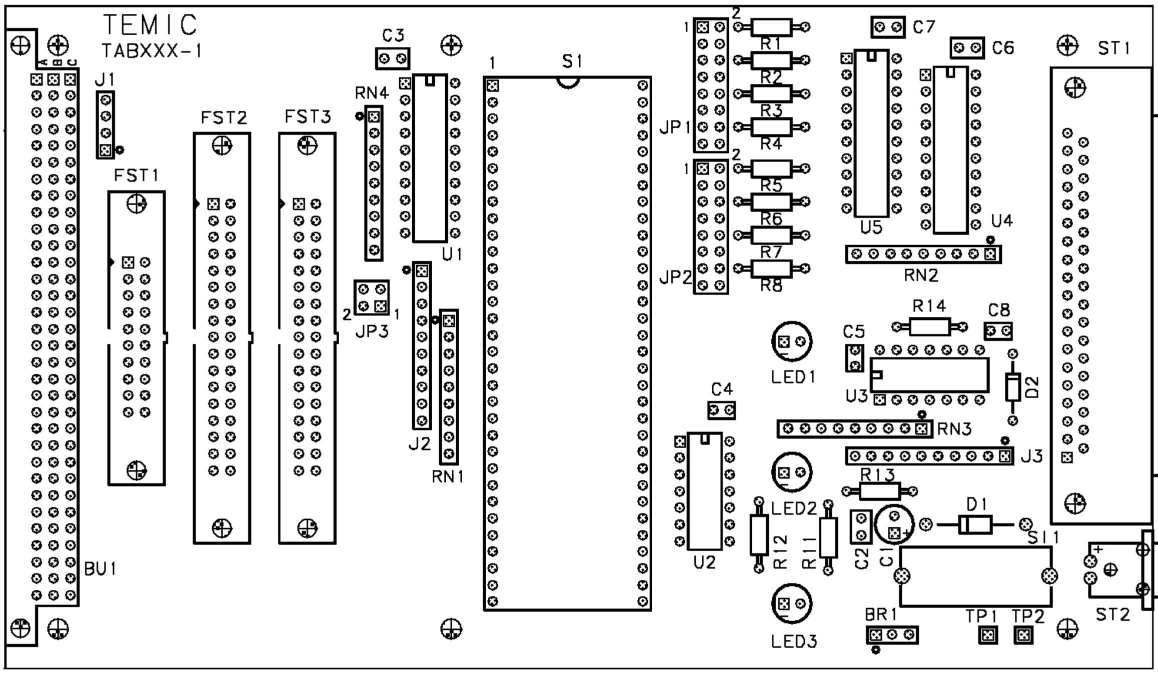


Figure 8. Pin configuration TAB adapter C636 and M44C636

7.3 Target Application Board TAB260

Features

- The printed circuit board is placed into a protective plastic case
- Standard connectors DB37 (ST1) and VG96 (BU1)
- Additional flat cable sockets
- LED's for power supply control (emulator, target board, V_{EXT})
- Also qualified for emulation when M44C510 or M40C092 are used as an emulation µC
- Additional AC/DC adapter for separate power supply (+5 V) available in three different country specific versions:
  – Europe : 230 V~/5.5 V= (50 Hz)
  – U.K. : 230 V~/5.5 V= (50 Hz)
  – U.S.A. : 120 V~/5.5 V= (60 Hz)



12552

Figure 9. Target application board TAB260 – switch positions

7.3.1 Periphery Connectors

Table 3 shows the signal assignment on the VG96 connector when an M44C260 (28 pin), M44C510 or M40C092 are used as an emulation target µC. The VG96 connector is the interface to your application hardware which may be built on a separate board. It is also possible to use a 64 wire ribbon cable as an interface link between the connector and the hardware. The VG96 (BU1) numbering scheme depends on the used connector. To verify the specified signal assignment use an Ohm-meter between pin 1 of the target MARC4 and the bottom left pin of the VG96 (BU1) connector which corresponds to VG96-1a in table 3.

Note: When applying the M44C260, M40C092 or M44C510 as target MARC4, it is necessary to use the corresponding target board adapter.

Table 3. Signal assignments on the VG96 (BU1) connector for M44C260, M40C092 and M44C510

Pin	M44C260			M40C092			M44C510		
Nr.	Row a	Row b	Row c	Row a	Row b	Row c	Row a	Row b	Row c
1	GND	GND	GND	GND	GND	GND	GND	GND	GND
2	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
3	——	——	IP43	——	——	——	——	——	BP43
4	——	——	IP42	——	——	——	BPC1	——	BP42
5	——	——	IP41	BP43	——	BP40	BPC0	——	BP41
6	——	——	IP40	BP42	——	BP53	——	——	BP40
7	——	——	——	BP41	——	——	BPA0	——	——
8	——	——	BP33	BP23	——	BP52	BPA1	——	BP70
9	——	——	BP32	BP22	——	BP51	BPA2	——	BP71
10	——	——	BP31	BP21	——	BP50	BPA3	——	BP72
11	——	——	BP30	BP20	——	——	TIM1	——	BP73
12	——	——	——	BP63	——	——	——	——	BPB0
13	——	——	BP23	BP13	——	——	——	——	BP61
14	——	——	BP22	——	——	BP60	——	——	BP60
15	——	——	BP21	——	——	BP10	——	——	BPB3
16	——	——	BP20	——	——	——	——	——	BPB2
17	——	——	NWP	——	——	——	——	——	BPB1
18	——	——	BP03	——	——	BP03	——	——	BP03
19	——	——	BP02	——	——	BP02	——	——	BP02
20	——	——	BP01	——	——	BP01	BP53	——	BP01
21	——	——	BP00	——	——	BP00	BP52	——	BP00
22	——	——	——	——	——	——	BP51	——	——
23	——	——	BP13	——	——	BP13	BP50	——	BP13
24	——	——	BP12	——	——	BP12	——	——	BP12
25	——	——	BP11	——	——	BP11	——	——	BP11
26	TR0IN	——	BP10	TR0IN	——	BP10	TR0IN	——	BP10
27	TR1IN	——	TCLSLO	TR1IN	——	TCLSL	TR1IN	——	TCLSLO
28	TR2IN	——	NRESO	TR2IN	——	NRESO	TR2IN	——	NRESO
29	TR3IN	——	NRESIN	TR3IN	——	NRESIN	TR3IN	——	NRESIN
30	——	——	CLO	——	——	CLO	——	——	——
31	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
32	GND	GND	GND	GND	GND	GND	GND	GND	GND

Additional Sockets

Additional sockets are provided to interconnect the application hardware to the target board TAB260 or to check the signal assignment.

Note: The target MARC4 is connected to three flat cables sockets (FST1, FST2 and FST3). The FST1 is used for the 20-pin M44C260, the FST2 for the M40C092 and the FST3 for the 28-pin M44C260. The socket pins are organized in such a way when viewing the target board from above they correspond to the pin layout of the used target device (see tables 4, 5, 6 and figures 10, 11, 12).

The listed pin numbers in the following tables are the pin numbers of the flat cable sockets.

Table 4. Signal assignment of flat cable socket FST1

M44C260 (20 Pin)			
Pin		**Pin**	
1	BP02	**11**	——
2	BP01	**12**	——
3	BP03	**13**	——
4	BP00	**14**	IP40
5	——	**15**	BP20
6	BP31	**16**	IP41
7	——	**17**	BP10
8	BP30	**18**	BP13
9	——	**19**	BP11
10	GND	**20**	BP12

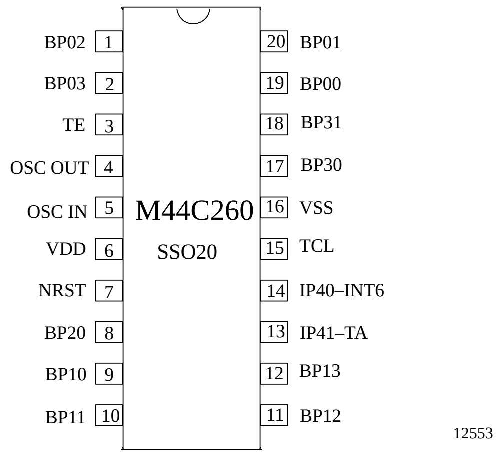


Figure 10. Pin connections of M44C260 (20 pin)

Table 5. Signal assignment of flat cable socket FST2

M44C092 (20 Pin)			
Pin		**Pin**	
1	——	**11**	BP50
2	——	**12**	BP22
3	BP40	**13**	——
4	BP43	**14**	BP21
5	BP53	**15**	——
6	BP42	**16**	BP20
7	BP52	**17**	BP60
8	BP41	**18**	BP63
9	BP51	**19**	BP10
10	BP23	**20**	BP13

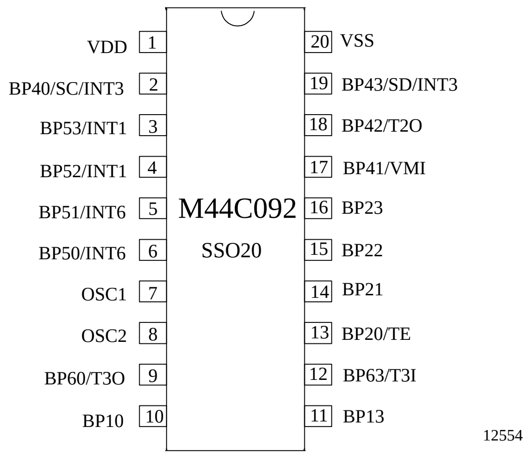


Figure 11. Pin connections M44C092 (20 pin)

Table 6. Signal assignment of flat cable socket FST3

M44C260 (28 Pin)			
Pin		**Pin**	
1	BP02	**15**	——
2	BP01	**16**	——
3	BP03	**17**	BP20
4	BP00	**18**	IP40
5	NWP	**19**	BP21
6	BP33	**20**	IP41
7	——	**21**	BP22
8	BP23	**22**	IP42
9	——	**23**	BP23
10	BP31	**24**	IP43
11	——	**25**	BP10
12	BP30	**26**	BP13
13	——	**27**	BP11
14	GND	**28**	BP12

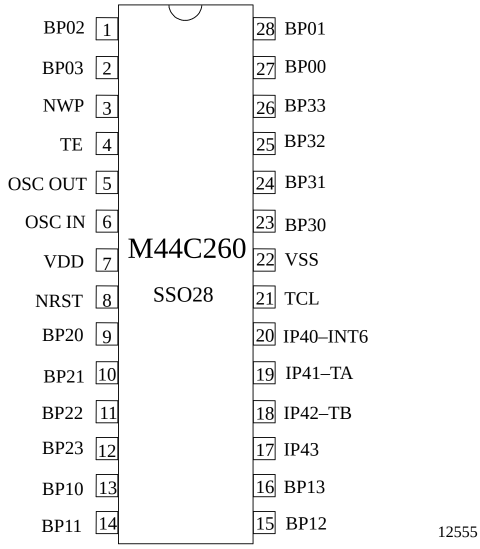


Figure 12. Pin connections for M44C260 (28 pin)

Additional Signals Used in Emulation Mode

Table 7. Signal assignment of additional pin header

Pin Header			
	J1	**J2**	**J3**
PIN	**Signal name**	**Signal name**	**Signal name**
1	TR0IN	IOS0E	TCLSL
2	TR1IN	IOS1E	NRST
3	TR2IN	IOS2E	TST2
4	TR3IN	IOS3E	NRESET
5	Not available	NHOLD	NRES
6	Not available	NWRITE	NRESL
7	Not available	NREAD	——
8	Not available	NCYCLE	P0DIR
9	Not available	CLKSL	P1DIR
10	Not available	SLEEPS	SLDIR

7.3.2 Configuration Setup

Port Configuration

Table 8. Pull-up, pull-down resistor at Port 0

JP1				
Signal name	**Pin**	**ON**	**Pin**	**ON**
BP03	01 - 02	Pull-up	03 - 04	Pull-down
BP02	05 - 06	Pull-up	07 - 08	Pull-down
BP01	09 - 10	Pull-up	11 - 12	Pull-down
BP00	13 - 14	Pull-up	14 - 16	Pull-down

Table 9. Pull-up, pull-down resistor at Port 1

JP2				
Signal name	**Pin**	**ON**	**Pin**	**ON**
BP13	01 - 02	Pull-up	03 - 04	Pull-down
BP12	05 - 06	Pull-up	07 - 08	Pull-down
BP11	09 - 10	Pull-up	11 - 12	Pull-down
BP10	13 - 14	Pull-up	14 - 16	Pull-down

The shaded columns show the production setup of pull-up/pull-down jumpers.

Shifted Signals CLKSL and TCLSL

The jumper **JP3** will supply the shifted signals CLKSL and TCSLS to VG96 (BU1). In the default production setup **JP3** is not inserted.

7.3.3 Supply Voltages

The TAB is powered either at **ST2** by a separated power supply or at **BU1** (VG96) by an external power supply of the customer's application board.

Table 10. Range of power supply

	Symbol	**Min.**	**Typ.**	**Max.**
Supply voltage +5 V	+5 V	4.5 V	5 V	5.5 V
Supply voltage GND	GND		0	

LED1 and **LED2** will check the corresponding power supply. **LED3** indicates the emulator board is switched on.

Power Supply – JP BR1

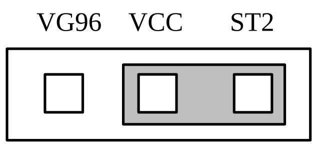


Figure 13. Jumper setting for power supply

The default settings support the external power supply from the AC/DC adapter at **ST2** .

7.3.4 Target Board Adapters

M44C260

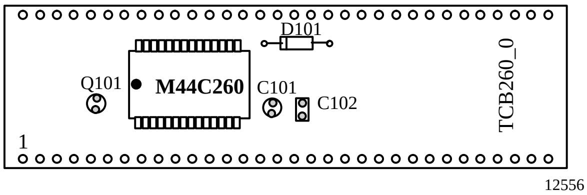


Figure 14. M44C260 target board adapter

32-kHz on-board oscillator (Q)

The watch crystal (Q) is connected to the pins OSCIN and OSCOUT on the target board adapter M44C260.

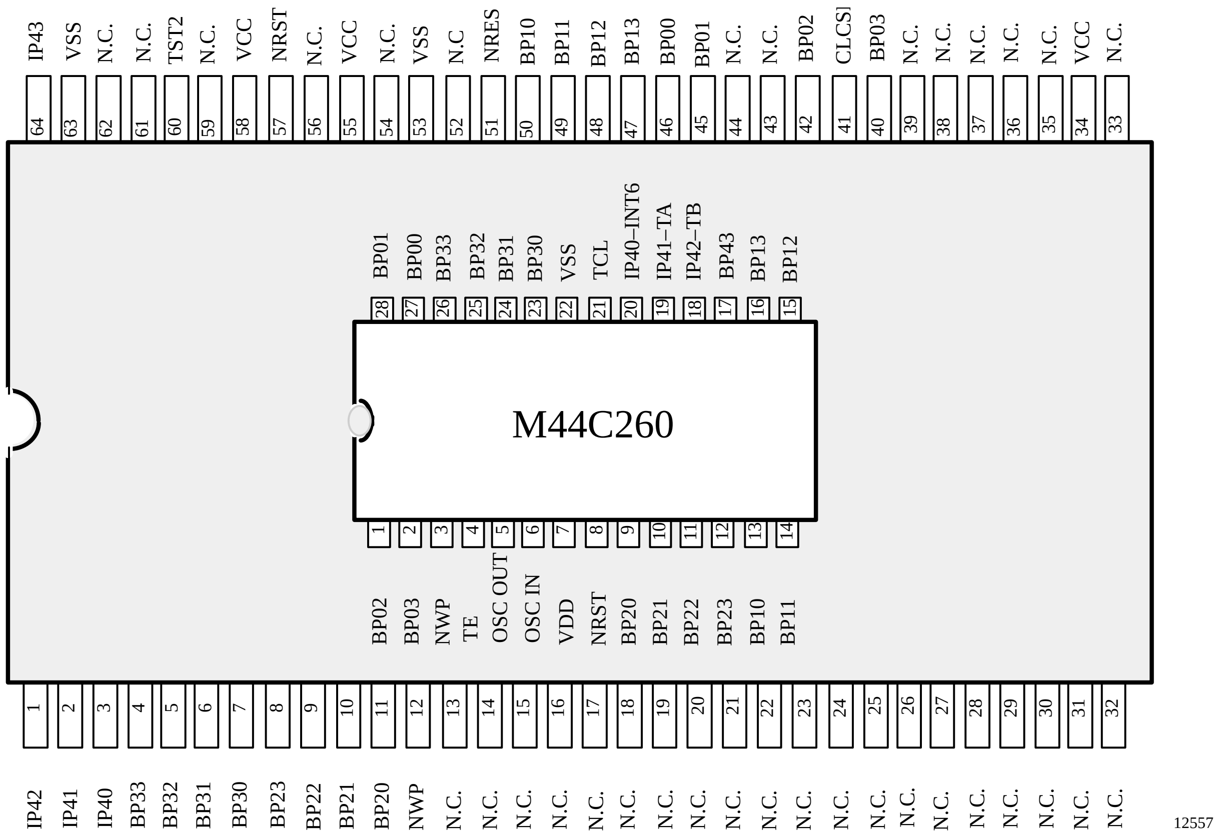


Figure 15. Pin configuration TAB adapter C260 and M44C260

M44C510

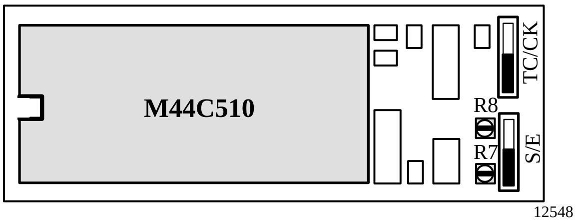


Figure 16. M44C510 target board adapter

Trim resistor R8 – Triger Level Setup for Internal RESET Pulse

The trimming resistor **R8** (see figure 15) defines the trigger level of the NWD_OUT signal (default $V_{DD}/2$). Figure 15 also shows the trimming resistor **R7** which defines the emulation control signal (TE) trigger level (default 2/3 V_{DD}).

Switch S/E

For stand-alone operation mode, the switch must be in position **S** and for emulation operation mode in position **E**. Switch **S/E** inverts the emulation control signal (TST2).

Switch TC/CK – Periphery Clock Mode

The positon of the switch **TC/CK** depends on the target µC's SUBCL option. If the periphery clock of the target µC is generated by the SYSCL (SUBCL = SYSCL/64) the switch must be in position **CK**. If the periphery clock is based on the 32-kHz oscillator and SYSCL is stopped in sleep mode the switch has to be set on position **TC**. By using TC/Ck it is possible to emulate the mask option SYSCL running or SYSCL stopped during CPU is in sleep mode.

Note: All default trimmer values on the target board adapter have been preset to provide optimal adapter performance and should not be changed.

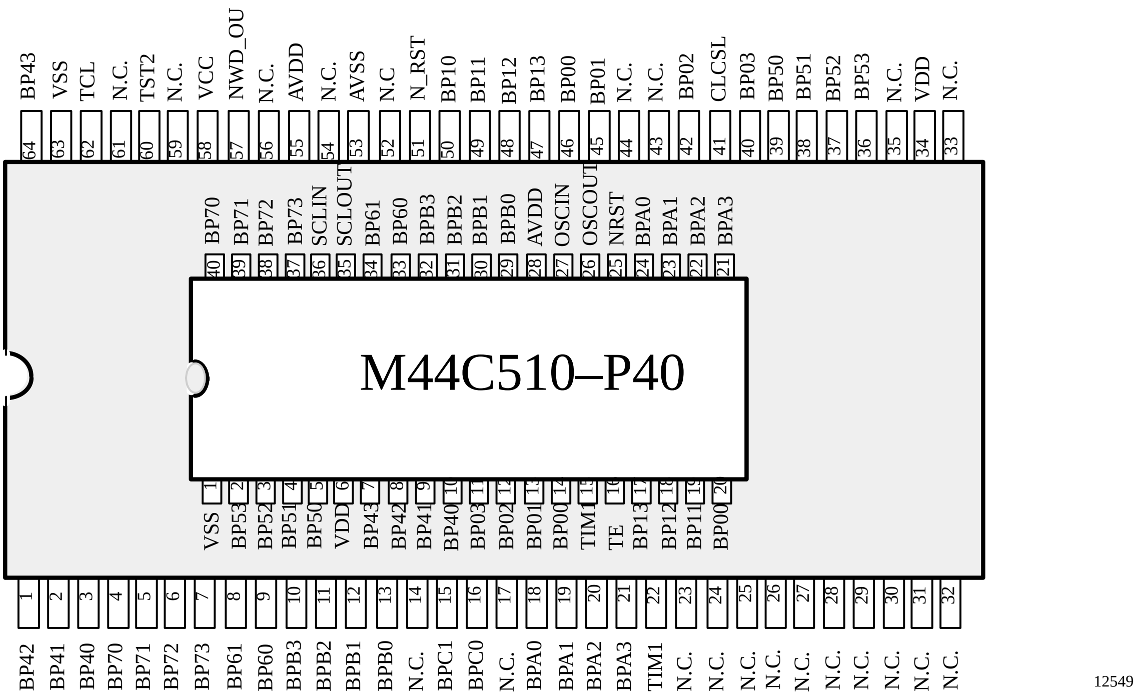


Figure 17. Pin configuration TAB adapter C510 and M44C510

7.4 DB37 Connector and Shielded Emulator Cable

Table 11 specifies the signal assignment of the emulator cable and the DB37 connnector of the both target application boards.

Table 11. MARC4 emulator interface – DB37 signal assignment

Emulator	Dir.	DB37	Target Application Interface
SYSCLK	Out	1	System clock, not stopped in SLEEP (CMOS)
NHOLD	Out	2	To level shifter → BP13 (I/O control)
BP01_E	I/O	3	To level shifter for BP01
BP02_E	I/O	4	To level shifter for BP02
BP03_E	I/O	5	To level shifter for BP03
OD	Out	6	To level shifter for Port 0 read strobe OD2
NST_E	Out	7	Port 0 write strobe NST (CMOS)
BP13_E	I/O	8	To level shifter for BP13
+5V	Out	9	V_{CC} from PC during emulation
BP12_E	I/O	10	To level shifter for BP12
BP10_E	I/O	11	To level shifter for BP10
V_{SS}	Out	12	GND, V_{SS}
NRST_E	Out	14	To level shifter of NRST
BP11_E	I/O	16	To level shifter for BP11
NTST2_E	Out	17	To level shifter for TST2 or TE
SLEEP_E	Out	18	SLEEP signal of EVC (CMOS, active high)
TCL_SL	Out	19	To level shifter for TCL
BP00_E	I/O	20	To level shifter for BP00
Port0_Dir	Out	21	Port 0 direction control for level shift logic
Port1_Dir	Out	22	Port 1 direction contro for level shift logic
SL_Dir_E	Out	25	To level shifter → OD (I/O control)
NWRITE	Out	26	To level shifter → BP12 (I/O control)
NREAD	Out	27	To level shifter → BP11 (I/O control)
NCYCLE	Out	28	To level shifter → BP10 (I/O control)
IOBUS3	I/O	29	To level shifters ↔ BP03 (I/O bus [3])
IOBUS2	I/O	30	To level shifters ↔ BP02 (I/O bus [2])
IOBUS1	I/O	31	To level shifters ↔ BP01 (I/O bus [1])
IOBUS0	I/O	32	To level shifters ↔ BP00 (I/O bus [0])
Trace0_E	In	33	Trace input 0 (CMOS level required)
Trace3_E	In	34	Trace input 3 (CMOS level required)
Trace2_E	In	35	Trace input 2 (CMOS level required)
Trace1_E	In	36	Trace input 1 (CMOS level required)
NWD_OUT	In	37	Watchdog or codet reset input from target (CMOS)

7.5 LCD Interface Board

The LCD interface board is supplied with one of the following standardized LCD modules:

3:1 MUX LCD with up to 8 Digits

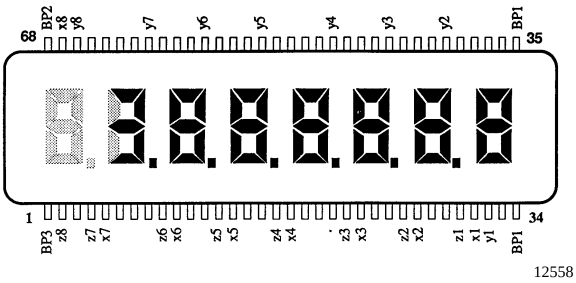


Figure 18. HAMLIN type 4216

4:1 MUX LCD with up to 6 Digits

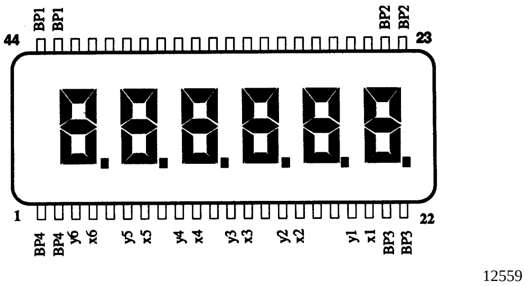


Figure 19. HAMLIN type 4200

Both types of LCD are available with 3 V or 5 V LCD drive level option. The LCD module may be mounted either on the front side or the rear side of the PCB.

Table 12.Ordering information for LCD module

Ordering Information	3 Volt Drive Level	5 Volt Drive Level
HAMLIN 3.1 MUX	4216–313–430	4216–313–420
HAMLIN 4:1 MUX	4200–313–430	4200–313–480

A programming example for a 4:1 multiplex display driving software can be found in your subdirectory "TOOLS" at your MARC4 base directory as source file "HAMLIN.INC". This software module is used in the UNITEST demonstration program too.

For detailed information on the predefined wiring of the segments and backplanes from the MARC4 segment drivers to the LCD, see tables 13 and 14.
In case, you want to use one of these displays in your prototyp application, figures 18 and 19 show the pin-out of the supplied LCD module.

Table 13 shows the LCD segment (frontplane) to the M44C636/M43C505 backplane signal allocation.

Table 13.HAMLIN LCD segment allocation

VG96	3:1 MUX LCD				4:1 MUX LCD		
COM0	**BP1**	–	a	b	**BP1**	a	f
COM1	**BP2**	f	g	c	**BP2**	b	g
COM2	**BP3**	e	d	P	**BP3**	c	e
COM3	——	–	–	–	**BP4**	P	d

Table 14.VG96 to MARC4 segment mapping of HAMLIN LCD on LCD interface board

	3:1 MUX LCD					4:1 MUX LCD					
COM	Digit		0	1	2	Digit		0	1	2	3
S20	**X7**	7	a	g	d						
S19	**Y7**	7	b	c	P						
S18	**Z6**	6	–	f	e						
S17	**X6**	6	a	g	d						
S16	**Y6**	6	b	c	P						
S15	**Z5**	5	–	f	e						
S14	**X5**	5	a	g	d						
S13	**Y5**	5	b	c	P						
S12	**Z4**	4	–	f	e	**Y6**	6	f	g	e	d
S11	**X4**	4	a	g	d	**X6**	6	a	b	c	P
S10	**Y4**	4	b	c	P	**Y5**	5	f	g	e	d
S09	**Z3**	3	–	f	e	**X5**	5	a	b	c	P
S08	**X3**	3	a	g	d	**Y4**	4	f	g	e	d
S07	**Y3**	3	b	c	P	**X4**	4	a	b	c	P
S06	**Z2**	2	–	f	e	**Y3**	3	f	g	e	d
S05	**X2**	2	a	g	d	**X3**	3	a	b	c	P
S04	**Y2**	2	b	c	P	**Y2**	2	f	g	e	d
S03	**Z1**	1	–	f	e	**X2**	2	a	b	c	P
S02	**X1**	1	a	g	d	**Y1**	1	f	g	e	d
S01	**Y1**	1	b	c	–	**X1**	1	a	b	c	P

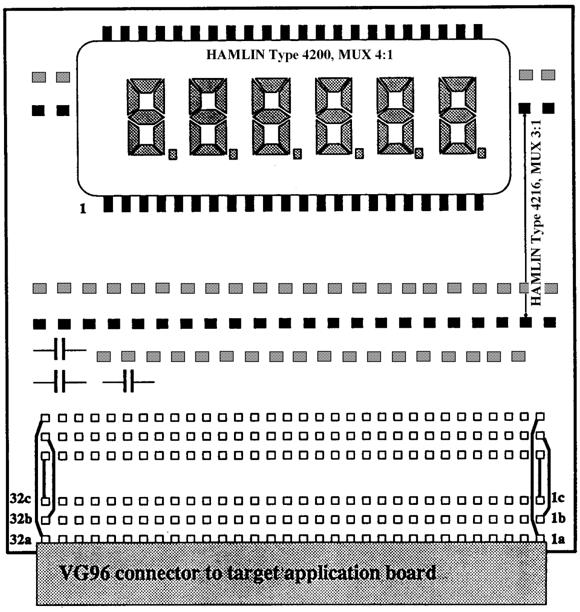


Figure 20. LCD interface board – top view

7.6 Important Hints

To avoid possible damages:

- One should handle the target application interface boards as all other PC plug cards. It is necessary to connect the target application interface to the emulator board before starting the PC.
- The e3400EVC on the target application board is used in stand-alone operation mode only, and should therefore be removed in emulation mode to avoid unnecessary stressing.

Schematic Diagrams

For detailed information on the TAB505 and TAB260 as well as their adapters, the corresponding schematic diagrams are attached to the hardware.

Listing of the available schematic diagrams:

Target application board TAB505
Target board adapter M44C510
Target board adapter M44C636

Target application board TAB260
Target board adapter M44C260
Target board adapter M44C092

8 Piggybacks

8.1 Introduction

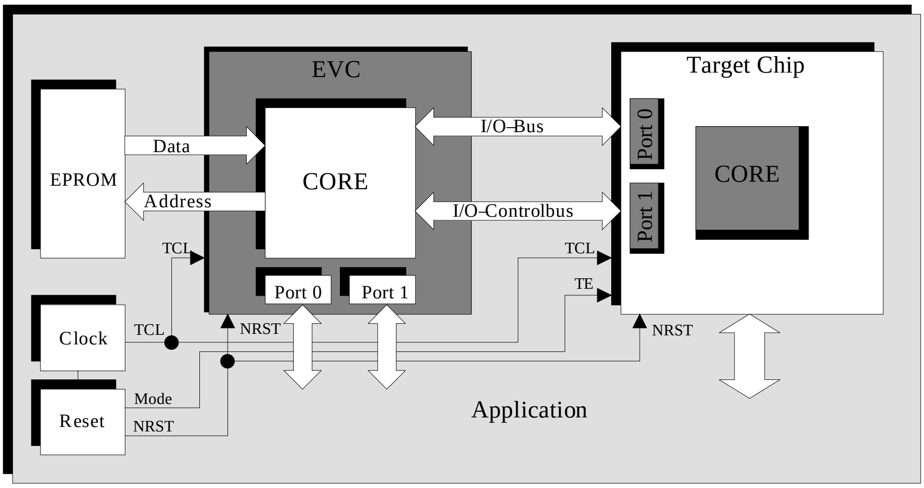


Figure 1. Functional block diagram

A piggyback is an ideal tool for real-time program evaluation in the target environment especially for prototype demonstrators. They are fully pin compatible to the corresponding prototype/emulation package of the masked version (see figures 2 and 4). Therefore, the mask version will be a pin-to-pin replacement of the piggyback in the prototype application board.

Although the user may think of the piggyback hybrid as one microcontroller, two are actually contained on a single PCB. It contains the ROM-less bond-out chip (e34000EVC) and the application target device (e.g., M44C636, M44C510) operated in emulation mode (see chapter 7 target application board in stand-alone operation)

8.2 M40C510 – PGY

Features

- Supply voltage range from 3 to 5.5 V
- Standard 27C256/27C512 type EPROM/OTPROM in 32-pin PLCC is attached externally
- Up to eight different programs can be stored and selected
- Adjustable – externally supplied – processor clock from 0.2 MHz up to 3 MHz
- 40 pin DIL package
- Size: Length 60 mm, width 32 mm, height 18 mm

8.2.1 General

The top view of the piggyback hybrid is shown in figure 2. The pin-out of this device is identical to the M44C510-P40. Please note that BPC0 and BPC1 are available in the special M40C510C-001 configuration named M40C510-C0C1 only (see table 1). Any standard 27C256/27C512 type EPROM can be placed into the 32 pin PLCC socket mounted on top of the package. Due to the bank select facility of 8 x 4 KBytes, this EPROM can be programmed with eight different program variants.

8.2.2 Available Configurations

Table 1. I/O configurations of available M40C510 piggyback versions

I/O Options	M40C510C-001 M40C510-009	M40C510C-C0C1 *	M40C510C-002 M40C510-912	M40C510–914	M40C510–916
Color Code	**Red**	**White**	**Brown**	**Yellow**	**Copper**
BP00	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP01	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP02	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP03	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP10	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP11	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP12	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP13	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU	CMOS_PU
BP40	Open drain_PU	Open drain_PU	CMOS_PU	CMOS_PD	CMOS
BP41	Open drain_PU	Open drain_PU	CMOS_PU	CMOS_PD	CMOS
BP42	Open drain_PU	Open drain_PU	CMOS_PU	CMOS_PD	CMOS
BP43	Open drain_PU	Open drain_PU	CMOS_PD	Open drain	CMOS
BP50	Open drain_PU	Open drain_PU	CMOS_PD	CMOS_PD	CMOS_PD
BP51	Open drain_PU	Open drain_PU	CMOS_PD	CMOS_PD	CMOS_PD
BP52	Open drain_PU	Open drain_PU	CMOS_PD	CMOS_PD	CMOS_PD
BP53	Open drain_PU	Open drain_PU	CMOS_PD	CMOS_PD	CMOS_PD
BP60	Open drain_2k PU	Open drain_2k PU	CMOS	CMOS_2k PU	CMOS
BP61	Open drain_2k PU	Open drain_2k PU	CMOS	CMOS_2k PU	CMOS
BP70	Open drain_PU	Open drain_PU	CMOS	Open drain	Open drain_PU
BP71	Open drain_PU	Open drain_PU	CMOS	CMOS	Open drain_PU
BP72	Open drain_PU	Open drain_PU	CMOS	CMOS	Open drain_PU
BP73	Open drain_PU	Open drain_PU	CMOS	CMOS	Open drain_PU
BPA0	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	CMOS_PU
BPA1	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	CMOS_PU
BPA2	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	CMOS_PU
BPA3	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	CMOS_PU
BPB0	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	Open drain
BPB1	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	Open drain
BPB2	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	Open drain_PU
BPB3	CMOS_30k PU	CMOS_30k PU	CMOS_PD	CMOS_30k PU	Open drain_PU
BPC0	———	Open drain_PU	———	———	———
BPC1	———	Open drain_PU	———	———	———
TIM1	Open drain_PU	Open drain_PU	CMOS	CMOS_PD	CMOS
BPA_Reset	NO_RST	NO_RST	NO_RST	NO_RST	NO_RST
Watchdog	2 sec	2 sec	2 sec	1 sec	Disabled
SUBCL_SRC	SYSCL/64	SYSCL/64	SYSCL/64	SYSCL/64	32 kHz oscillator

Note: Other I/O configurations may be made available on request within four weeks of order

* The M40C510C-C0C1 and M40C510C-C1T1 are special variants of the piggyback M40C510C-001. On the configuration of M40C510C-C0C1 the BPC0 and BPC1 are bonded-out at OSCOUT and OSCIN. On the M40C510C-C1T1 BPC1 is connected internaly to the pin TIM1.

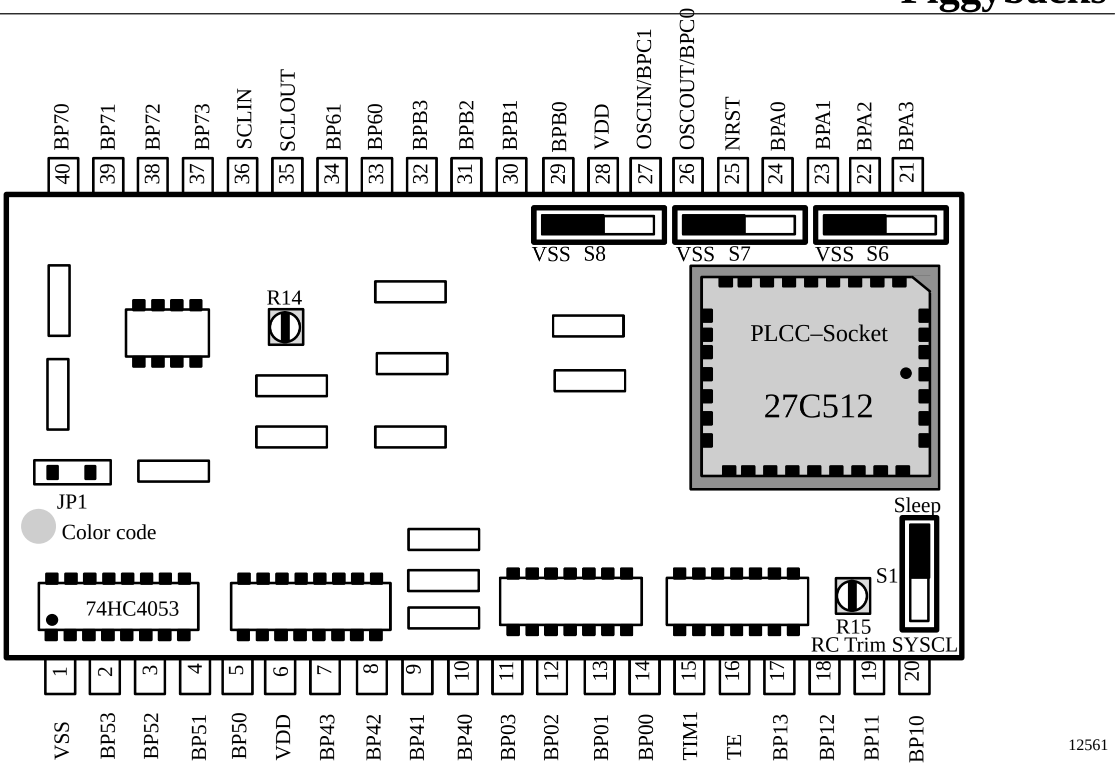


Figure 2. Top view of the M40C510 piggyback

8.2.3 Piggyback Setup

Figure 2 shows the pin-out of the piggyback as well as the placement of the different adjustable components on top of the hybrid. For more detailed information on the piggyback M40C510, the corresponding schematic diagram is attached with the hardware.

Supply Voltage (V_{DD})

The supply voltage range is specified from 3 to 5.5 Volts.

Note: The maximum clock frequency of the piggyback is a function of the supply voltage and the min. access time (e.g., 90 ns) of the inserted EPROM.

Switch S1 – Periphery Clock Mode

The position of the switch **S1** depends on the target µC’s SUBCL option. If the periphery clock of the target µC is generated by the SYSCL (SUBCL = SYSCL/64), the switch must be set in position SYSCL. If the periphery clock is based on the 32-kHz oscillator and SYSCL is stopped in sleep mode, the switch **S1** has to be set in position SLEEP. By using switch **S1** it is possible to emulate the mask option SYSCL running or SYSCL stopped during CPU in sleep mode.

Program Memory Bank Switches S8, S7, S6

For program storage it is recommended to use a 27C256 or 27C512 CMOS (E)PROM in a 32 pin PLCC package. The access time for 5 V CMOS EPROMs should be 90 ns or below. The M44C510 microcontroller can address up to 4096 bytes of program memory. With the memory bank switches **S8 ... S6** it is possible to keep a maximum of eight program versions in one EPROM. Table 2 shows the different switch settings with the corresponding program starting address for your EPROM programmer.

Table 2. Program memory bank switch setting

EPROM Type			27C256	27C512
Switches			ROM Start Address	ROM Start Address
S8	**S7**	**S6**		
VSS	VSS	VSS	0000h	8000h
VDD	VSS	VSS	1000h	9000h
VSS	VDD	VSS	2000h	A000h
VDD	VDD	VSS	3000h	B000h
VSS	VSS	VDD	4000h	C000h
VDD	VSS	VCC	5000h	D000h
VSS	VDD	VDD	6000h	E000h
VDD	VDD	VDD	7000h	F000h

Jumper JP1 – Disable of Internal RESET Events

A reset signal, which forces the µC in a well defined condition, can be triggered by either initial supply power-up, a watchdog time-out, activation of the NRST input or occurence of a coded reset on Port A (see figure 3).

If a watchdog reset or a coded reset on Port A is available via the corresponding mask option (see table 1) and this function should be suppressed in the application, **JP1** has to be removed.

Trim Resistor R14 – Trigger Level Setup for Internal RESET Pulse

The trimming resistor **R14** defines the trigger level of the NRST signal output. Please do not try to re-adjust the production setup.

Trim Resistor R15 – System Clock Setup

The system clock frequency used to drive the processor clock (i.e., SYSCL) may be varied in the range from 0.2 MHz up to 3 MHz (at 5 Volts). The system frequency is increased or decreased by adjusting the trim resistor **R15**. To adjust the system clock frequency the switch **S1** must be set in position SYSCL. The production setup is 1 MHz.

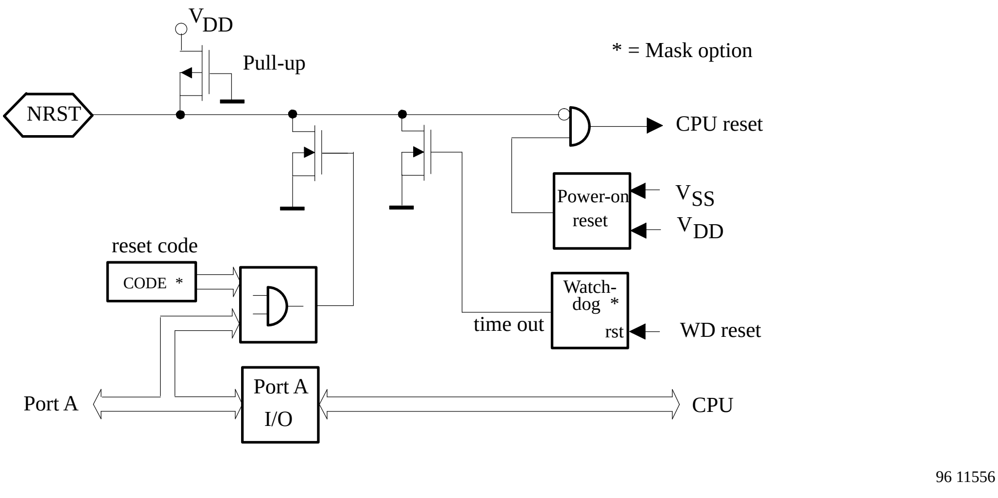


Figure 3. Reset configuration of M4xC510

8.3 M40C636-PGY, M40C505-PGY

Features

- Supply voltage range from 3 to 5.5 V
- Standard 27C64 type EPROM in 28-pin DIL is attached externally, two different program versions can be stored and selected
- Adjustable – externally supplied – processor speed from 0.2 MHz up to 2 MHz
- 64 pin DIL package compatible
- Size: Length 90 mm, width 28 mm, height 20 mm

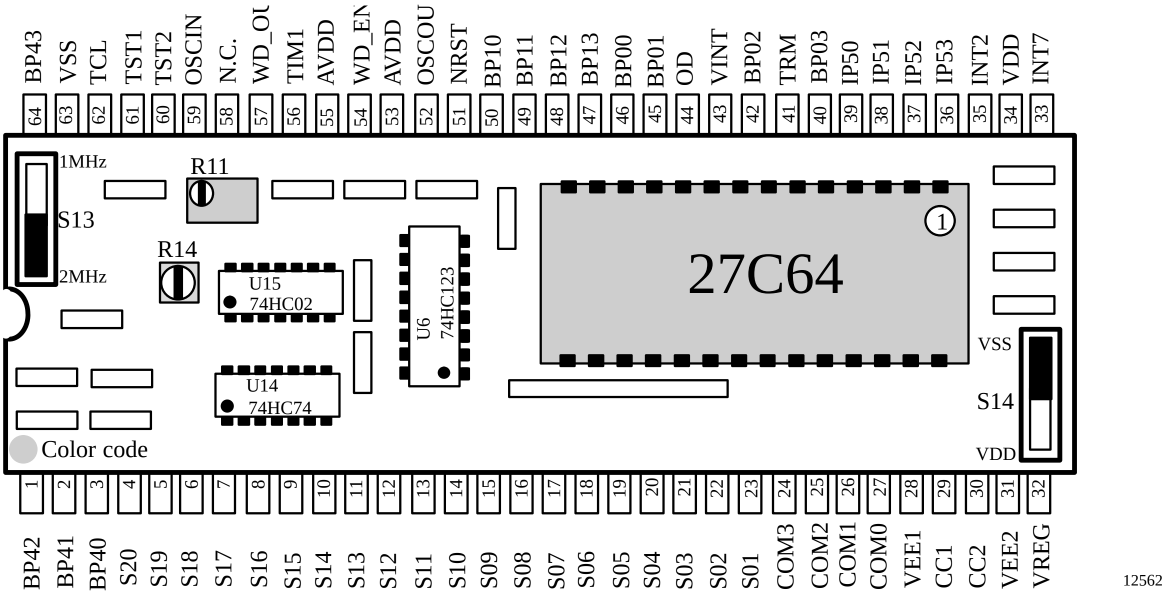


Figure 4. Top view of the M40C636 piggyback

8.3.1 General

The top view of the piggyback hybrid is shown in figure 4. The pin-out of this device is identical to the M43C505-P64 (see figure 6). Please note that the pin connection of the piggyback M40C636 is not identical to the pin-out of DIL64 evaluation samples.

If you build a M44C636 prototype application board based on the M40C636-PGY pin-out (see figure 4), you will need the TAB636 adapter to convert the M44C636-P64 (see figure 6, right) into the pin-out shown in figure 4. The TAB636 adapter contains the 32-kHz crystal on board as it is available on the M40C636-PGY.

The pin-out of the M43C505-P64 (see figure 6, left) and the M40C505-PGY is fully compatible. If you replace the piggyback with the prototype device (in DIL64), please take care that the 32-kHz crystal is attached between OSCIN and OSCOUT.

Any standard 27C64 type EPROM can be placed into the 28 pin DIL socket mounted on top of the package. Due to the bank select facility, the EPROM can contain two different program variants.

8.3.2 Available Configurations

Currently it is possible to choose between one of the following I/O configurations, whereby Port 0 and Port 1are fixed to CMOS_PU.

Table 3. I/O configurations of available piggyback versions

I/O Options	M40C636	M40C505–001
Color Code	**Blue**	**Red**
BP40	CMOS_PU	CMOS_PU
BP41	CMOS_PU	Open drain
BP42	CMOS & 32 kHz	Open drain
BP43	CMOS_PD	Open drain
Port 5	Pull-down	Pull-up
Port 5 INT	Pos. edge INT1	Neg. edge INT4
Coded RST	RST4	Not available
INT2	CMOS_PD	CMOS
INT7	Pull-down	Pull-up
TIM1	CMOS_PU	Not available
Buzzer	2 kHz *	2 kHz
LCD	3 V LCD	5 V LCD

* programmable

Note: Other I/O configurations may be made available within four weeks of placing your request

8.3.3 Piggyback Setup

Figure 4 shows the pin-out of the piggyback as well as the placement of the different adjustable components on top of the hybrid. For more detailed information on the piggyback the corresponding schematic diagram is attached to the hardware.

Supply Voltage (V_{DD})

The supply voltage range is specified from 3 to 5.5 V.

Note: The maximum clock frequency of the piggyback is a function of the supply voltage range and the minimum access time (e.g., 150 ns) of the inserted EPROM.

Trim Resistor R11 and Switch S13 – Externally RC Oscillator Setup

The externally supplied RC oscillator frequency used to drive the processor clock (i.e. TCL) may be varied in the range from 0.2 MHz up to 2 MHz. The oscillator frequency is increased or decreased by adjusting the trim resistor **R11**. The switch **S13** enables the raw selection (divide by 2) of the trimmable frequency range.

The corresponding system clock (SYSCL) is measurable at pin 4 of U15 (74HC02) when the MARC4 microcontroller is not in SLEEP mode. Otherwise, you may use pin 10 of U15 for a continuous frequency output. Pin 4 of U15 also allows the observation of the program activity (duty cycle) too.

Program Memory Bank Switch (S14)

For program storage, a standard 27C64 CMOS EPROM in a 28 pin DIL package is used. The access time for 5 V CMOS EPROMS should be 150 ns or below.

Table 4. Memory bank switch setting

Address range	Switch S14
0000h to 0FFFh	V_{SS}
1000h to 1FFFh	V_{DD}

Power-on Reset

A reset signal, which forces the µC in a well defined start-up condition, can be triggered by different modes (see figure 5). At the microcontroller M44C636, a start-up condition can be forced by an external reset pin (NRST), a coded reset at Port 5, a watchdog time-out and a power-on reset function. A coded reset and a watchdog time-out function is not available on the µC M43C505. On the piggyback hybrid, the power-on reset consists of a RC network with a time constant of some milliseconds.

To implement an additional external RESET input for your application, the signal on pin 51 of the piggyback should be used to attach an external switch connected to V_{SS}.

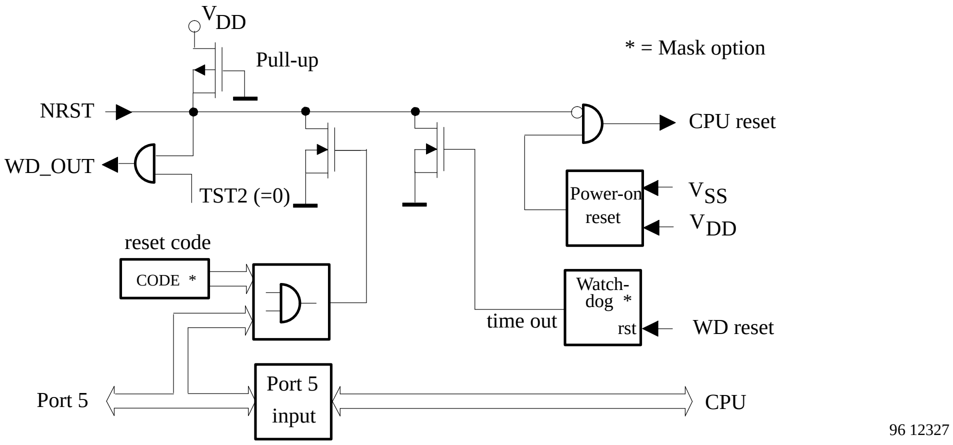


Figure 5. Reset configuration of M44C636

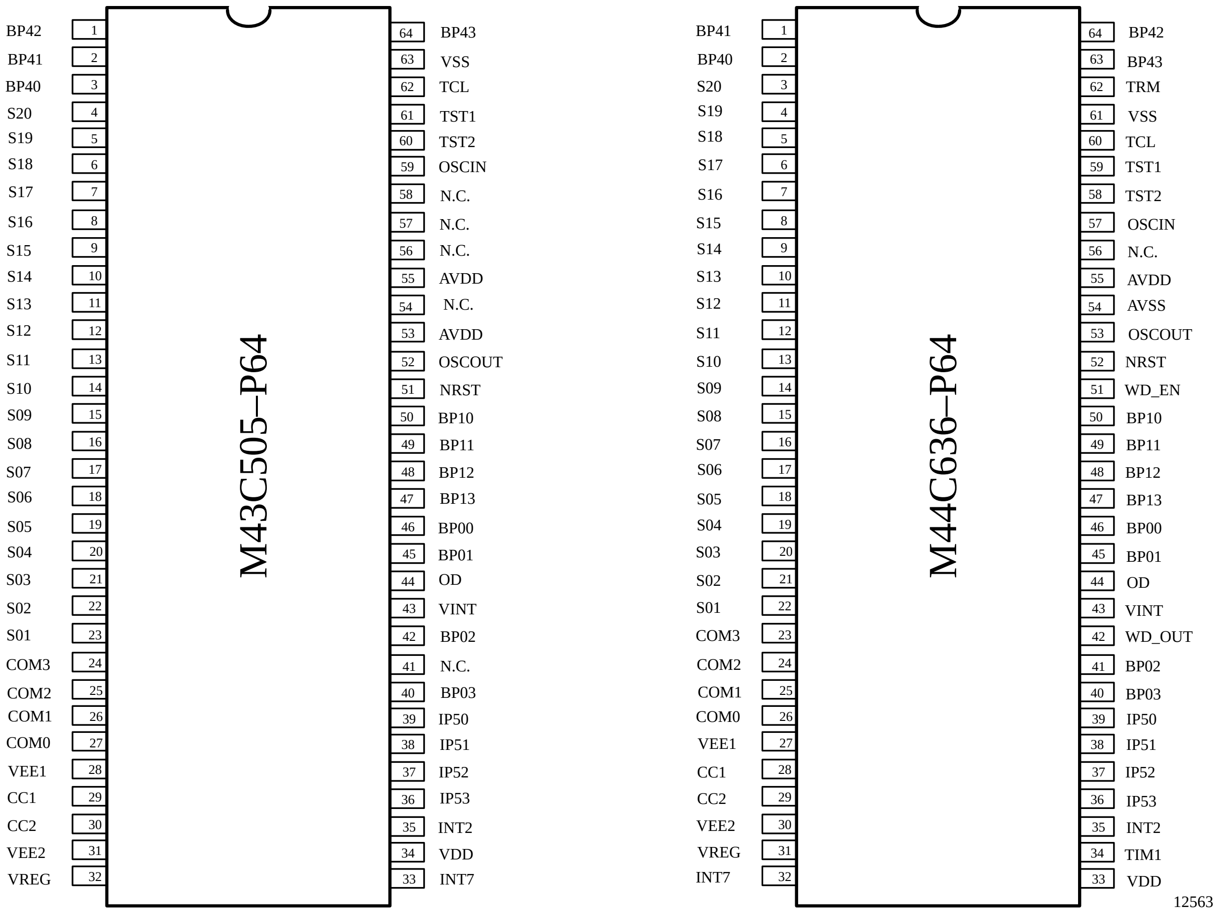


Figure 6. Pin connections – 64 pin ceramic DIL of M43C505 and M44C636

Schematic Diagrams

For detailed information on the M40C510-PGY and the M40C636-PGY the corresponding schematic diagrams are attached with the hardware.

9 OTP Programmer

9.1 Introduction

This programmer’s device can be used for writing or reading out the internal EEPROM memory of the M48C260 and M48C092 OTP microcontrollers. It offers all possibilities for memory manipulation which are available in a conventional EPROM programmer.

Features

- Easy adaption onto the PC
- Comfortable and user-friendly programmer’s shell with mouse support
- Protecting plastic case with zero-force 28-pin DIL socket with additional adapter for SSO28-0.8 package versions
- Short programming time
- Additional AC/DC adapter for separate power supply (+5 V) available in three different country-specific versions:
  – Europe : 230 V~/5.5 V= (50 Hz)
  – U.K. : 230 V~/5.5 V= (50 Hz)
  – U.S.A. : 120 V~/5.5 V= (60 Hz)
  This adapter can also be used for the target application board TAB260.
- Cable to connect the OTP programmer to a parallel port of your PC

9.2 Getting Started

First of all, connect the programmer’s device with the power supply (socket ST2) and with the PC’s parallel port.

If you want to start the OTP programmer, verify that the following files are available in the MARC4 base directory:

MARC4OTP.EXE
MARC4OTP.TVR
MARC4OTO.DEV
MARC4OTP.INI

Check that the correct path for the OTP programmer has been entered in the setting window ”Directories” (see ’Installation Guide’) of the SDS2-IDE.

Starting the OTP programmer with the SDS2 enviroment

To start the OTP programmer within the SDS2 enviroment, select the command ”OTP-Prog.” at the menu line.

Starting the OTP programmer as an independent program

If the MARC4 directory has not been included in the AUTOEXEC.BAT search path, move into the MARC4 system directory (e.g.,C:\MARC4) and enter the following command:
C:\MARC4>MARC4OTP

Note: Please be aware that some lap-top computers do not support all control signals required to operate the OTP programmer (see section 9.7).

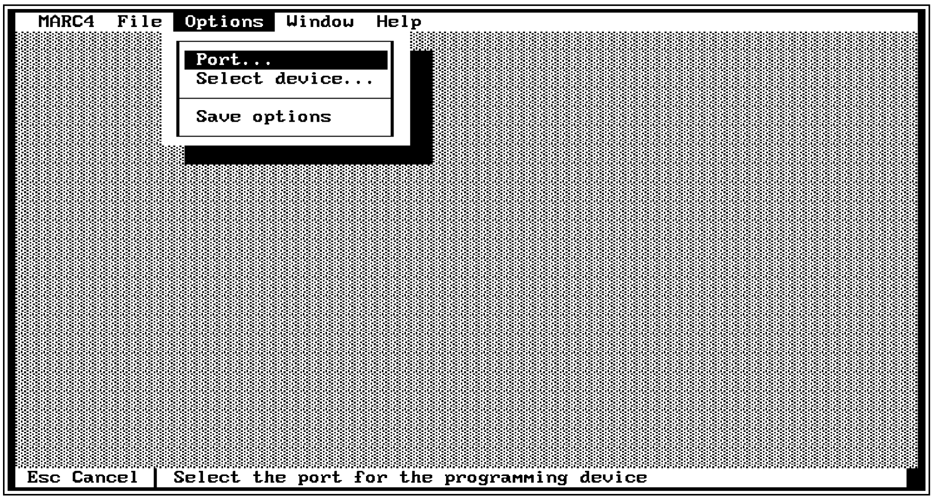
9.3 Set Programmer’s Options

Use the pull-down menu ”Options” and select ”Port” to set the correct PC’s parallel port and select ”Select device” to set the microcontroller which is to be used. Under the submenu command ”Save options”, these settings can be stored in the configuration file.

Note: The device list of M48C260 is now available. Please contact your TEMIC sales person for a M48C092 upgrade.

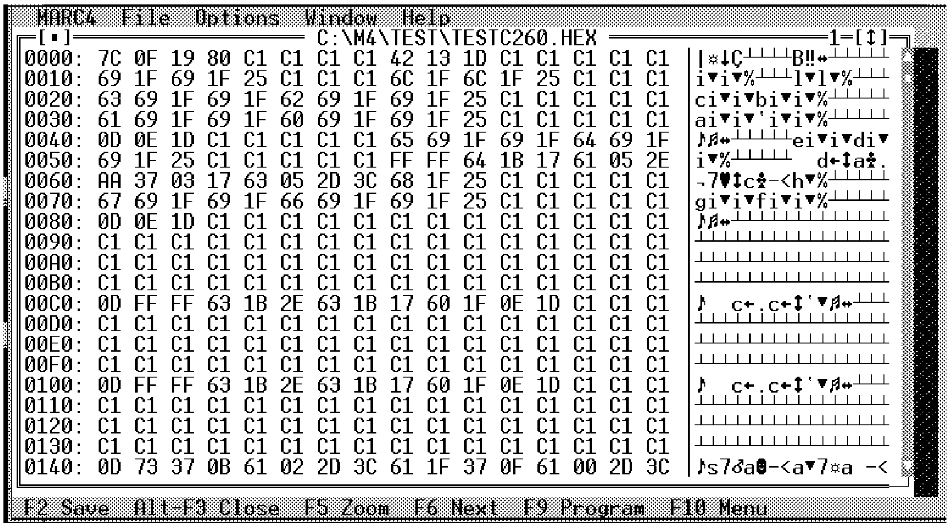
9.4 Using the OTP Programmer

Use the pull-down menu ”File” and select ”Open” to load any MARC4 binary ’HEX’ file.
The loaded file is represented in an edit window in hexadecimal form at the left side and as ASCII dump at the right side of the window. By pressing **<Page-up>**, **<Page-down>** keys or the arrow keys, the active window can be scrolled over the screen. In this way, the binary object file can be edited easily.



12564

Figure 1. OTP programmer’s user display



12565

Figure 2. OTP program screen display with loaded file

Note: Only hexadecimal commands are allowed in the left half of the edit window and ASCII signs in the right half.

Alternating between both fields can be carried out by using **<TAB>** or **<SHIFT–TAB>**.

To start the programming procedure use the pull-down menu ”MARC4” and select ”Program”. A window appears where the address which has just been programmed is shown. At the same time, the LED ”Program” lights up on the programming device. This LED indicates that the programming procedure is in operation.

By pressing the **<ESC>** key the programming procedure can be interrupted at any time.

After the programming procedure has been completed, an automatic verification of the data recorded is carried out. If all data has been recorded correctly, the message ”Verify OK” appears. If a comparative error appears, a corresponding error message is given out.

Note: It is not permitted to remove the OTP device from the socket during the programming procedure is in progress, otherwise the IC could be damaged.

9.5 Error Messages

Cannot open resource file

The resource file MARC4OTP.TVR was not found. Check that the resource file is available in the MARC4 base directory.

Cannot open device file

The MARC4OPT.DEV file for describing the module was not found. Verify that this file is available in the MARC4 base directory.

Illegal device file

The file for describing the module does not have the correct format. Install the program again.

Unable to create init file

Check that the MARC4OTP.INI file is not write protected.

Parallel port LPTX not available

The selected parallel port is not available. Check that the port named is installed correctly.

File too large for selected device

You have tried to load a file which is too large for the chosen module. Adjust the compiler options to the correct memory size and compile the program again.

File smaller than ROM size

You have tried to load a file which is smaller than the memory of the chosen module. Set the compiler options at the correct memory size and compile the program again.

Not enough memory for safety pool

There is not enough memory available to load the file. Close the programs which are no longer required.

Cannot open file

An invalid file name has been entered, or the file is damaged.

Verify error at address $XXXXXX

By comparing the memory contents with the actual file, a comparative error was discovered. Check the cabel connections to the PC and check whether the correct parallel port has been set.

Blank check error at address XXXX

An error appeared during the preliminary tests to check if the OTP-EEPROM is clear. Check the cabel connections to the PC and check whether the correct parallel port has been set.

9.6 Elimination of Errors

Table 1. Error debugging

Error	Elimination
The LED ”Prog” does not light up during the programming procedure	Check the connection to the PC Check whether the correct parallel port has been set
The LED ”Power” is not lit up	Check whether the power supply is plugged in Check the fuse SI1
The menu functions ”Programming” and ”Verify” are deactivated	Load an object file first
The menu function for adjusting the type of module is deactivated	Close all windows which are open

9.7 Description of the Parallel Port Signals

Table 2. Parallel port signals

Pin Nr.	Name (Printer)	Symbol	Name (MARC4)
1	STROBE	TE	TE
2	D0	D0 OUT	BP00
3	D1	D1 OUT	BP01
4	D2	D2 OUT	BP02
5	D3	D3 OUT	BP03
6	D4	TCL	TCL
7	D5	NCYCLE	BP10
8	D7	NREAD	BP11
9	D7	NWRITE	BP12
10	ACK	D3 IN	BP03
12	PE	D2 IN	BP02
13	SLCT IN	D1 IN	BP01
14	AUTO FD	NRES	NRES
15	ERROR	D0 IN	BP00
16	INIT	PWON	

Note: Please be aware that some lap-top computers do not support all control signals required to operate the OTP programmer.

Schematic Diagram

For more detailed information on the OTP programmer, see the corresponding schematic diagram which is attached to the hardware.

Members of the MARC4 Family

M43C505 - Low-Current 3- and 5-V Solution for Consumer Applications

- Wide supply voltage range (2.4 V to 5.5 V)
- Very low current consumption
- 4096 x 8 bit ROM, 253 x 4 bit RAM
- 16 programmable I/Os
- 2-MHz fast system clock (1 MIPS)
- 32-kHz crystal oscillator
- 20 4 LCD temp.-compensated drivers
- 2 external/ 3 internal interrupt sources
- Prescaler/ interval timer
- Internal POR and brown-out

Existing applications comprise temperature measurement and -control, battery charging, bicycle computers, timers, radio-controlled clocks and CD players.

Existing software modules for time keeping, calendar, stop watches, display drivers for various multiplex rates, accurate dual-slope temperature measurement and interface software for TEMIC's radio-controlled clock receivers are part of the comprehensive qFORTH software library.

A power-saving sleep- and stop mode increases battery life time significantly in hand-held applications, while offering 1 MIPS computing power during active time. Internal POR, oscillator and pull-up/-down resistors simplify PCB layout and minimize system costs.

Software is free of charge for these applications which increases the confidence level and reduces the time-to-market for new developments.

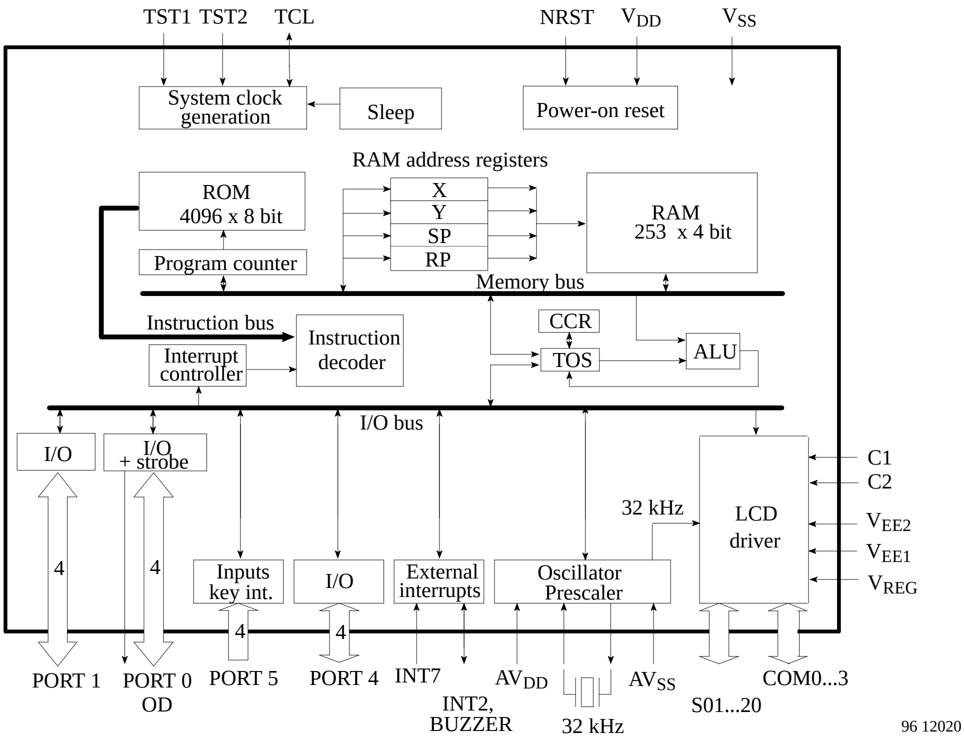


Figure 1. M43C505

TEMIC
Semiconductors

M44C260 - Perfect Solution for Security and Access Control

- Wide supply voltage range (2.4 V to 6.2 V)
- Very low sleep current
- 4 KByte ROM, 256 x 4 bit RAM
- 128 bit EEPROM on board
- 18 programmable I/Os
- 4.2-MHz fast system clock (FLL)
- 32-kHz crystal oscillator
- 6 interrupt sources
- Prescaler/ interval timer
- Multi-functional timers/ counters incl. IR remote control carrier generation
- Watchdog and POR
- OTP M48C260

The M44C260 is especially optimized for IR remote control and security and access control applications, e.g., for automotive and industrial applications.

The on-board 128-bit EEPROM offers the capability of storing and changing identifiers as well as security codes. Any application which requires the ability to store a small amount of data will also benefit.

The multi-function timer/counter modules which are also on–board include modes to directly generate the signal for an IR transmitter device such as TEMIC's U426B.

The wide supply voltage range combined with the very small current consumption increases battery life time in mobile applications.

The OTP M48C260 simplifies and reduces the development time.

For detailed information please refer to TEMIC's "Automotive Safety and Convenience Data Book 1996".

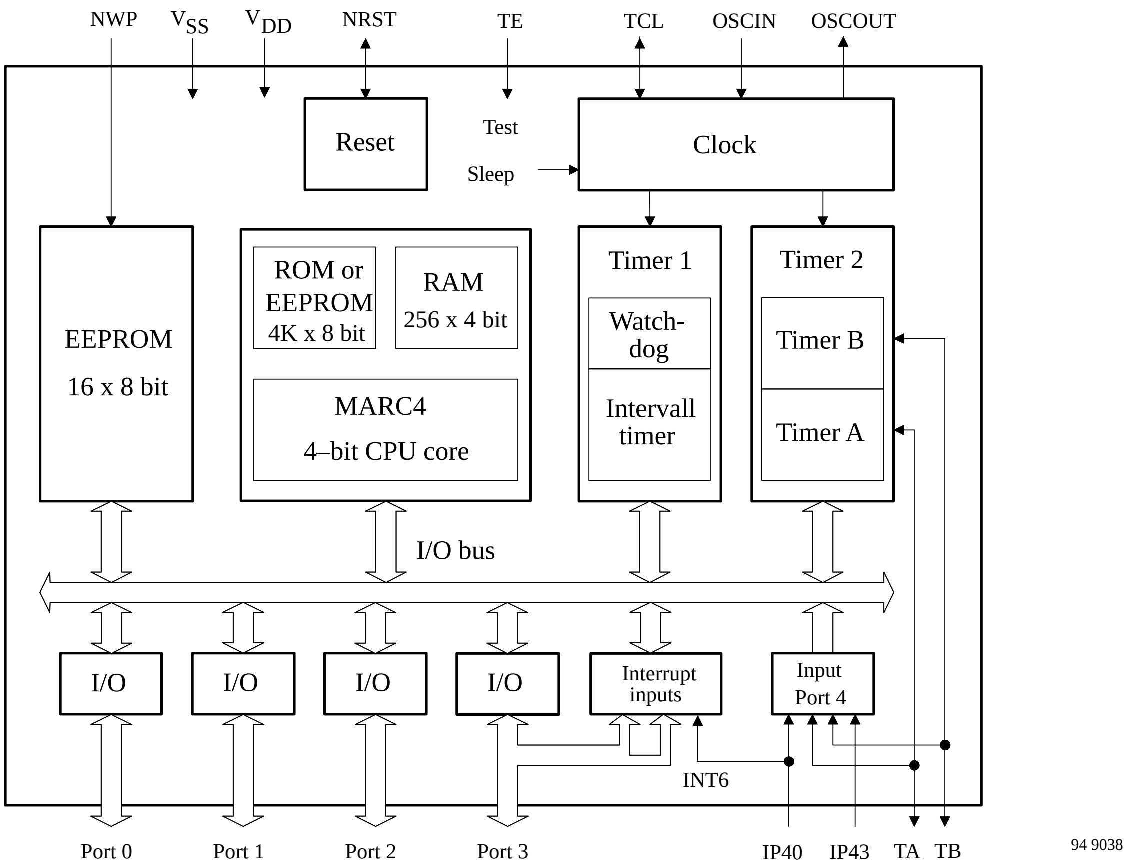


Figure 2. M44C260

M44C090/092 - Low-Current Solution for Wireless Communication

- Software selectable system-clock sources, crystal oscillator, external clock, RC oscillator with/ without external resistor
- Wide supply voltage range (1.8 V to 6.2 V)
- Very low sleep current
- 2/4 KByte ROM, 128 x 4-bit RAM
- 512 bit EEPROM optional
- 12/16 programmable I/Os
- 32-kHz crystal oscillator
- Up to 7 external/ 7 internal interrupt sources
- Prescaler/ interval timer
- 2-wire serial interface
- Multi-functional timers/ counters incl. IR/ RF remote control carrier generation
- Watchdog, POR and brown-out function
- OTP M48C092
- SO8 package (M44C090)

The two MARC4 products M44C090 and M44C092 offer the highest integration for IR and RF data communication and remote control. These controllers are optimized for the transmitter as well as the receiver applications.

TEMIC's system know-how was used to integrate the modulator into the M44C090 and the modulator as well as the demodulator for commonly-used wireless protocols into the M44C092.

Both controllers perfectly match the RF front end device U2740B and the IR driver chip U426B. This - along with the very small SSO package and the approach to minimize the number of external components - leads to extremely compact remote control units, e.g., for electronic keys. Finally, the very low current consumption and the extended supply voltage range optimizes battery life time.

Development is supported by the OTP M48C092 which covers the features of the M44C092 and both includes the performance of the M44C090.

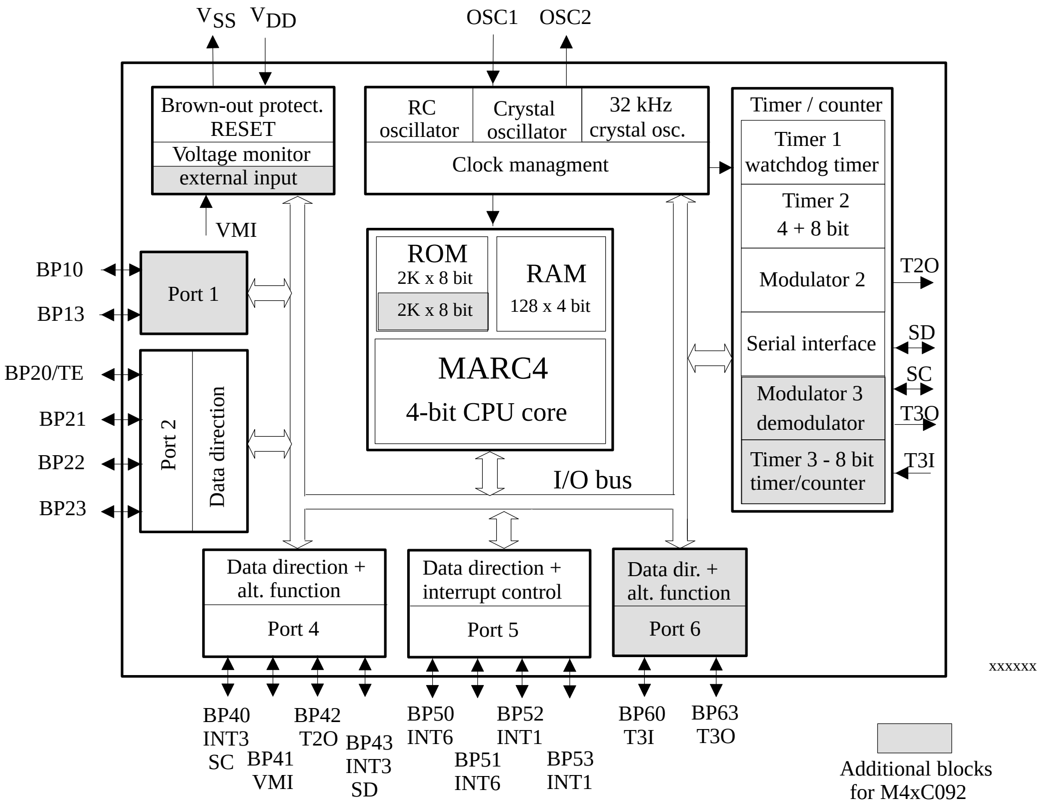


Figure 3. M44C090/092

M44C510 - Flexible and Powerful Solution for Embedded Control

- 4 mask-selectable system-clock sources, crystal oscillator, ceramic resonator, RC oscillator with/ without external resistor
- Wide supply voltage range (2.4 V to 6.2 V)
- Very low current consumption
- 4 KByte ROM, 256 x 4 bit RAM
- 32 bitwise-programmable I/Os
- High-current outputs
- 32-kHz crystal oscillator
- 10 external and 4 internal interrupt sources
- Prescaler/ interval timer
- Two 8-bit multi-functional timers/ counters
- Watchdog timer, internal POR and brown-out
- Minimum external components
- Very small package (SSO44)

The M44C510 is a solution for embedded control applications. Various mask options provide an optimum price-performance ratio for the system.

Due to the pull-up/-down, push-pull and open-drain functions of the bit–wise programmable I/Os, external components are unneccessary. LEDs and relays can be connected directly to the M44C510 by using up to eight I/Os driving 20 mA each. Mask selectable clock sources cover a wide range of application requirements. Watchdog, POR and a brown-out function monitors correct operation. More than ten timer/counter modes offer D/A conversion, event counting, 16-bit modes and even melody modes. The wide supply voltage range along with the very small current consumption supports battery–powered systems.

Software modules available include keyboard software, LCD and LED display driver, serial port protocols, radio–controlled clock decoders and timer as well as temperature measurement modules. For detailed information please refer to "M44C510 Keyboard Application Design Guide 06.96"

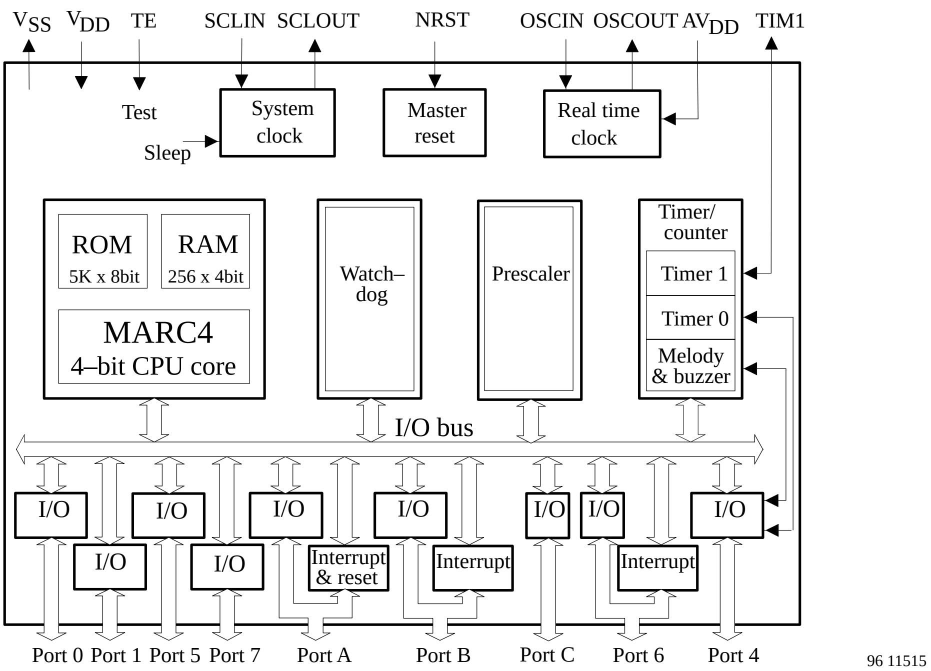


Figure 4. M44C510

M44C588 - Versatile High–End Controller for General Purposes

- Various mask-selectable system clock sources to define application-specific system price/ performance ratio
- Dual clock mode for minimum current consumption
- Wide supply voltage range (1.8 to 6.2 V)
- 9 KByte ROM, 512 x 4 bit RAM
- Up to 32 I/Os incl. high-current ports
- 32-kHz crystal oscillator
- Up to 32 4 LCD segments
- Prescaler
- 8 external and 5 internal interrupts
- Watchdog, POR and low battery detection for enhanced system security
- Synchronous 8-bit serial port
- Multi-function timer/ counter incl. IR/ RF remote control carrier generation

High-end, battery-powered consumer applications such as bicycle computers, feature watches, diver computers and high-end, radio-controlled clocks/watches which all require both computing power and low current consumption will benefit from the M44C588.

The dual clock mode and core frequencies of 4 MHz (2 MIPS) on the one hand and 32 kHz slow operation/ sleep mode (consuming only micro-amps) on the other hand make the M44C588 the best solution for these tough requirements.

The programmable I/Os with pull-up/-down options, integrated oscillators, 20-mA drive capability, internal watchdog, POR and low battery detection minimize the number of system components, resulting in reduced system costs and PCB size. The integrated temperature–compensated display drivers for up to 128 LCD segments enable even sophisticated display solutions. Data transfer to external storage devices such as serial EEPROMs is simplified by the serial port.

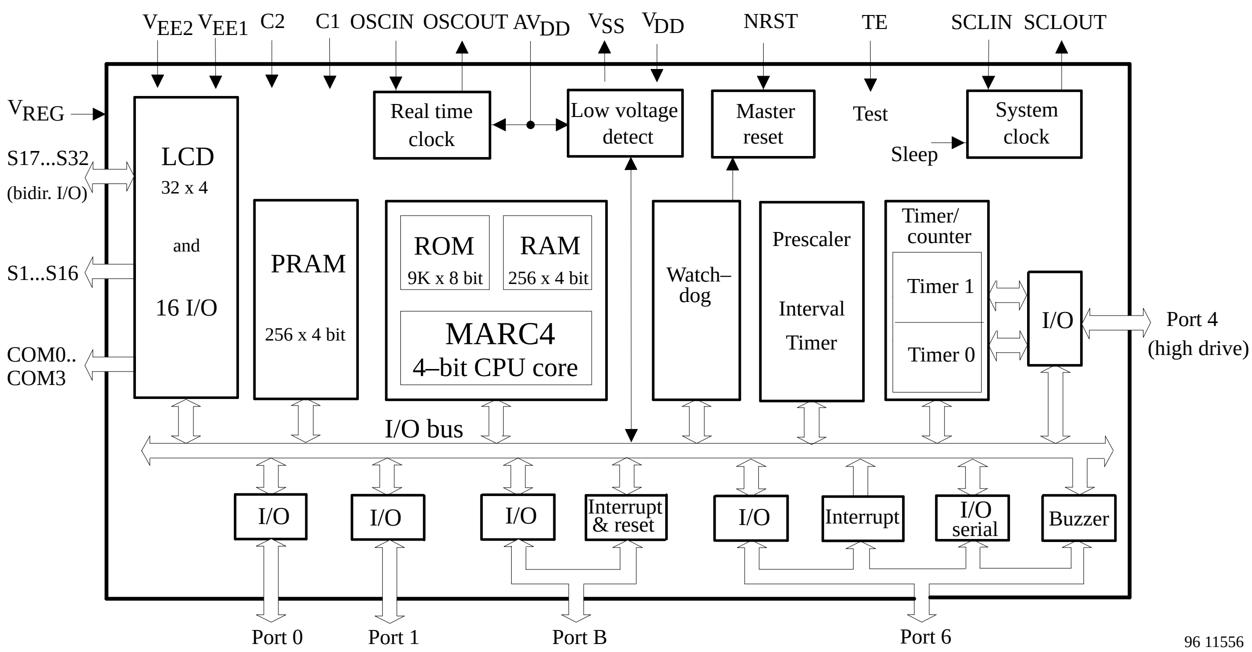


Figure 5. M44C588

M44C636 - Perfect Solution for Low-Current Applications

- 1.2 V to 2.2 V/ 1.8 V to 3.6 V (mask opt.)
- < 1 mA sleep mode current, 200 mA active current
- On-chip RC system clock oscillator
- 4 KByte ROM, 253 x 4 bit RAM
- 16 programmable I/Os
- 32-kHz crystal oscillator
- 20 4 temperature-compensated LCD driver segments
- Prescaler/ interval timer
- Two independent 8-bit timers/ counters
- Watchdog and POR

The M44C636 is pushing the limits of low-current consumption to the values of the discharge of batteries. By combining sleep and active periods, system currents of less than 2 µA can be designed. The M44C636 is therefore suitable for applications such as feature watches, radio-controlled clocks/watches, timers powered by back-up capacitors and even telecom applications such as telephone-rate counters directly powered by transmission lines.

Mask options adjust the extended supply voltage range of the M44C636 to 1.5-V or 3-V batteries. For 3-V applications, an internal voltage regulator powers the core, reducing the active peak current to 200 µA. The typical system current in watch applications is under 2 µA.

Two multi–function timers/counters and motor output drivers support 3-V watch applications including motor-pulse chopping. Internal watchdog, brown-out function and POR supervise correct operation.

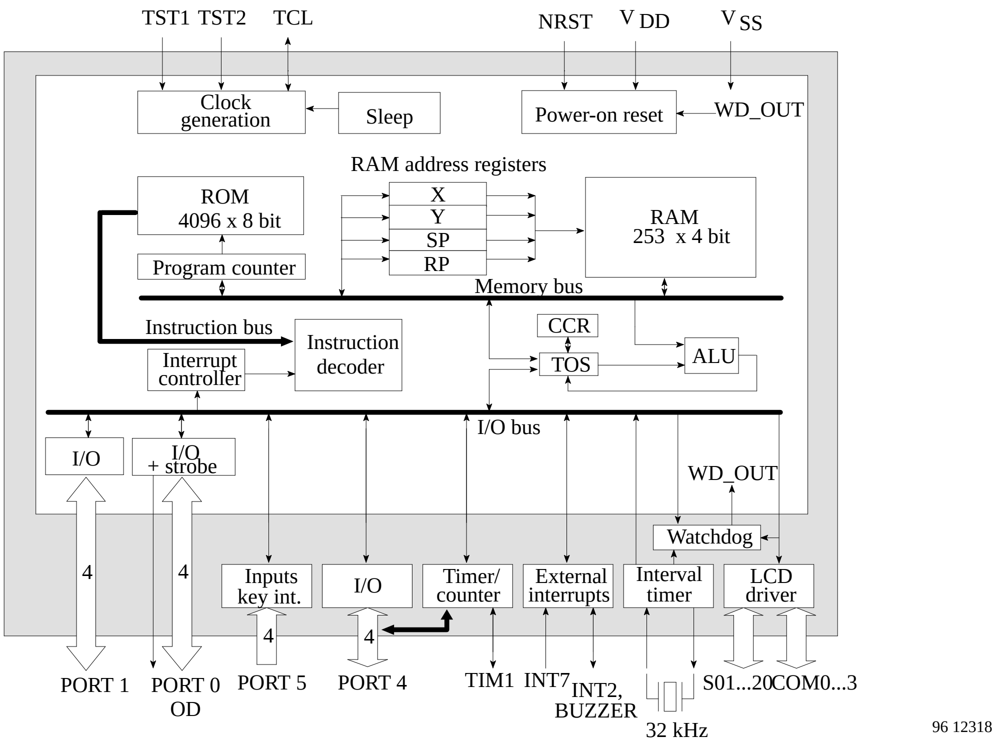


Figure 6. M44C636