EM66xx 4-bit Micro controller family

Contents of this binder :

- ❏ **Development System Manual**

- ❏ **Peripheral Interface Modules Manual**

- ❏ **LCD Editor Module Manual**

- ❏ **MFP Programming Interface Manual**

Version 4.5, October 2005
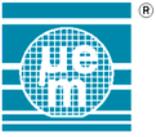
EM MICROELECTRONIC - MARIN SA

A COMPANY OF THE SWATCH GROUP

EM66xx Microcontroller

Development System Manual

❑ **Software Development System**

❑ **EM66xx Programming Model**

❑ **EM66xx Instruction Set**

EM66xx

Development System

Table of Contents

EM66xx

Development System

1. EM66xx Microcontroller Development System

The EM66xx microcontroller development system is a set of software and hardware tools used for the development of applications for the EM66xx family of 4 bit microcontrollers from EM Microelectronic Marin SA.

The development system is composed of a PC based software and a universal in-circuit emulator. The development system software offers a complete integrated working environment allowing project definition, editing, assembly, software simulation and connection to an universal in-circuit emulator for in-circuit testing.

The hardware emulation of the complete family of 4 bit microcontrollers is achieved with a configurable universal in-circuit emulator.

This document describes the installation and utilisation of the software tools.
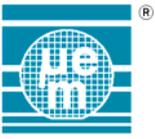
2. Software Development System

2.1. System Requirements

The EM66xx 4-bit microcontroller development system requires the following minimum requirement:

- A PC with an 80486 or higher processor, running Microsoft Windows 98, ME, XP, Windows NT4.0 or 2000.

- A VGA monitor (SVGA recommended)

- A Hard disk with at least 4MBytes of free space

- 16 megabytes of available memory (32 megabytes or higher is strongly recommended).

- A serial port capable of communication at 9600 or 19200 baud (for connection of the universal in-circuit emulator).

- A CD-ROM drives for the installation of the software development system.
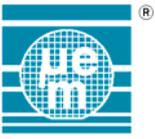
2.2. Installation

To install the EM66xx software development system:

1. Insert the CD-ROM into your CD drive.

2. Wait a few moments for the autorun facility to activate.

3. Select "Dev. System 4-bits"

4. Follow the on-screen instructions to install the software.

During the installation process the following files will be copied to the target directory specified by the user :

- EMMON.EXE   :The core of the development system
- EMMON.INI   : The development system initialisation file
- V66xxASM.EXE   : The assembler module.
- V66xxDIS.DLL   : Disassembler functions
- INSTRUCT.INI   : The instruction definition file
- CONDIREC.INI   : Conditional directive definition files
- DIRECTIV.INI   : Directive definition
- OPERATOR.INI   : Operator definitions
- ERRORMSG.INI   : Error message definitions
- LCDEDIT.EXE   : LCD Editor main programm
- STDV66xx.DLL : The standard interface for the software simulation
- 66xx_LCD.DLL : LCD simulation module
- 66xxxxxxx.DLL : Peripheral definition modules (one per simulated controller)
- others

2.3. Developing an EM66xx Application

This chapter describes in a general manner the development of an application using the EM66xx software development system. The development system is a totally integrated environment you can use to develop your applications for the whole EM66xx family of microcontrollers. The software development system is comprised of a project window, a text editor, an assembler a disassembler and a software simulator, which also serves as, interface to the universal in-circuit emulator.

The development process can be broken down into two stages: creating the application and developing the application. The first step, creating an application, comprises the creation of a project definition file, which defines the project parameters. Projects refer to the source file(s) which make up the application as well as the definition of the target controller of the application to be used during the simulation / emulation process. The way in which the project definition interacts with the development process is summarised below.

```
┌─────────────────────────┐      ┌──────────────────┐      ┌───────────────────────────┐
│   Project Definition    │      │ Main Source File │      │      Include files        │
│        (*.PRJ)          │ ───▶ │                  │ ◀─── │                           │
│ Main Assembly source file│      │     (*.ASM)      │      │(registers / constants /   │
│    Target controller    │      │                  │      │        library)           │
└─────────────────────────┘      └──────────────────┘      └───────────────────────────┘
                                          │
                                          ▼
                              ┌────────────────────────┐
                              │ EM66xx Assembler Module│
                              └────────────────────────┘
                    ┌──────────────┬────────────┴────────┬────────────────┐
                    ▼              ▼                      ▼                ▼
           ┌─────────────┐ ┌─────────────┐ ┌────────────────────┐ ┌─────────────────┐
           │ Binary File │ │  List File  │ │Symbol reference file│ │ Assembly Errors │
           │   (*.BIN)   │ │   (*.PRN)   │ │      (*.SYM)       │ │     (*.ERR)     │
           └─────────────┘ └─────────────┘ └────────────────────┘ └─────────────────┘
                    └──────────────┼────────────────────┘
                                   ▼
                         ┌──────────────────┐
                         │    VICE / ICE    │
                         │    Emulation     │
                         └──────────────────┘
```
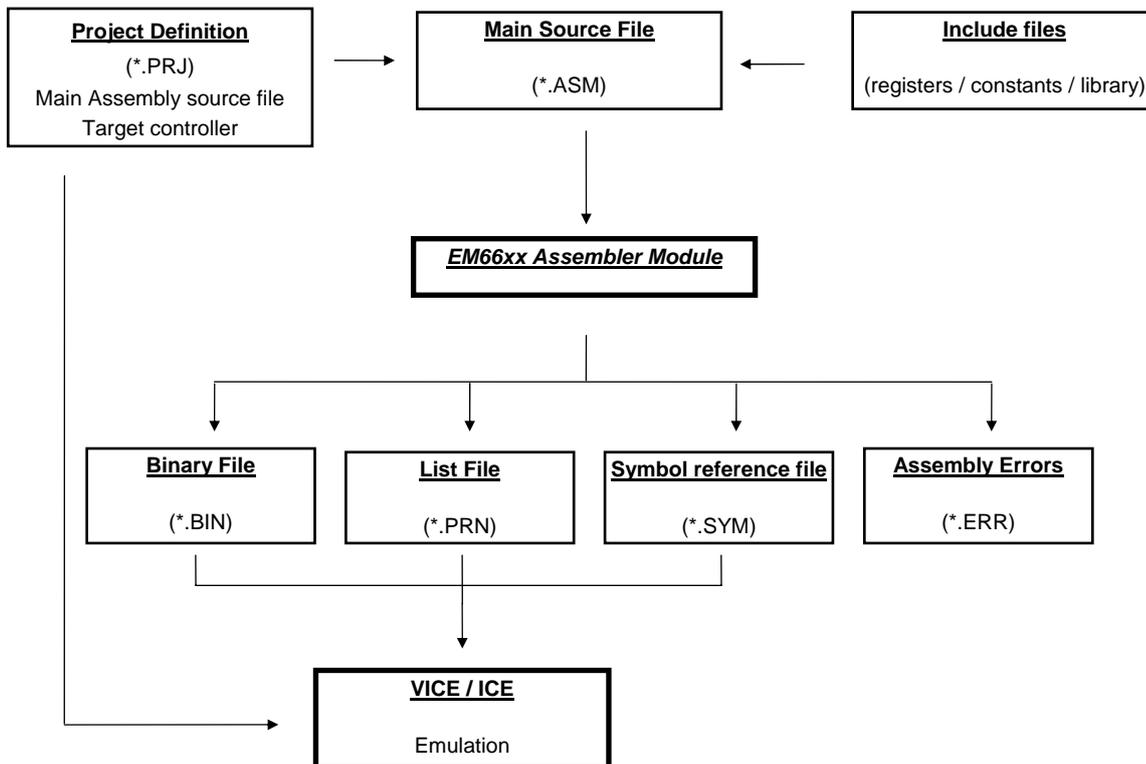
Fig. 1 EM66xx Development system structure

By using a project definition file it is possible to select once the global parameters which will then be used by the development system.

Once the project has been defined the source files can then be created. The source files used by the EM66xx development system are text files, which by default have the extension *ASM*. They can created and edited with the editor incorporated the development system or they can be edited with any external text editor. The use of the internal editor has the advantage that there is an interaction between the assembler output window and the text editor. This interaction allows very rapid error

analysis since the software engineer can directly go to the source line which created the assembly error by double clicking on the error message. The development system allows several documents to be open at the save time. The maximum number of documents open is determined by the memory size of the system, however, at present each source file is limited to a maximum size of 32Kbytes. The internal editor supports fully the cut and paste options of windows.

The structure of a generic EM66xx assembler program is shown below. In the following example the naming convention used for the program labels is not obligatory but simply represents the function of the program block.

```
;       GENERIC.ASM
                                    ;-----------------------------------------------
                                    ; Register definition include files user defined and / or
                                    ; controller dependent
                                    ;-----------------------------------------------
        INCLUDE EM66xxREG.ASM  ; Include register definitions

ZERO        EQU      00H            ;-----------------------------------------------
MAX         EQU      0FH            ; Definitions of constants
                                    ;-----------------------------------------------


                                    ;-----------------------------------------------
                                    ; Definitions of variables
                                    ;-----------------------------------------------
RAM0        EQU      00H            ; min RAM
RAMMAX      EQU      05FH           ; max. RAM


                                    ;-----------------------------------------------
                                    ; Definition of the program counter value
                                    ;-----------------------------------------------
            ORG      0H             ; the following code is based at address 0


                                    ;-----------------------------------------------
                                    ; Assembler program body
                                    ;-----------------------------------------------

RESET:      JMP      MAIN           ; Boot address - jump to core of program
                                    ;-----------------------------------------------
                                    ; INTERRUPT HANDLER STARTS HERE
                                    ;-----------------------------------------------
HANDLER:
            STA      RAM0           ; save the value of the accumulator at
                                    ; the start of the interrupt routine
            ...      ....
            LDR      RAM0           ; restore Accu before leaving interrupt handler
            RTI                     ;-----------------------------------------------
                                    ; End of Interrupt Handler
                                    ;-----------------------------------------------
                                    ;-----------------------------------------------
                                    ; Main program core
                                    ;-----------------------------------------------
MAIN:       ...      ....
            ...      ....
            ...      ....


            INCLUDE APPLISUB.ASM   ; Include sub routines (possibly library modules)

                                    ;-----------------------------------------------
            END                     ; signifies the end of the source code any code placed
                                    ; after this directive will be ignored
                                    ;-----------------------------------------------
```

Fig. 2 EM66xx Generic Program structure

Once the application is assembled without errors it can be tested. This can be done either using the software simulator or the universal in-circuit emulator. Both systems are operated from an identical interface and provide the following development tools.

- Source level debugging of the application.

- Instruction break selection without reassembly.

- Conditional data break selection without reassembly.

- Instruction tracing (including source code)

- Variable watch functions.

All the debugging functions such as instruction breakpoints, instruction traces and conditional data breaks can be modified without the reassembly of the application.

2.4. The Project menu

The definition of new projects and the opening of existing projects is achieved from the « project » option of the main menu bar which is shown below.
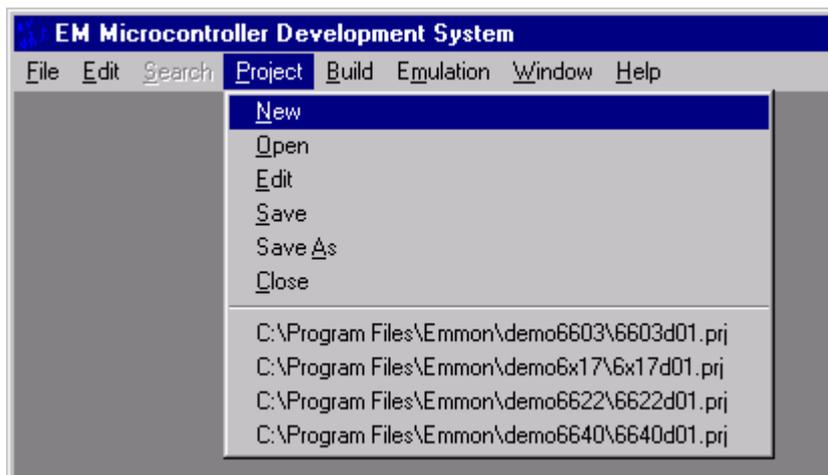
Fig. 3 Project Menu

At the bottom of the project menu are the four most recent projects opened in the development system. If the desired project is in this list it can be reloaded by selecting it from this list. The other options in the project menu are the following:

When starting a new project, first create your main assembler source file (see 2.5), then create your new project (see 2.4.1)

2.4.1. New

This option opens the dialogue box shown below enabling the definition of the new project parameters. The main assembler source file must be created before this operation (see chapter 2.5).
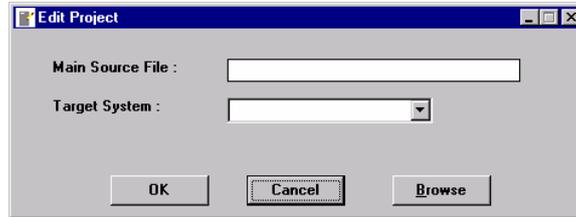
Fig. 4 Project definition window

In the field « Main Source File » the complete name (path and filename) of the main source file of the project should be entered. The main file can also be selected by using the « Browse » button. This button opens a dialogue box that enables the selection of the file. The field « Target System » is a drop down selection box which allows the selection of the target controller. The number of target controllers defined in the system is dependent on external configuration modules which are dynamically linked to the application at execution time. In this way the development system can be updated for new controller by just adding the new controller configuration file. It should be noted that a project is not automatically saved to permanent storage when it is created. This should be performed using the « Save » or « Save As » options of the project menu described below.

2.4.2. Open

This option opens a dialogue box which allows an existing project to be opened. If a project is already open it is automatically closed before the new project configuration is loaded.

2.4.3. Edit

This option opens the dialogue box which allows the modification of the project parameters (as described in section 2.4.1).

2.4.4. Save and Save As

These two options allow the current project definition to be saved to permanent storage.

2.4.5. Close

This option closes the current project definition without saving the changes.

2.5. The File Menu

The menu options used for manipulation of the source files are found in the « File » option of the main menu bar, which is shown below. The four most recent source files are shown at the bottom of the file menu allowing them to be quickly reopened.
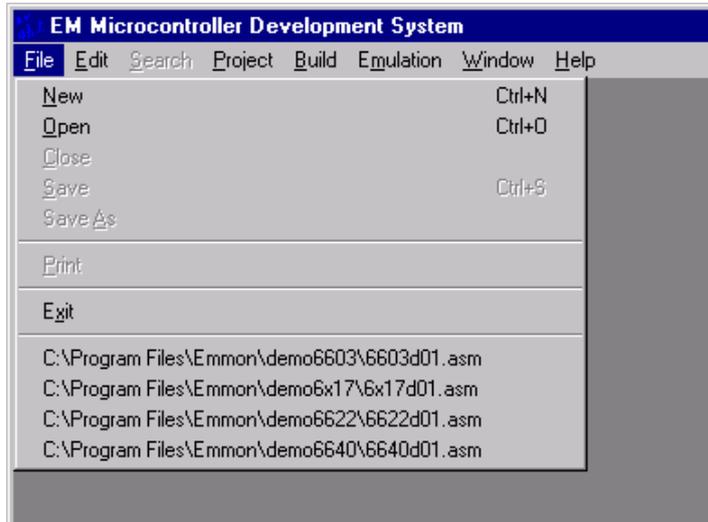
Fig. 5 File Menu

2.5.1. New

Creates a new source file. The document remains unnamed until it is saved, This option is also available with the button in the application toolbar.

2.5.2. New Template

Used to create a new assembler file from a basic skeleton, The document must be renamed when it's saved.

2.5.3. Open

Open an existing source file. This option is also available with the button in the application toolbar.

2.5.4. Save

Save the currently active source file to permanent storage. If the document is unnamed the menu option « Save As » is automatically invoked. This option is only available when a source file is open and is the active window.

2.5.5. Save As

Save an unnamed document to permanent storage, or save an existing document under a new name. This option is only available when a source file is open and is the active window.

2.5.6. Close

Close the active document. This option is only available when a source file is open and is the active window.

2.5.7. Print

Prints the current source file to the default system printer.

2.5.8. Exit

Allows the user to quit the development system.

2.6. Edit Menu

The edit menu contains the options related to the cut and paste functions as well as the definition of the fonts for the different types of windows.
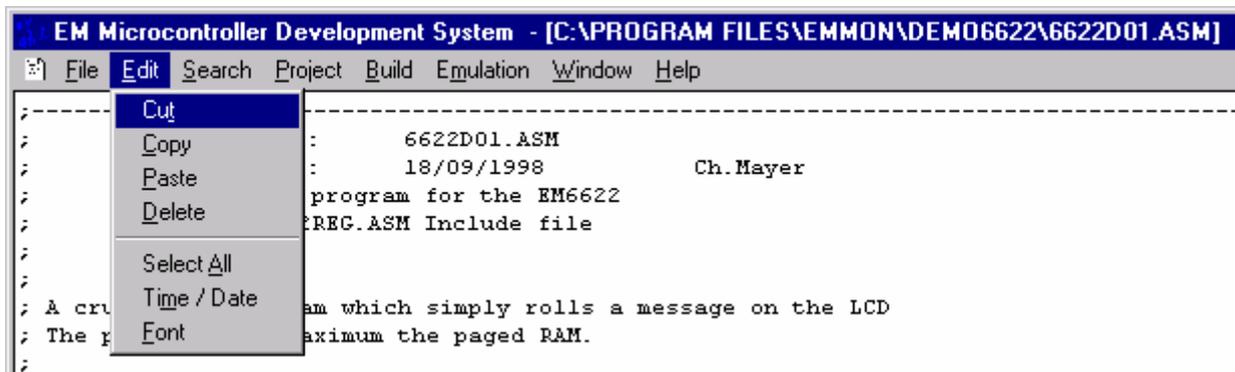
Fig. 6 Edit Menu

2.6.1. Cut

Cuts the selected text from the current editor window to the clipboard.

2.6.2. Copy

Copies the selected text from the current editor window to the clipboard.

2.6.3. Paste

Insert the clipboard contents into the current editor document at the cursor position.

2.6.4. Delete

Delete the selected text from the current editor window.

2.6.5. Select All

Selects the complete contents of the current editor window.

2.6.6. Time Date

Insert the time and date at the cursor position in the active editor window.

2.6.7. Font

Select the font type and size for the active window. This option applies to the editor window, the trace buffer window, the emulator control window and the emulator watch window. For the editor it is not possible to individually set the font type for each document and the current selection will be applied to all the text editor windows.

2.7. Search Menu

This menu is only active when an editor window is active. It allows the searching of a text string in the current editor window.

2.8. Build

The build menu contains the options which allows the assembly of an existing source file as well as the disassembly of a binary file generated by the development system.
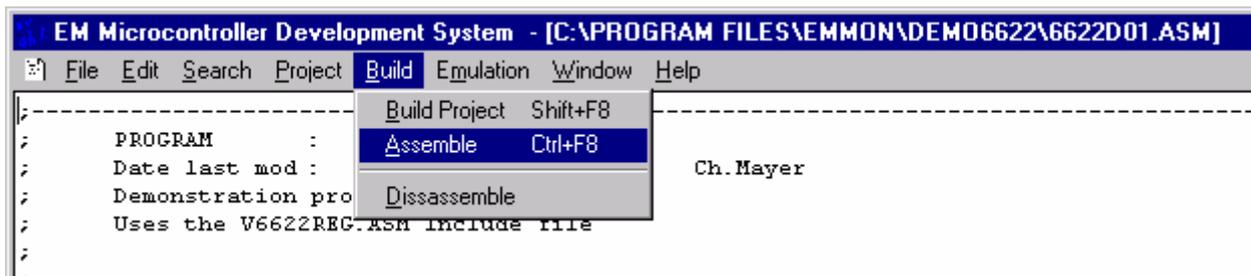
Fig. 7 Build options menu

2.8.1. Build Project

This option builds the current project without prompting for the name of the file to assemble.

2.8.2. Assemble

Assembly of a program is achieved using the « Assemble » option of the menu. Once this option is selected a dialogue box is opened requesting the name of the file

to be assembled. By default the main source file as defined in the current project is proposed. However, it is possible to override the project definition and assemble any source file. Once the file is selected Assembly is started with the « OK » button. The assembly is performed as a background task with the assembler generated messages shown in the output window. An example of the result of an assembly is shown in Fig. 8.

In this example 1 error message was generated. The error was number 200 (undefined variable or constant) and was generated in line 111 of the source file « *6622DEMO.ASM* ». To edit the source file which generated the error double click on the error definition in the output window. If the source file is already open the cursor is positioned to the line which generated the error. If the source file is not open it is automatically loaded into an editor window and then the cursor is automatically positioned to the line, which generated the error.
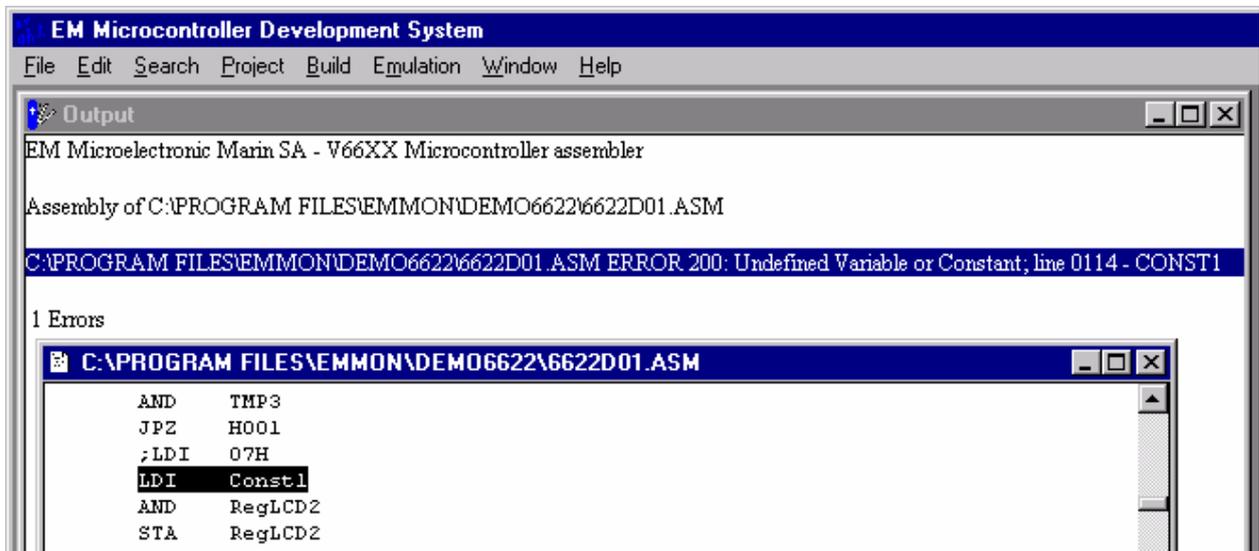
Fig. 8 EM66xx Assembly error output

If no assembly errors are encountered, then the last line printed on the output window will be « Prog-Size = ... » telling the user the length of his program (in Nr. of Instructions).

Very Important : the maximum user program size should be at least 5 words smaller then the ROM size of the used μC ; the 5 last adresses have to be used by EM- Marin for testing purposes. For The EM668x a minimum of 37 words must be free to place some basic manufacturing test routine.

2.8.3. Disassembly

The option disassembly uses a binary file generated by the assembler to regenerate an assembler source file (ASM file), a list file (PRT file), a variable list file (SYM file), a cross reference file (XRF file) as well as a ROM state file (STA file). The source file generated can be modified and reassembled in the same way as a normal source file. The list file and the variable list file have the same format as those generated by the assembler and can be reused in the simulator and emulator. The cross reference file gives the line number for each variable and label referenced.
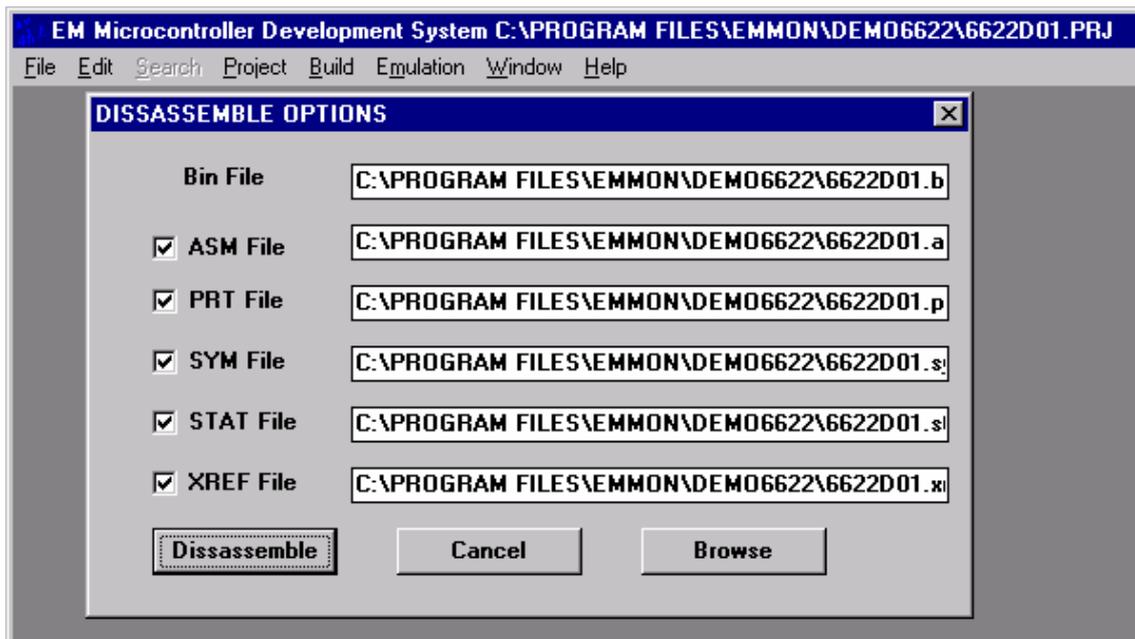
Fig. 9 Disassembly options

The file names are dependent on the active project. If no project is defined the filenames must be entered. The generation of each data file can be activated or deactivated by using corresponding the check box. The browse button opens a standard dialogue box which allows the user to select the required file. The disassemble process is initiated with the Disassemble button. Once the process is terminated the button Annuler allows the user to quit the disassembler option box.

WARNING : If a source file is selected it will automatically be overwritten during the disassembly process.

2.9. Emulation

It is possible to emulate EM66xx microcontroller programs in two ways. The first is by a procedure referred to *VIRTUAL EMULATION (VICE)*, where all the functions of the microcontroller are simulated by a software module. The second is referred to as *IN-CIRCUIT EMULATION (ICE)* and uses a configurable universal in-circuit emulator. In both cases the emulator interface is practically the same, offering the following features :

- Source level debugging

- Data watch and data breakpoint functions

- Instruction breakpoints

- Instruction trace functions

- Trace break functions

The user interface with the ICE and VICE emulators is composed of four windows and is shown below.
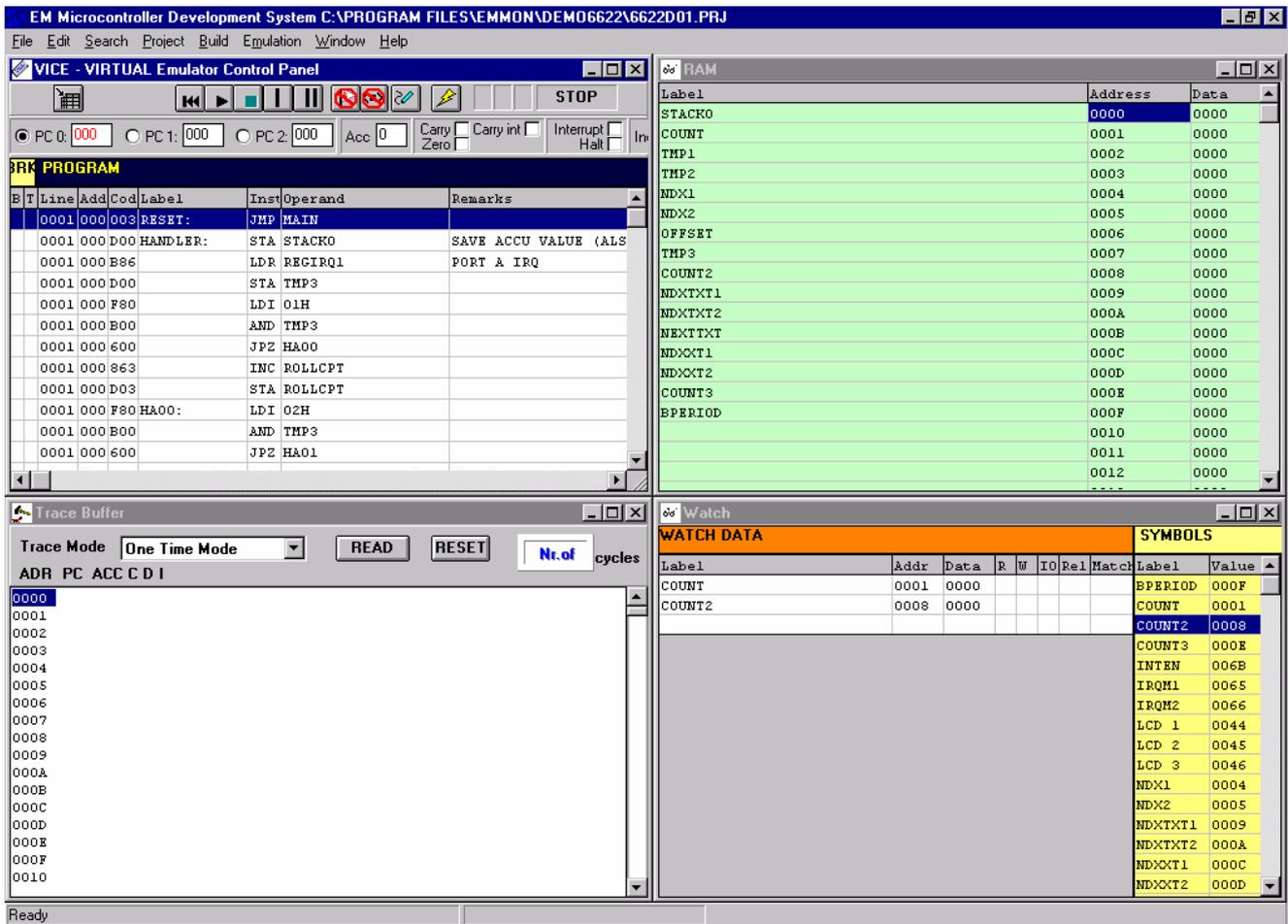
Fig. 10 The EM66xx Development system emulator interface

The most important window is the emulator control window. In this window the source code is loaded and executed, instruction breakpoints can be set, trace points can be set. The second window is the Watch window which contains a list of the symbol definitions, the currently watched variables as well as the conditional break points. The third window is the trace buffer window, and is only activated if the trace enable button is activated. This window displays the contents of the trace buffer and allows the printing of the contents. A forth window is also active if the VICE system is running. This window is a virtual representation of the micocontroller IO ports. The developer can see the state of the output ports as well as modify the state of the input ports. This section of the interface is dependent on the type of microcontroller defined in the project definition and consequently is not described here. For a more detailed definition see the manual corresponding to the microcontroller being emulated.

2.9.1. Emulator Control Window

The virtual emulator as well as the in-circuit emulator are both controlled from the same interface. Consequently it is only possible to work with one system at a time. The VICE / ICE control window is shown below.
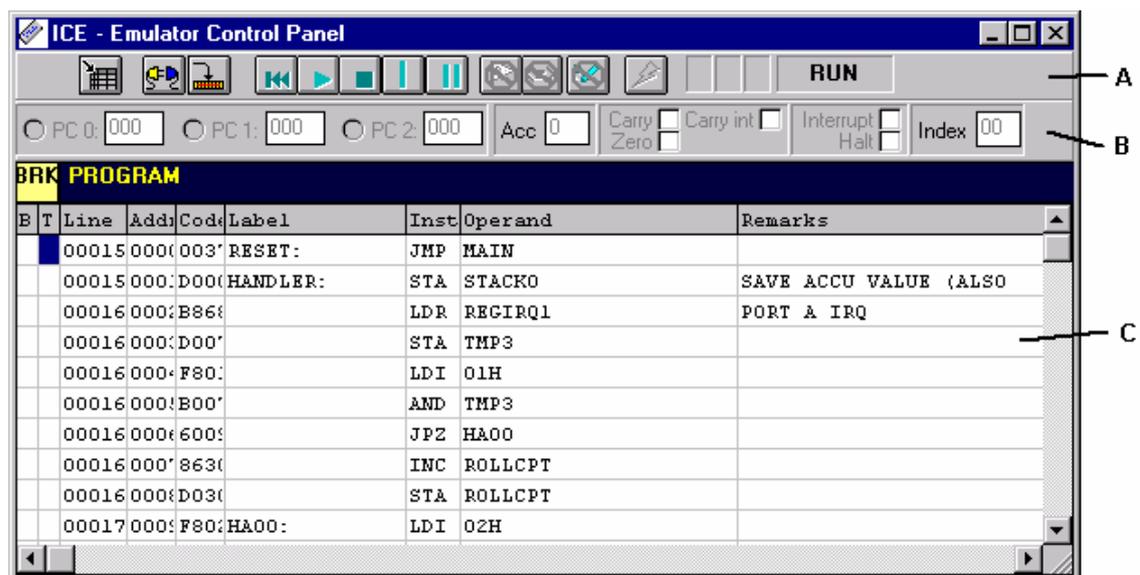
Fig. 11 The emulator control window

The control window is composed of three sections, the execution toolbar (A), the microcontroller status panel (B) and the source code buffer (C).

2.9.1.1.Execution toolbar

The execution toolbar contains all the buttons necessary for the control of the VICE / ICE as well as showing the VICE / ICE break states.

2.9.1.1.1. Connect/Disconnect Button (ICE ONLY)

This button shows the state of the connection between the emulator control window and the in-circuit emulator. When the in-circuit interface is selected the control window tries to connect itself to the in-circuit emulator using the communication parameters specified in the ICE config menu options. If it does not succeed, the control window remains unconnected. The user can connect and disconnect the Monitor by clicking on this button. This button is not available during virtual emulation.

2.9.1.1.2. Load a binary file into the emulator control window

This button opens a dialogue box which allows the user to select the binary file which will be loaded into the emulator control panel. By default the binary is that defined in the project definition file. However, it is possible to override this selection and load any binary file which has been assembled using the EM66xx assembler. Once the binary file is selected the source code buffer of the controlled is filled with the corresponding source of the binary file. It should be noted that every time a program is reassembled it should be reloaded into the emulator control window before the program changes take effect.

2.9.1.1.3. Download program to emulator (ICE ONLY)

This button downloads the source code buffer in to the in-circuit emulator. It also sets any breakpoints or trace points set by the programmer. It should be noted that every time a new or modified program is loaded into the emulator control window source buffer it must be downloaded into the in-circuit emulator before any modifications will take affect. The down loading of a program does not reset the state of the emulator. Consequently it is necessary to reset the program with the « Reset Execution Button » if the program is to be restarted.

2.9.1.1.4. Reset Execution Button

The meaning of this button depends on the working mode (VICE or ICE) as follows :

in VICE-mode : it produces a general RESET (cold reset), that is, all registers are reset to initial values and the RAM is zeroed.

In ICE-mode : it only produces a program reset (warm reset), that is, only stack pointer, program counters, index registers are initialized.

2.9.1.1.5. Continue Button

When the emulator has suspended its execution, this button restarts the program execution at the address pointed to by the active PC.

2.9.1.1.6. Stop Button

When the emulator is running, this button enables the user to suspend the execution of the application program. The emulator stops execution before the instruction pointed to by the active PC.

2.9.1.1.7. Step Button

When the emulator has suspended its execution, this button enables the user to execute a single instruction. The instruction is that pointed to by the active program counter.

2.9.1.1.8. Animation Button

When the Emulator has suspended its execution, this button enables the user to activate the animation mode of the emulator. In this mode 1 instruction is executed every 1 second until the mode is deactivated with the stop button. Animation starts at the instruction pointed to by the active program counter.

2.9.1.1.9. Instruction Break Enable Button.

This button	
	Instruction Break Enabled
	Instruction Break Disabled

2.9.1.1.10. Data Break Enable Button

This button globally enables or disables the break condition on memory access. When this function is enabled each memory access is evaluated for a break condition (for a description of data breakpoint definition see the section « Watch window »). The emulator breaks AFTER execution of the instruction which performed the memory access. If this button is disabled then no data breakpoint evaluation will be performed.

Button State	
	Data Break Enabled
	Data Break Disabled

2.9.1.1.11. Trace Enable Button

This button globally enables or disables the trace mode.

<u>Note</u> : only after enabling the trace mode for the first time (after starting a debugging session) the Trace-Window will be opened and initialized ; subsequent disablings will not close this window, but will only disable instruction tracing

Button State	
	Trace Enabled
	Trace Disabled

2.9.1.1.12. Step Interrupt Enable Button.

This button enables or disables the interruption during a STEP of execution. This enables the user to execute the main procedure of an application program when the emulator is still receiving interrupt signal.

Button State	
	Interruption enabled during a STEP
	Interruption disabled during a STEP

2.9.1.1.13. I D T Break Status

These markers indicates the current break condition. « I » means an instruction break is active, « D » means a data break is active and « T » means a trace break is active.

I	Instruction Break Flag Break
D	Data Break Condition
T	Trace Break Condition

2.9.1.1.14. Run status

This symbol indicates when the emulator is executing an application program (RUN) or is stopped (STOP).

2.9.1.2. Microcontroller state Panel

Fig. 13 State of the microcontroller core

2.9.1.2.1. PC selection and value

When the Emulator has suspended its execution, the value of the current PC is shown in red. The user can select the current PC (Program Counter) thanks to the Option button and modify the value of the PC directly in the text box. The PC's can take value from 0 to FFF

2.9.1.2.2. Accumulator value

When the Emulator has suspended its execution, the current value of the Accu is shown. The user can modify the value of the Accumulator. The accumulator can take value from 0 to F.

2.9.1.2.3. microcontroller state flags

When the Emulator has suspended its execution, the user can modify the state of the Z, Cy and Cy_int flags.

2.9.1.2.4. Index value

When the Emulator has suspended its execution, the user can modify the value of the index registers. The index can take value from 0 to 7F. The two index registers are automatically updated from the values entered
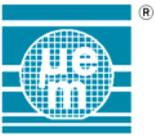
2.9.1.3.Source Code Buffer

The source code buffer contains the source code of the application being executed in the emulator. All data concerning the program is shown in a table form as shown below. It is also possible from this window to activate and deactivate instruction breakpoints as well as defining code sections to be traced.

2.9.1.3.1.Source code data

The data concerning the program being emulated is shown in the following columns

Column	Contents
B	Enable an instruction break at this line
T	Trace this instruction
Line	Line number in the list file (*.prn)
Addr	The ROM address
Code	The 16 bit machine code generated by the assembler
Label	Any labels defined in the source code
Instr	The instruction
Operand	The eventual instruction operands
Remarks	Any eventual comments associated with the instruction

2.9.1.3.2.Activating / Deactivating Instruction Breakpoints

Instruction breakpoints can be set by simply clicking in the column « B » on the line where the breakpoint is required. When a breakpoint is set it is indicated by a cross in the « B » column as shown below.
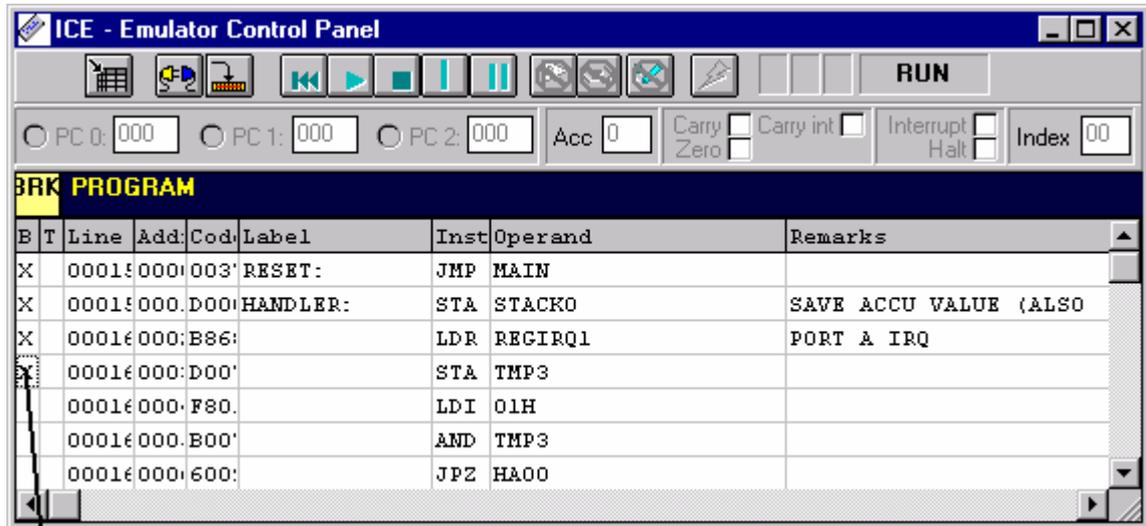
Fig. 14 Setting instruction breakpoints in the emulator control panel

To remove an instruction break point simply click in the « B » column to toggle the instruction break function. It should be noted that for instruction breakpoints to be active they should be globally enabled using the « Instruction Break Enable Button » in the Execution toolbar.

In order to clear all breakpoints at once, just click the mouse on the « B » (on the title line).

When breakpoints are set with the emulator stopped, they are directly written to the in-circuit emulator. However, if the breakpoint is set while the emulator is running it is necessary to perform a down load of the program before the change in the breakpoints will be implemented.

2.9.1.3.3.Defining Code to be Traced

Any program instruction can be traced by clicking in the « T » column on the corresponding instruction line. A traced instruction is indicated by an « x » in the « T » column as shown below.
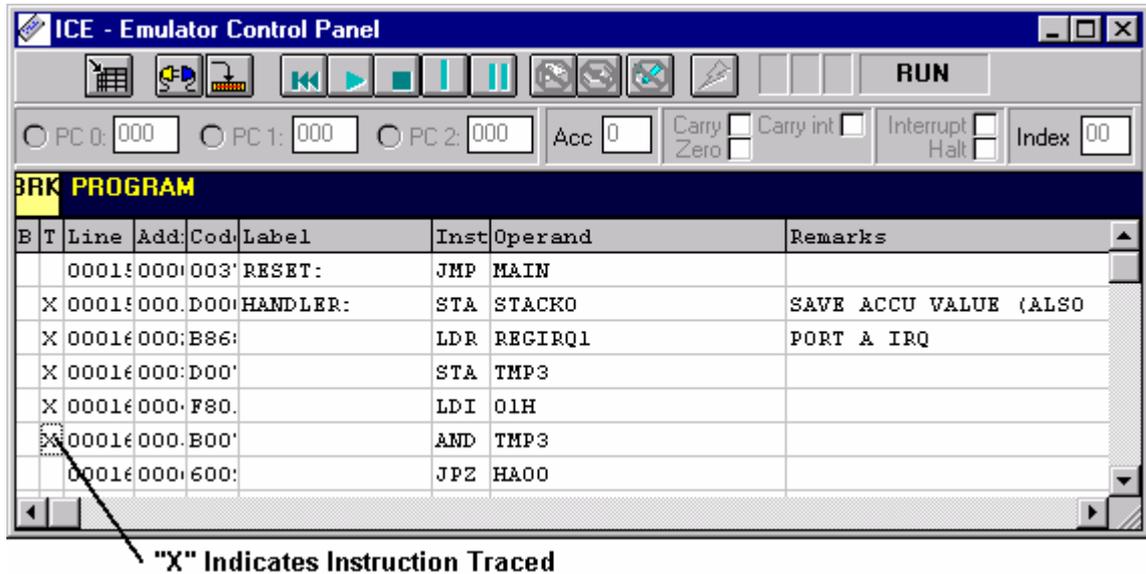
Fig. 15 Selecting instructions to be traced in the emulator control panel

You can activate many adjacent trace lines at once by dragging the mouse (keeping the left button down, while moving the mouse) along the trace column, from a start line to an end line and then releasing the left button ; the whole selected block will be marked with « x ».

You can also activate all the lines of a program at once, just be depressing the « T » (on the title line)

To remove a trace, depress the Ctrl-Key first and, while keeping it down, click on the « x » you want to eliminate. Analog to this, you can remove many (or all) trace marks at once by first depressing the Ctrl-Key and, while keeping it down, clicking or dragging the mouse, as explained before for activating traces.

When traces are set with the emulator stopped the trace point is directly written to the in-circuit emulator. However, if the trace point is set while the emulator is running it is necessary to perform a down load of the program before the change in the trace will be implemented.

2.9.2. Emulator RAM Window

The **RAM Window** is opened together with the Control Window and is a mirror of the complete RAM/Reg Map of the emulated/simulated controller, ordered by address.

Fig. 16 The RAM-Window. Editing a variable's value

To edit a variable's value, click the mouse on the corresponding cell and an edit box will open. To confirm the changes made you have to press ENTER. If you decide not to confirm the change, press ESCAPE.

Note : any changes made on the RAM Window are immediately reflected on the Watch Window and vice-versa .

2.9.3. Emulator Watch Window

The Emulator Watch Window contains all the symbol definitions of the application currently loaded in the emulator control window source code buffer. It is automatically updated each time a new binary file is loaded into the emulator control window. The emulator watch window, as shown below is composed of two tables.
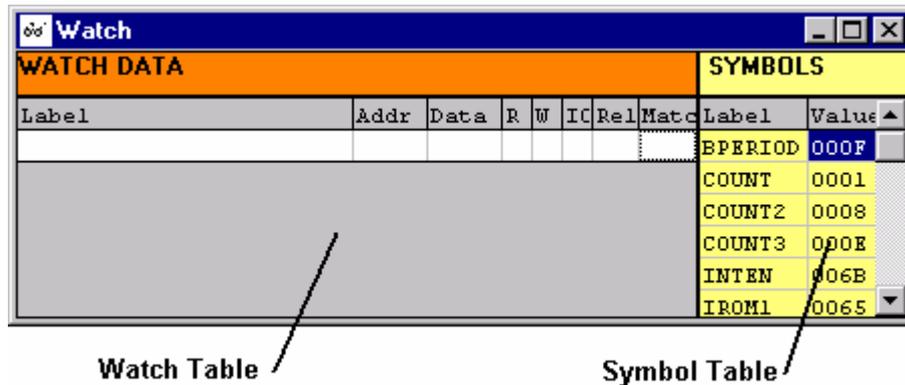
Fig. 17 The data watch panel

The table on the right is the symbol table and is a list of all the RAM/Reg symbols defined in the application. Next to the symbol definition is the value of the symbol. The table on the left is the Watch data table. This table contains all the variables which are currently being watched. With the VICE the watch data is updated after each instruction, however, with the ICE the watch data is only updated once the emulator enters stop mode due to an emulator break or a user requested stop.

2.9.3.1.The Watch Table

The watch table is composed of the following components

Column	Contents
Label	The label being watched
Addr	The address of the watched label
Data	The data value of the variable at the last emulation break
R	Enables the relation testing during the read of the address
W	Enables the relation testing during the write of the address
Rel	Defines the relation used in the relationship test
Match	Defines the reference value used in the relation test

2.9.3.2.Adding or deleting Variables in the Watch Table

A variable can be added to the watch table by clicking on the variable in the Symbol table or the source code buffer in the emulator control window. Then click on the column header « Label » in the watch window. The variable is then added at the bottom of the watch table.

To delete a variable from the watch table select the label in the watch table and press the « Del » key.

2.9.3.3.Modifying Variables in the Watch Table

It is possible to change the value of any variable being watched ; to do this, click the mouse on the value to be edited and an edit box will then open to enable you to make changes. Once you have completed edition, you have to depress ENTER, so the system will validate the changes and close the edit box. If you depress ESCAPE, the edit box will close without validating any changes.

It is also possible to introduce a new symbol (only valid during debugging) : this is accomplished by clicking on the address field of the empty line and typing in the address of the new watch symbol ; the name will be automatically generated.
This new symbol won't be reflected in the RAM-window, which will always reflect the real source information.

<u>Note</u> :
This option is only available when the emulator is in Break mode, consequently the emulator must be stopped before any data modification is performed.
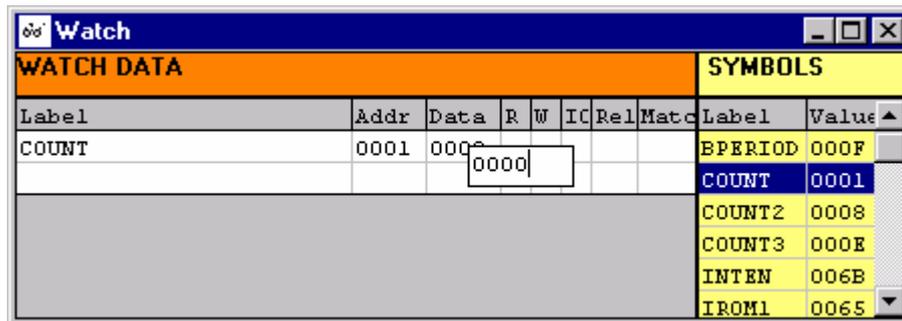
Fig. 18 Changing the value of a watched variable

2.9.3.4.Setting Data Breakpoints.

It is possible to define breakpoints when a read and / or write of a particular address occurs. This is enabled by selecting the « R » option for read and the « W » option for write in the watch table. It is also possible to add a logical relationship to this data breakpoint (for example break only when 0F is written to the address location). To define a relation select the « Rel » or « Match » option and enter the relationship. The following relationships are possible.

Relationship	Signification
<>= MATCH	Always true (break)
< MATCH	Break when variable is less than MATCH
<= MATCH	Break when variable is less than or equal to MATCH
> MATCH	Break when variable is greater than MATCH
>= MATCH	Break when variable is greater than or equal to MATCH
<>MATCH	Break when variable is NOT equal to MATCH
=MATCH	Break when variable is equal to MATCH

Data breakpoint evaluation is only active if the data breakpoint enable button is selected in the « Execution Toolbar» of the Emulator control window.

2.9.4. Emulator Trace Window

Once the trace enable button in the emulator control window is selected the emulator trace window (shown below) is loaded.
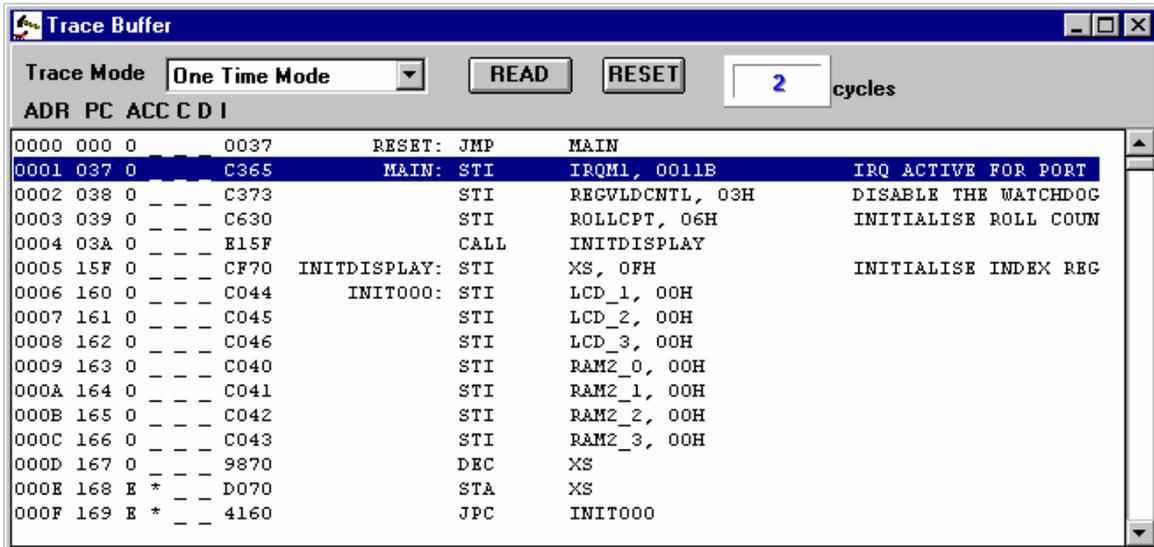
Fig. 19 The trace buffer window

The trace window is made up of the following components.

Reset Button

This button reset the trace memory.

Read Button

Uploads the trace buffer contents from the in-circuit emulator

Note : in fact the Read Button is not needed anymore, because actually, if tracing is active, the trace buffer will be automatically updated as soon as the emulator stops.

Address value

This value is the address of the trace memory.

Trace Mode selection

This option list enables the user to chose between the two trace modes. The first mode fills the trace memory and stops, generating a trace break. This enables to record the first instruction of an execution. In the second mode, when the trace memory is full, the Emulator resets the trace address and refills the memory again ;

no trace break is generated. This option enables the last executed instruction to be stored.

Trace memory format

The trace memory has the following format :

PC	Value of the Program Counter
ACC	Value of the Accumulator
C	Value of the Carry bit
D	Value of the Data break
I	Value of the Instruction Break

To make the trace buffer more readable the source code statements are added to the trace buffer contents. The data shown are the labels the instructions and the comments.

Printing the trace buffer contents. It is possible to print the trace buffer contents to the default system printer by selecting the option « Print » from the file menu while the trace buffer window is active. It is possible to select the trace buffer address to print by entering the start address and end address in the trace buffer print dialogue box shown below.

Print Trace Buffer

Start Address (Hex): 00H
End Address (Hex) 0FH

Print   Cancel
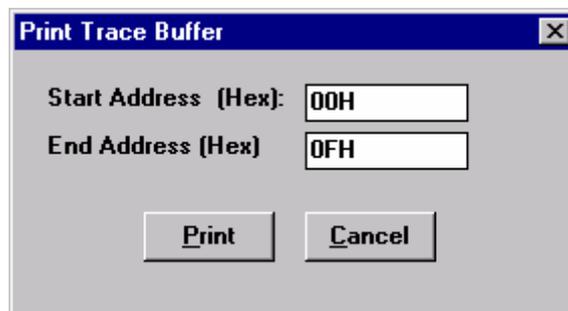
Fig. 20 Trace buffer print dialogue box

The trace buffer is printed using the font defined for the trace buffer window

2.9.5. VICE IO Window

The IO Window contains a graphical representation of the periphery of the simulated controller ; it is therefore only available in VICE Mode and its exact form is controller dependant. Nevertheless, the principle remains the same, so that we will explain it's use taking the 6503 example
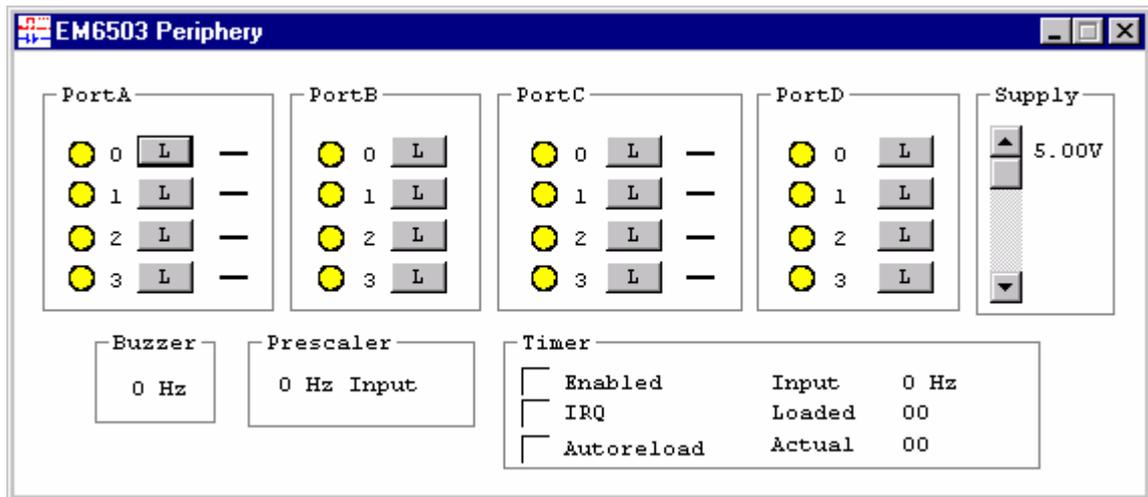
Fig. 21 The I/O Window

The state of each port is shown by the circular LED. When the port is high, the LED is red and when the port is low the LED is yellow. For input ports, the state of the port can be set by using the push buttons. By putting the switch to « H » the port is set to high and by putting the switch to « L » the port is set to low. For the other peripheral modules the current state is indicated ; some of them allow interaction or input from the user (SVLD, for instance).

Note : details concerning the I/O Window for different controllers are given in the **Peripheral Interface Modules** Manual, which is included in this binder.

2.9.6. In-circuit Emulator Communication Configuration

The software development system is connected to the in-circuit emulator is by a serial port. The serial communication parameters are set using the « Ice Config » option in the « emulator » menu. Selection of this option activates the communication dialogue box shown below.
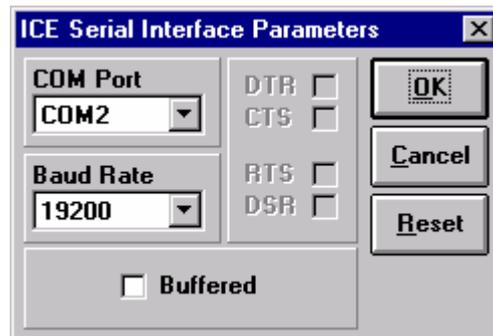
Fig. 22 In-circuit emulator communication parameters

2.9.6.1.COM Port

Selects the serial port used for the communication with the in-circuit emulator. The default setting is COM1

2.9.6.2.Baud Rate

Selects the baud rate of the serial communication. The possible options are 2400, 4800, 9600, 14400 and 19200 baud. However, actually the in-circuit emulator supports only 9600 an 19200 baud. The default setting is 19200 baud

2.9.6.3.DTR CTS RTS DSR

Shows the actual state of the communication control lines

2.9.6.4. Buffered

Selecting the buffered option enables the buffering of the serial communication by the development system software. This is recommended for normal use or if communication problems with the in-circuit emulator are encountered.

2.9.6.5.OK button

Validates the selected communication parameters. The old communication port is closed and the selected port opened with the selected communication parameters.

2.9.6.6.Cancel Button

Cancels any modifications and exits. The communication port remains active with the existing parameters

2.9.6.7.Reset Button

Resets the communication. The communication state is reset and the serial port is closed and reopened

2.9.7. MTP Interface

If you have an MTP-Programmer connected correctly to your computer, then this interface will allow you to program any MTP-controller (EM65XX Series) directly from your development system. In order to open this window, choose the « MTP » under the « Emulation » menu.
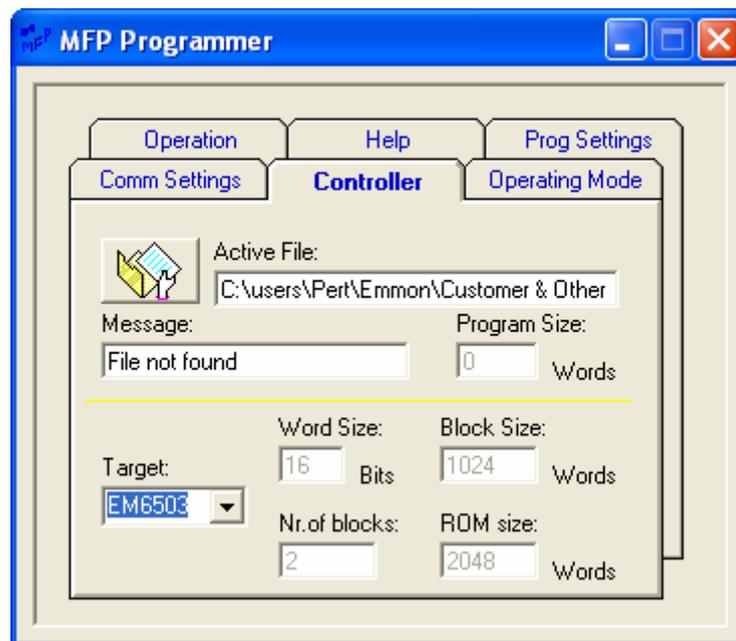
Fig. 23 The MTP-Window.

<u>**Note**</u> : for detailed information on this topic, see the **MTP manual**, which is also part of this binder.

3. EM66xx Programming Model

This section provides an overview of the EM66xx 4-bit microcontroller family and the corresponding programming model. For a more detailed description of each controller please refer to the corresponding specifications.

3.1. Global Architecture

The general structure of the EM66xx family of microcontrollers is shown below.

Fig. 24 General structure of EM66xx 4bit microcontrollers

At the heart of the EM EM66xx microcontroller family is a 4-bit arithmetic and logic unit (ALU), which is capable of executing 32 different instructions. The ROM memory used has a maximum capacity of 4K 16-bit instructions. The data consists of 126 4 bit words which is distributed between the RAM and peripheral registers. The entire address space can be accessed in an indexed manner using the register indexes.

The controllers have three program counters (PC) which means 2 possible program levels when interrupts are enabled, or three program levels if the interrupts are disabled.

3.1.1. Instruction Register

This register stores the next instruction that will be executed.

3.1.2. Control Unit

This functional block generates the internal clock phases of the microcontroller, controls the halt mode, the interrupt requests, decodes the instructions and sends the control signals to the other blocks in microcontroller.

3.1.3. Program Counter (PC)

The program counter determines the ROM address of the next instruction to be executed. The processor has a stack of three program counters (PC0 to PC2). The current program counter is designated by the stack pointer (SP).

3.1.4. Stack Pointer (SP)

The stack pointer determines the current program counter (PC). At reset the SP is reset to PC0.

3.1.5. Accumulator (Accu)

The accumulator stores the 4 bit result of the ALU operations.

3.1.6. Status Register

The status register is composed of three flags. The Z flag which is set to 1 if the result of the last ALU operation equals zero. The Cy flag and the Cy_Int flag which receive any addition, subtraction or shift carries. The Cy flag is used during normal program execution and the Cy_Int flag is automatically selected during interrupt handling thus conserving the state of the Cy flag during interrupt.

3.1.7. Index Registers

The index registers XH (index high, 3 bits) and XL (index low, 4 bits) are combined to define a 7 bit offset in the IO (RAM / periphery) address space.

3.1.8. Interrupts

An interrupt can be generated by one or more peripherals. These signals are combined with a logical OR gate to produce a single interrupt request to the microcontroller core. The resulting interrupt request results in the microcontroller executing a CALL 01 instruction and setting the INT flag. The RTI instruction signals the end of the interrupt routine. It executes a return to the instruction which should have been executed at the moment of the interruption (as opposed to a RET
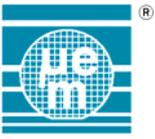
instruction which returns to the address following the a CALL instruction) and resets the INT flag.

If interrupts are enabled then the programmer should be careful to only use two program counters in the main body of the program, since one should be reserved for the interrupt handler call. The interrupt handler routine must be placed at the address 001H and should be terminated with a RTI instruction. On entering the interrupt handler the Cy_int flag is automatically selected. However, the value of the Accu should be saved to a temporary RAM location. The value of Accu should also be restored prior to the execution of the RTI instruction. For an overview of the advised program structure please see fig. 2.

The EM66xx micro controllers only have one interrupt handler and cannot perform nested interrupts. If a second interrupt signal arrives when the controller is already interrupted then the signal is not processed until the controller has finished processing the first interrupt. Immediately after the RTI instruction the controller will renter the interrupt handler without executing an instruction in the main body of the program.

3.2. Instruction Set

The following abbreviations are used in the instruction set description:

Addr : ROM address.
PC : Program Counter
SP : Stack Pointer
Reg : Either a RAM or peripheral register address
IO : The RAM, peripheral register address space
IX : The index register (XH and XL combined)
Dat : A 4-bit datum
Accu : Accumulator

The EM66xx instruction set can be classified according to the following categories:

3.2.1. General and Program Flow Control Instructions

Assembler Syntax	Function	State Z	State Cy
JMP Addr	PC(SP) = Addr	-	-
JPV1 Addr	*if TESTVAR1 = 1 PC(SP) = Addr	-	-
JPV2 Addr	*if TESTVAR 2 = 1 PC(SP) = Addr	-	-
JPV3 Addr	*if TESTVAR 3 = 1 PC(SP) = Addr	-	-
JPC Addr	if Cy(Cy_Int) = 1 PC(SP) = Addr	-	-
JPNC Addr	if Cy(Cy_Int) = 0 PC(SP) = Addr	-	-
JPZ Addr	if Z = 1 PC(SP) = Addr	-	-
JPNZ Addr	if Z = 0 PC(SP) = Addr	-	-
CALL Addr	SP = SP + 1; PC(SP) = Addr	-	-
RET	SP = SP - 1; PC(SP) = PC(SP) + 1	-	-
RTI	SP = SP - 1	-	-
HALT	PC(SP) = PC(SP) + 1; standby mode	-	-
NOP	no instruction	-	-

* TESTVAR1, TESTVAR2 and TESTVAR3 are internal signals which, depending on the target controller, may be connected to different sources. Normally the signals are connected to input port A. The JPV1, JPV2 and JPV3 instructions allow a conditional jump to be directly executed on the state of the signals. For the assignment of the TESTVAR signals please refer to the specifications of the target micro controller.

3.2.2. Register (RAM) Load and Store Operations

(all instructions imply PC(SP) = PC(SP) + 1)

3.2.2.1. Direct load and store operations

Assembler Syntax	Function	State Z	State Cy
STI Reg, Dat	IO(Reg) = Dat	-	-
STA Reg	IO(Reg) = Accu	-	-
LDI Dat	Accu = Dat	z	0
LDR Reg	Accu = IO(Reg)	z	0

3.2.2.2. Indexed load and store operations

Assembler Syntax	Function	State Z	State Cy
STIX Dat	IO(IX) = Dat	-	-
STAX	IO(IX) = Accu	-	-
LDRX	Accu = IO(IX)	z	0
LDRXS	Accu=shr[IO(IX)]	z	see fig. 26

3.2.3. Binary Operations

3.2.3.1.Shift left

The principal of the shift left instruction is the following:
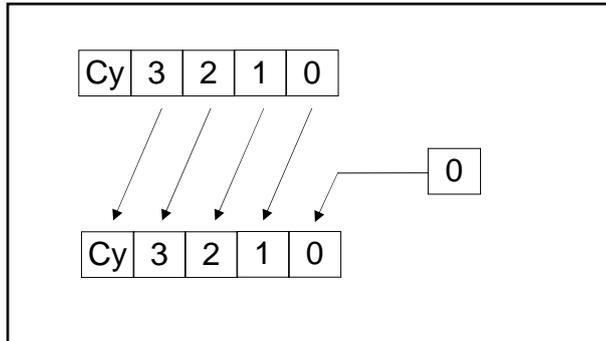
Fig. 25 Shift left operation

Assembler Syntax	Function	State Z	State Cy
SHL *(or* SHLR♦ *)* Reg	Accu = IO(Reg)*2	z	see fig. 25
SHLX	Accu = IO(IX) * 2	z	see fig. 25

♦ now there are 2 definitions for the same instruction. SHL is the one actually adviced for new projects, because it is more consistent with the « shift right » instruction group. The former version (SHLR) will only be supported for compatibility purposes with existing programs.

3.2.3.2.Shift right

The principal of the right shift operation is the following :
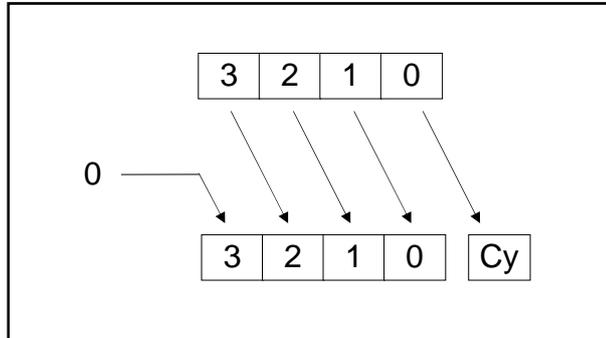
Fig. 26 Shift right operation

Assembler Syntax	Function	State Z	State Cy
SHRA	Accu = shr(Accu)	z	see fig. 26
SHRR Reg	Accu = shr(Accu = IO(Reg))	z	see fig. 26

3.2.3.3.Direct binary operations

Assembler Syntax	Function	State Z	State Cy
AND Reg	Accu = IO(Reg) AND Accu	z	0
NAND Reg	Accu = IO(Reg) NAND Accu	z	0
OR Reg	Accu = IO(Reg) OR Accu	z	0
NOR Reg	Accu = IO(Reg) NOR Accu	z	0
XOR Reg	Accu = IO(Reg) XOR Accu	z	0
NXOR Reg	Accu = IO(Reg) NXOR Accu	z	0
CPLR Reg	Accu = NOT(IO(Reg))	z	0
CPLA	Accu = NOT(Accu)	z	0

3.2.3.4.Direct binary operations with shift right

Assembler Syntax	Function	State Z	State Cy
ANDS Reg	Accu = SHR[IO(Reg) AND Accu]	z	see fig. 26
NANDS Reg	Accu = SHR[IO(Reg) NAND Accu]	z	see fig. 26
ORS Reg	Accu = SHR[IO(Reg) OR Accu]	z	see fig. 26
NORS Reg	Accu = SHR[IO(Reg) NOR Accu]	z	see fig. 26
XORS Reg	Accu = SHR[IO(Reg) XOR Accu]	z	see fig. 26
NXORS Reg	Accu = SHR[IO(Reg) NXOR Accu]	z	see fig. 26
CPLRS Reg	Accu = SHR[NOT(IO(Reg))]	z	see fig. 26
CPLAS	Accu = SHR[NOT(Accu)]	z	see fig. 26

3.2.3.5.Indexed binary operations

Assembler	Function	State Z	State Cy
ANDX	Accu = IO(IX) AND Accu	z	0
NANDX	Accu = IO(IX) NAND Accu	z	0
ORX	Accu = IO(IX) OR Accu	z	0
NORX	Accu = IO(IX) NOR Accu	z	0
XORX	Accu = IO(IX) XOR Accu	z	0
NXORX	Accu = IO(IX) NXOR Accu	z	0
CPLRX	Accu = NOT(IO(IX))	z	0

3.2.3.6.Indexed binary operations with shift right

Assembler	Function	State Z	State Cy
ANDXS	Accu = SHR[IO(IX) AND Accu]	z	see fig. 26
NANDXS	Accu = SHR[IO(IX) NAND Accu]	z	see fig. 26
ORXS	Accu = SHR[IO(IX) OR Accu]	z	see fig. 26
NORXS	Accu = SHR[IO(IX) NOR Accu]	z	see fig. 26
XORXS	Accu = SHR[IO(IX) XOR Accu]	z	see fig. 26
NXORXS	Accu = SHR[IO(IX) NXOR Accu]	z	see fig. 26
CPLRXS	Accu = SHR[NOT(IO(IX))]	z	see fig. 26

3.2.4. Arithmetic Operations

The basic arithmetic operations (INC, DEC, ADD and SUB) can be direct, indexed, direct with shift right or indexed with shift right. When the instruction is combined with a shift right it is performed according to the following principal.
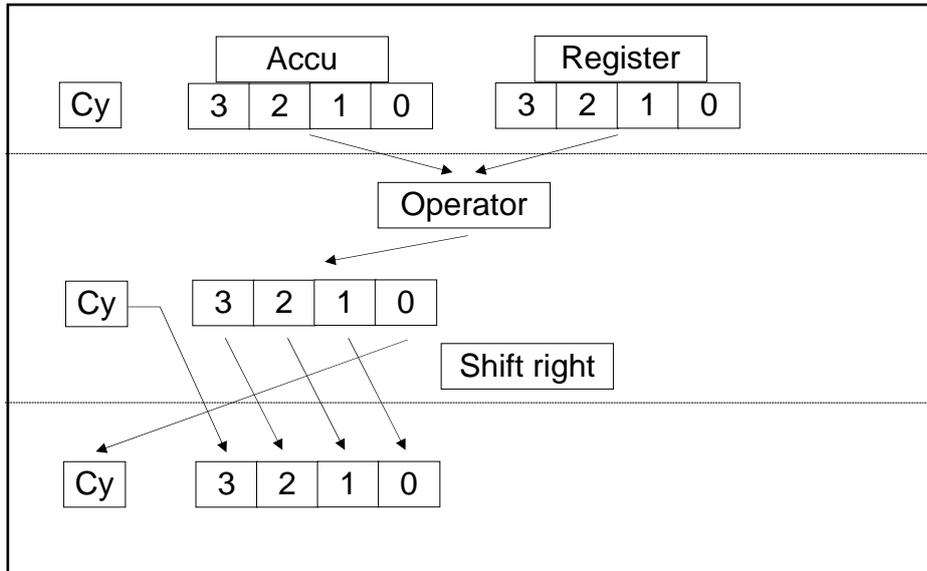
Fig. 27 Principal of arithmetic operations followed by shift right

3.2.4.1. Direct arithmetic operations

Assembler Syntax	Function	State Z	State Cy
ADD Reg	Accu = IO(Reg) + Accu	z	if (Accu >15) c = 1
INC Reg	Accu = IO(Reg) + 1	z	if (Accu >15) c = 1
SUB Reg	Accu = IO(Reg) - Accu	z	if(Accu>=0) c=1
DEC Reg	Accu = IO(Reg) - 1	z	if(Accu>=0) c=1

3.2.4.2.Direct arithmetic operations with shift right

Assembler Syntax		Function	State Z	State Cy
ADDS	Reg	Accu = SHR[IO(Reg) + Accu]	z	see fig. 27
INCS	Reg	Accu = SHR[IO(Reg) + 1]	z	see fig. 27
SUBS	Reg	Accu = SHR[IO(Reg) - Accu]	z	see fig. 27
DECS	Reg	Accu = SHR[IO(Reg) - 1]	z	see fig. 27

3.2.4.3.Indexed arithmetic operations

Assembler	Function	State Z	State Cy
ADDX	Accu = IO(IX) + Accu	z	if (Accu >15) c = 1
INCX	Accu = IO(IX) + 1	z	if (Accu >15) c = 1
SUBX	Accu = IO(IX) - Accu	z	if(Accu>=0) c=1
DECX	Accu = IO(IX) - 1	z	if(Accu>=0) c=1

3.2.4.4.Indexed arithmetic operations with shift right

Assembler	Function	State Z	State Cy
ADDXS	Accu = SHR[IO(IX) + Accu]	z	see fig. 27
INCXS	Accu = SHR[IO(IX) + 1]	z	see fig. 27
SUBXS	Accu = SHR[IO(IX) - Accu]	z	see fig. 27
DECXS	Accu = SHR[IO(IX) - 1]	z	see fig. 27

3.2.5. Binary representation of Instruction set

The following section describes the binary coding of the EM66xx instruction set.

Instruction	Binary representation
JMP <Adr>	`0000` Adr<12>
JPV1 <Adr>	`0001` Adr<12>
JPV2 <Adr>	`0010` Adr<12>
JPV3 <Adr>	`0011` Adr<12>
JPC <Adr>	`0100` Adr<12>
JPNC <Adr>	`0101` Adr<12>
JPZ <Adr>	`0110` Adr<12>
JPNZ <Adr>	`0111` Adr<12>
CALL <Adr>	`1110` Adr<12>
RET	`1111` `1010` ⊠
RTI	`1111` `0010` ⊠
NOP	`1111` `0000` ⊠
HALT	`1111` `0100` ⊠
STI <R>,<D>	`1100` D<4> `0` R<7>
STIX <D>	`1100` D<4> `1` ⊠
STA <R>	`1101` ⊠ `0` R<7>
STAX	`1101` ⊠ `1` ⊠
LDI <D>	`1111` `1000` ⊠ D<4>
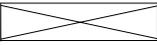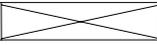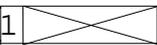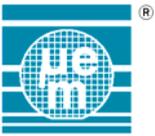
The rest of the instruction set can be represented in the following way.

ACCU = ACCU op RAM()

| 10 | op<5> | S | I | RAM(i)<7> |

If I = 1 the instruction is indexed and if S = 1 a shift right is performed after the operation.

The coding of the ALU operations (op<5>) is as follows.

Operation	Op code
ADD	01001
SUB	00110
INC	00011
DEC	01100
SHL	01111
AND	11000
NAND	10111
OR	11110
NOR	10001
XOR	10110
NXOR	11001
CPLA	10101
CPLR	10011
ACCU = RAM (LDR <R>)	11100
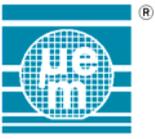
4. EM66xx Assembler Syntax

4.1. Introduction

This section describes in detail the syntax to be followed in assembly programs to enable EM66xx assembler to assemble correctly. A full description of the directives, variable and constant definitions, macros and pseudo-op statements supported by EM66xx assembler is also provided. The subjects covered in this section include:

General Assembly statement rules

How to include general 'documenting text' within programs

Symbol writing and Syntax

Numeric Expression Syntax

Assembler "Pseudo-op" Directives

A later section explains how to write assembly language programs, and in particular, how EM66xx assembler features can be utilised.

4.2. Instruction Syntax Basics

CPU instructions are written using mnemonics (acronyms) that represent the required operation. EM66xx assembler is configured for the correct mnemonics set required by the EM66xx microcontroller family as described in the EM66xx Instruction Set.

A common feature of all CPU/MPU mnemonics is the division into two basic syntax elements of:

OPERATOR   OPERAND(,OPERAND(,OPERAND))

The OPERATOR specifies in general terms the action that the instruction will perform, while the OPERANDS specify what the instruction will act upon (unless it is implied by the OPERATOR in which case no OPERANDS are necessary).

When an OPERAND is entered it must be separated from the preceding OPERATOR by SPACE/TAB character(s). Example instruction mnemonics of the EM66xx CPU are illustrated below:

HALT   ;Enter HALT mode

SUB   00FH   ;Subtract Accu from memory address

LDR   O1FH   ;Load Accu from memory address O1FH

CALL   TEST   ;Execute TEST subroutine

Within the OPERANDS some instructions may require a numeric value to be specified (typically a memory address or data value). You can write these values either as SYMBOLS, CONSTANTS or EXPRESSIONS which are evaluated by the assembler. Leading labels are automatically assigned with the current program location for the instruction which is referenced.

4.3. General Statement Rules

Assembly language statements comprise single text lines held in a text file, which are usually created within a word processor (though the word processor used must create 'standard' text files). Each statement encodes either a CPU/MPU instruction or assembler "Pseudo-op" directive.

```
-----------------------------------------------------------------------
(LABEL)    INSTRUCTION     (OPERANDS)      (COMMENT)
-----------------------------------------------------------------------
Name1      LDI        Reg, 10   ;This is an example
   JMP     Name1
   STAX                ;No OPERAND required


-----------------------------------------------------------------------
(LABEL)    DIRECTIVE       (OPERANDS)      (COMMENT)
-----------------------------------------------------------------------
Cr         EQU        0DH        ;Assemble Directives


-----------------------------------------------------------------------
(LABEL)                    (COMMENT)
-----------------------------------------------------------------------
Symbol:                    ;PC assigned to Symbol

;This is a simple comment for increased program readability
-----------------------------------------------------------------------
```

Fig. 28 EM66xx general statement rules

The text line is considered as having several 'fields' as follows:

Elements enclosed in parentheses are optional and determined by the program or the particular instruction. Upper/lower case characters are not distinguished unless enclosed within single quote marks, and blank text lines are ignored.
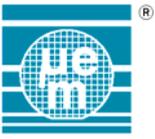
Elements must be separated by one or more TAB (ASCII 9) or SPACE characters. Comments must begin with a semi-colon (;). Only labels are written starting in column 1 of the text line.

4.4. Embedded Documentation

Embedded COMMENTS can be included in any statement and are taken to be all characters to the right of a semi-colon (;). This is standard practice of most assemblers and fully supported by EM66xx assembler.

;Embedded single-line Comment (accepted by most Assemblers)

4.5. Symbol Syntax Rules

Only labels are written starting in column 1 of the text line (or comments as described above), and when assembled are allocated the memory address of the instruction. These can then be used within an EXPRESSION to automatically reference the 'tagged' instruction address. Assembler "pseudo-op" directives may also require a LABEL as part of the syntax. Syntax rules for labels are:

Unlimited length.

All characters are significant.

May include only uppercase and lowercase letters, digits and the underscore character (A .. Z, a .. z, 0 .. 9, _).

Cannot begin with digits (0 .. 9).

May be terminated with a colon (:).

Examples of acceptable and non-valid symbols are given below:

ROOX0   ;Valid Label Examples

LOOP1

Mem_Setting

RESET: ;':' allowed if last character

0POS   ;illegal - begins with 0

LP*1 ;illegal - contains expression character

RESET:1   ;illegal - contains ':'

Stipulation of LABEL syntax rules is to avoid incorrect evaluation or misinterpretation. For instance, '0POS' would be assumed to be a numeric constant as it begins with 0, consequently an eventual 'Syntax Error' would be generated when it is referenced by an instruction.
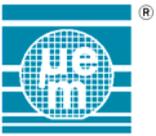
Some assemblers allow labels to be written with a colon (:) appended. This is optional for EM66xx assembler (the colon is ignored). Note that the term SYMBOL applies to any character string which is assigned, symbolises or represents a numeric value (whereas LABEL is used where a symbol is assigned the address of an instruction).

4.6.   Constant Syntax

Constants are direct numeric values written in the operand as (or part of) an EXPRESSION. Four numeric bases are supported with a variety of 'radix' options to denote the number base.

All constants are written with the least significant numeral of the number on the right.

Hex: 0FF37H $FF37

Binary: 1001B   %1001

Octal: 637Q   637o

Decimal: 127, -32000, 65000, 100D

String:'A','AB' (equivalent to: 41H,4142H)

4.7. Expression Syntax

Expressions are a combination of numeric constants, labels/symbols, and expression operators, which are evaluated by the assembler into a single value. Expression operators supported are listed below.

All expressions are evaluated using 32-bit calculations.

+	Addition.
-	Subtraction
*	Multiplication.
\	Division (only Integer part).
/	Division (rounded integer result)
^	Potentiation
MOD	Remainder of division.
AND	Logical AND operator.
OR	Logical OR operator.
XOR	Logical XOR operator.
SHR	Logical SHIFT RIGHT (shifting 0's in).
SHL	Logical SHIFT LEFT (shifting 0's in).

Compound operators using the value to the right include:

NOT n	Returns the 'inverted bit-state of n'.
HIGH n	Returns the 'upper byte (high 8-bit) of n'
LOW n	Returns the 'lower byte (low 8-bit) of n'
OFFSET n	Returns the 'lower word (low 16-bit) of n'
UPPER n	Returns the 'upper word (high 16-bit) of n'
SWAP n	Exchanges the upper/lower bytes (low 16-bit)

Comparisons operate on signed values. True returns 0FFFFFFFFH,

False returns 0.

= or EQ	True if equal.
<> or NE	True if not equal.
<= or LE	True if less than or equal (lower or equal).
< or LT	True if less than (lower than).
>= or GE	True if greater than or equal (higher or equal)
< or GT	True if greater than (higher than).

Other special operators:

$ Returns the current PC value.

@   Returns the current VC value.

Expressions are evaluated 'left to right' only, full (nestable) parentheses are allowed to alter the evaluation order.

All expression elements should be delimited by at least one SPACE character.

An expression at its simplest will be a single numeric constant or a label/variable identifier. Examples of typical expressions are illustrated below:

0FFFFH

Label / 16 + (Label mod 3)

4.8. Assembler Directives

4.8.1. ENDM ENDMACRO MACEND

Syntax : *ENDM* or *ENDMACRO* or *MACEND*

End of macro - Terminates the instructions associated with the current macro definition and is the complement to the MACRO directive described above.

4.8.2. EQU

Syntax : *symbol EQU expr*

Equates the expression value with the symbol. Can only occur once for each label symbol.

IOport1   EQU 0F3H   ;IOport1 assigned

4.8.3. INCLUDE

Syntax : *INCLUDE filename*

Includes assembly source statements from a specified source file. This statement can also occur within the INCLUDE assembly files (i.e. nesting is permitted). Example statements are:

INCLUDE   MATHS.ASM   ;Math Subroutines

INCLUDE   IOINIT.ASM   ;I/O Control Subs

INCLUDE   IOBUFF.ASM   ;I/O Buffer Subs

The supplied filename will be searched for within the drive/directory of the main source file of the project, unless a drive/directory path is specified within the filename.

If the specified file is found, the current statement position within the parent file is saved and the source statements of the INCLUDE file are processed. After assembling the INCLUDE file's statements, the parent file becomes active again and execution continues from the statement following the INCLUDE directive.

Typically, this directive reads source files as shown in the above example, where each file contains general purpose subroutines or a common set of MACRO or variable definitions.
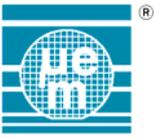
4.8.4. MACRO

Syntax : *label MACRO (par (,par(..)*

Define macro - A full description of macros and the relevant processing is described in the following MACRO DEFINITION section.

A label must be written preceding the MACRO directive, which will become the 'calling name' for the list of instructions that follow this directive (until the complementary MACEND directive). An example MACRO statement is as follows:

SaveAll MACRO AF, BC, DE, HL'   ;Save Registers

The optional formal parameters are used during macro expansion to be replaced by the call parameters which are passed to the macro when the macro is invoked.

4.8.5. ORG

Syntax : *ORG addr*

Origin - sets the PC (Program Counter) to the specified address values.

There are several effects that this directive may cause. If it is executed before any object-code generating statements, it effectively sets the start address for the object-code; otherwise it specifies the next address for following statements.

ORG 0100H; Set address to 0100 hex

ORG $ + 16   ; Increase address by 16

ORG statements may generate an error if the new PC value has overflowed the memory address space or recovered a previously used memory address. Also, you cannot use 'forward references' within the address expressions. This means it is not possible to refer to a symbol that has not yet been defined.

4.9. Macro Definitions

Macros are significant assembly time facilities, which are widely used to simplify program writing. In essence, a macro represents several assembly language statements, which are assigned to a symbol name. When the symbol name is specified as an instruction (i.e. a 'macro call') it will automatically be replaced with the appropriate list of statements. EM66xx assembler fully supports macro definitions including:

Flexible Parameter Passing.

Local Symbols.

Conditional Assembly.

Using these definition capabilities macros can be written which represent complicated program actions and can adjust the object code according to the 'macro call' operands.

A definition is created by surrounding the required assembly language statements by the MACRO and MACEND directives as follows:

```
MoveAll    .macro    PA, TA, PB, TB
   LDA      PA
   STA      TA
   LDA      PB
   STA      TB
 .macend
```

Fig. 29 Macro definition

All the symbols in a macro are local symbols. Formal parameters defined by the MACRO directive are replaced by the actual parameters when the macro is expanded by a macro call.

```
;Program example

Start      STI      VA, 0                    ;Start of prog
   STI      VB, 1
   MoveAll  VA, WA, VB, WB
   CALL     Test
   MoveAll  WA, VA, WB, VB
```

Fig. 30 Macro implementation

From the above program example, which includes 'macro calls' you, can see how this facility can also greatly improves program readability. However, you should be wary of creating a macro that would best be defined as a subroutine as you will cause the assembly program to be much larger than necessary.

The EM66xx assembler supports very flexible parameter passing. You can specify any number of parameters as the operand of the 'macro call' statement (all separated by commas), and whatever is specified replaces the parameter identifiers specified within the statements of the macro definition.

With this in mind we can illustrate the actual statements which would be assembled for the program example given above:

```
;Program example
Start       STI     VA, 0                   ;Start of prog
  STI     VB, 1
  MoveAll   VA, WA, VB, WB
;Expansion
MoveAll.PA.1           EQU     VA
MoveAll.TA.1           EQU     WA
MoveAll.PB.1           EQU     VA
MoveAll.TB.1           EQU     WB
  LDA     MoveAll.PA.1
  STA     MoveAll.TA.1
  LDA     MoveAll.PB.1
  STA     MoveAll.TB.1
  CALL    Test
  MoveAll   WA, VA, WB, VB
;Expansion
MoveAll.PA.2           EQU     WA
MoveAll.TA.2           EQU     VA
MoveAll.PB.2           EQU     WB
MoveAll.TB.2           EQU     VB
  LDA     MoveAll.PA.2
  STA     MoveAll.TA.2
  LDA     MoveAll.PB.2
  STA     MoveAll.TB.2
```

Fig. 31 Expanded macro implementation

LABELS can also be defined within macros, and remain local to each separate 'macro call'. Expressions, which reference a label, which is not defined within the macro, will cause an error.

Note : there is no support for nested macros !
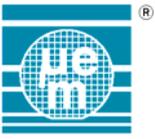
4.10. Assembler Error Messages

Error Number	Signification
100	Unknown instruction.
200	Undefined variable or constant
209	Incorrect use of parenthesis
212	Relocatable Expression not allowed
230	Reserved Word
240	Wrong Nr. Of Parameters
300	Symbol defined twice
400	Invalid numeric constant.
700	Unable to open an a source code or include file.
705	No end macro defined
708	Nr. of .IF and .ELSE doesn't match
799	MACRO without name
902	MACRO nesting not allowed
1204	Macro Name defined twice
1599	Program Memory overflow

5. Mask ROM

The binary file generated by the EM66xx assembler can be sent directly to EM Microelectronic Marin SA for mask ROM generation. The binary file should be accompanied by the selected metal mask options of the specific microcontroller as well as the binary checksum file generated by the last assembly of the project.

The definition of the possible metal mask options can be found in the target microcontroller specification manual.

The binary checksum file is generated each time the source code is assembled. The filename used is dependent on the project name and has the extension *STA*. The file is a text file and has the following format.

```
;-------------------------------------------------------------------------
;        D:\V66xx\V6622\TEST\MSCOUNT\MSCOUNT.STA
;-------------------------------------------------------------------------
;
;
;     ROM Characteristic File
;
;     Company             :
;     Reference           :
;     Software Version    :
;     Contact             :
;
;     Date                :          13/02/1996
;     Time                :          15:08
;
;     Binary file    :          D:\V66xx\V6622\TEST\MSCOUNT\MSCOUNT.bin
;     File size      :          912
;     Modified       :          Tue Feb 13 15:08:06 1996
;
;----------------------------------------------------------------------
;
;     Nb. Instructions   :     456 (0 - 01C7)
;     Check Sum          :          4B8F
;     Reference Controls :
;
; 0004 0002 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
; 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
; 0000 0000 0020 0071 001F 0001 0045 000F 0006 0009 0000 0008 0000 0000 0000 0000
; 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 006E 0000 0016
; 0005 0001 0005 0013 0000 0000 0000 0004
;
;----------------------------------------------------------------------
```

The Fields Company, Reference, Software Version and Contact should be filled in with the appropriate customer project parameters. The fields date and time indicate the date and time of the generation of the checksum file. The fields Binary file, File size and modified refer to the binary file analysed.

The binary file received will be used to regenerate the control fields Nb. Instructions, Check Sum and Reference Controls. These values will be cross-checked against the values sent in the checksum file.

EM66xx Microcontroller

Peripheral Interface modules

CONTENTS

1. General

During software simulation using the virtual emulator the state of the microcontroller periphery is shown by means of the state panel. This panel represents the functional organisation of the microcontroller periphery and allows an interaction with the application program being tested. In this way it is a complement to the WATCH window (see EM66xx Microcontroller Development System Manual) which shows the state of the microcontroller registers irrespective of function. The software simulation interface used is dependent on the target controller chosen during project definition. This document describes the features of each the microcontroller interface modules. All the microcontroller are simulated in accordance with the MFP specification.

2. EM6503 & EM6505

The peripheral interface module for the EM6503 is shown below.
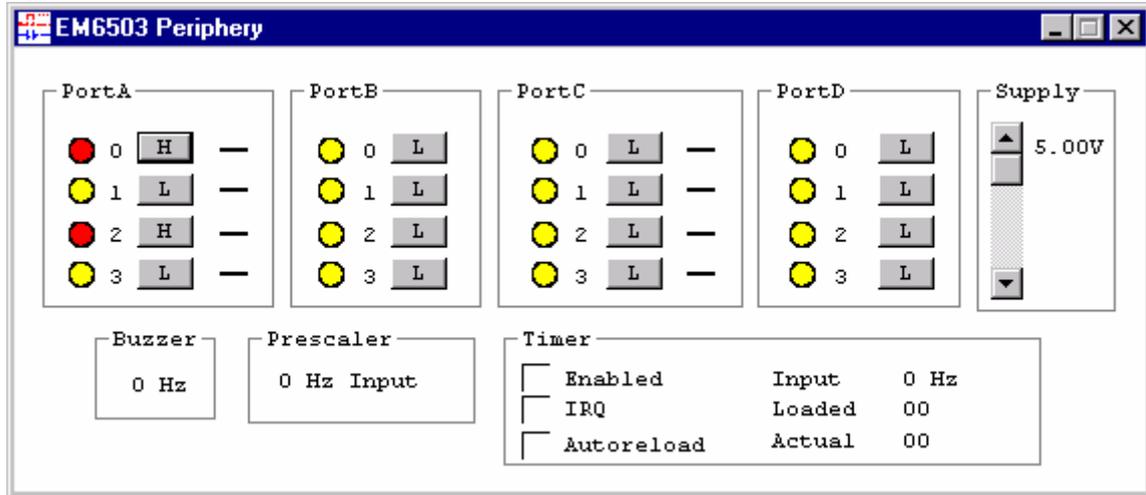
Fig. 2-1 Peripheral interface for the EM6503 microcontroller

2.1. Port A

Port A is an input port with individual maskable interrupts. The circular LED's as well as the push buttons indicates the state of port A (address 072H) and can have values shown in Table 2-1.

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 2-1 Possible input port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 073H) is shown next to the button. The possible interrupt mask states are shown in Table 2-2.

Symbol	State
⌐	Interrupt generated on the falling edge of the port
⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 2-2 Representation of possible IRQ mask states.

2.2. Port B

Port B is a bitwise select input output port. The LED and buttons show (address 075H) the state of port B. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 076H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 2-3.

LED state	Button	State
● Red	H	Port input and high
○ Yellow	L	Port input and low
● Red	NC	Port output and high
○ Yellow	NC	Port output and low

Table 2-3 Possible states for input output port.

2.3. Port C

Port C can be configured as either a 4 bit input port or a 4 bit output port (bit 1 address 07CH). The possible combinations for the LED and button states are the same as for port B and are shown in Table 2-3. Port C has maskable interrupt capabilities (address 079H). The state of the interrupt mask is show next to the port and can have the states shown in Table 2-2.

2.4. Port D / Serial Write Buffer

Port D can configured as either a 4 bit input port or a 4 bit output port (bit 2 address 07CH). When port D is set to output and serial write mode is selected (bit3 address 068H) the serial clock is output on port D0 and the serial data on port D1. The possible combinations for the LED and button states for port D are shown in Table 2-3. When in serial write mode the symbol « NC » is replaced by « SCK » (serial clock) for port D0 and « SD » (serial data) for port D1.

2.5. Prescaler

The selected prescaler interrupt frequency is indicated. This corresponds to the value written to bits 0-1 in the register 07DH.

2.6. Buzzer

Indicates the output frequency selected and corresponds to bits 0-1 in the register 07EH.

2.7. Timer

Fig. 2-2 Timer parameters

The following parameters are defined for the 8-bit timer module.

Parameter	Signification
Enabled	The symbol ☒ signifies the timer is enabled and running. The symbol ☐ signifies the timer is inactive. (Corresponds to bit3 in register 07EH).
IRQ	The symbol ☒ signifies that an IRQ will be generated when the counter counts down to 0 (irq mask bit 3 register 07DH). The symbol ☐ signifies that the IRQ function is inactive.
Autoreload	The symbol ☒ signifies that the autoreload function (bit 3 register 063H) is active and the symbol ☐ that it is inactive.
Input	Specifies the counting frequency used by the counter.
Loaded	The value with which the counter was initialised and which will be used for reloading if the autoreload function is active.
Actual	The actual counter value.

Table 2-4 Timer parameters values

2.8. Supply level detection

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps 0f 0.1V can be obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V will not stop the software simulation of the microcontroller.

3. EM6504

The peripheral interface module for the EM6504 is shown below.
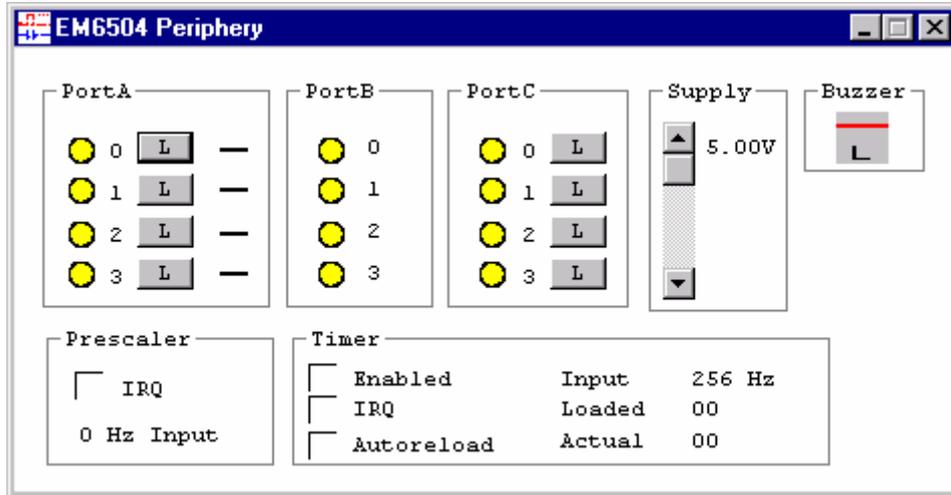
fig 3-1 Peripheral interface for the EM6504 microcontroller

3.1. Port A

Port A is an input port with individual maskable interrupts. The circular LED's as well as the push buttons indicates the state of port A (address 072H) and can have values shown in Table 3-1

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 3-1 Possible input Table port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 073H) is shown next to the button. The possible interrupt mask states are shown in Table 3-2.

Symbol	State
⌐L	Interrupt generated on the falling edge of the port
⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 3-2 Representation of possible IRQ mask states.

3.2. Port B

Port B is an output only port. The LED's indicates the state of the port. A red LED indicates the port is high (logical 1) and a yellow LED indicates the port is low (logical 0). The output directly corresponds to the register 078H.

3.3. Port C

Port C can be configured as either a 4 bit input port or a 4 bit output port (bit 1 address 07AH). The possible combinations for the LED states are the same as for port B and button states are the same as for port A and are shown in Table 3-1.

3.4. Prescaler

Parameter	Signification
IRQ	The symbol ☒ signifies that an IRQ will be generated according to the frequency indicated. The symbol ☐ signifies that no IRQ will be generated even though the IRQ flag will be set . (Corresponds to bit3 in register 07DH).
Input	This field indicates the bas frequency that will be used for the prescaler events (bits 0-1 register 074H).

Table 3-3 Prescaler parameters

3.5. Buzzer

Shows the selected output state of the buzzer (corresponds to bits 0-1 register 07EH). The possible values are shown in Table 3-4.

Value	Signification
2KHz	2048 Hz output
H	Continuous high
L	Continuous low

Table 3-4 Possible buzzer output values

3.6. Timer

```
┌Timer─────────────────────────────┐
│  ☐  Enabled      Input    256 Hz  │
│  ☐  IRQ          Loaded   00      │
│  ☐  Autoreload   Actual   00      │
└───────────────────────────────────┘
```

Fig. 3-2 Timer parameters

The following parameters are defined for the 8 bit timer module.

Parameter	Signification
Enabled	The symbol ☒ signifies the timer is enabled and running. The symbol ☐ signifies the timer is inactive. (Corresponds to bit3 in register 07EH).
IRQ	The symbol ☒ signifies that an IRQ will be generated when the counter counts down to 0 (irq mask bit 3 register 07DH). The symbol ☐ signifies that the IRQ function is inactive.
Autoreload	The symbol ☒ signifies that the autoreload function (bit 3 register 063H) is active and the symbol ☐ that it is inactive.
Input	Specifies the counting frequency used by the counter.
Loaded	The value with which the counter was initialised and which will be used for reloading if the autoreload function is active.
Actual	The actual counter value.

Table 3-5 Timer parameters values

3.7. Supply level detection

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V will not stop the software simulation of the microcontroller.

4. EM6507

The peripheral interface module for the EM6507 is shown below.
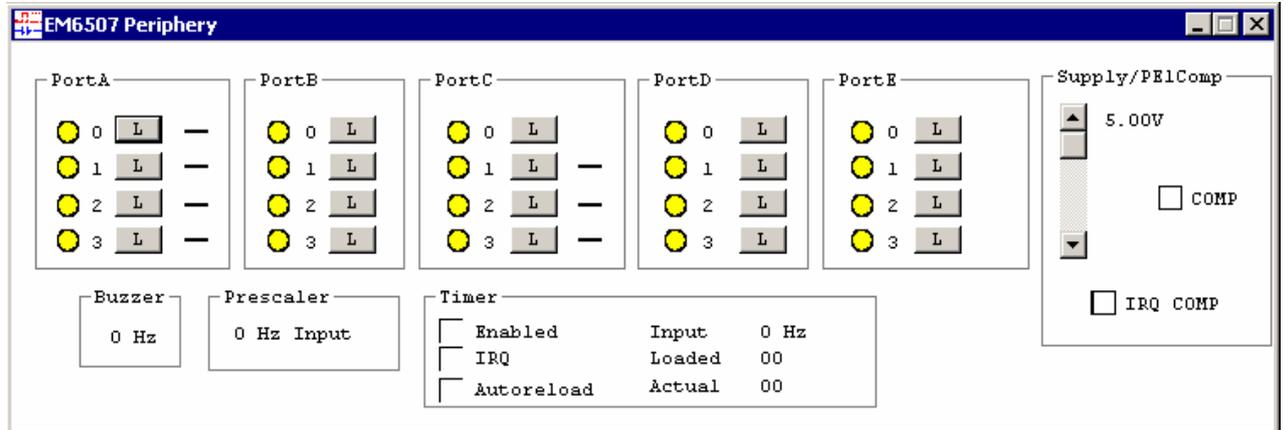
Fig. 4.1 Peripheral interface for the EM6507 microcontroller

4.1. Port A

Port A is an input port with individual maskable interrupts. The circular LED's as well as the push buttons indicates the state of port A (address 072H) and can have values shown in Table 4-1

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 4-1 Possible input Table port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 073H) is shown next to the button. The possible interrupt mask states are shown in Table 4-2.

Symbol	State
⌐	Interrupt generated on the falling edge of the port
⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 4-2 Representation of possible IRQ mask states.

4.2. Port B

Port B is a bitwise select input output port. The state of port B (address 075H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 076H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 4-3.

LED state	Button	State
Red	H	Port input and high
Yellow	L	Port input and low
Red	NC	Port output and high
Yellow	NC	Port output and low

Table 4-3 Possible states for input output port B.

4.3. Port C

Port C can be configured as either a 4 bit input port or a 4 bit output port (bit 1 address 07CH). The possible combinations for the LED and button states are the same as for port B and are shown in Table 4-3. Port C has maskable interrupt capabilities (address 079H). The state of the interrupt mask is show next to the port and can have the states shown in Table 4-2.

4.4. Port D / Serial Write Buffer

Port D can be configured as either a 4 bit input port or a 4 bit output port (bit 2 address 07CH). When port D is set to output and serial write mode is selected (bit3 address 068H) the serial clock is output on port D0 and the serial data on port D1. The possible combinations for the LED and button states for port D are shown in Table 4-3. When in serial write mode the symbol « NC » is replaced by « SCK » (serial clock) for port D0 and « SD » (serial data) for port D1.

4.5. Port E

Port E is a bitwise select input output port. The state of port E (address 066H) is shown by the LED and buttons. The possible combinations for the LED and button states are the same as for port B and are shown in Table 4-3. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 067H) the button is set to NC (not connected) and the LED indicates the state of the output port. The button is set COMP when the comparator is selected (bit 0 address 07BH)

4.6. Prescaler

The selected prescaler interrupt frequency is indicated. This corresponds to the value written to bits 0-1 in the register 07DH.

4.7. Buzzer

Indicates the output frequency selected and corresponds to bits 0-1 in the register 07EH.

4.8. Timer

```
┌─ Timer ──────────────────────────────┐
│  ┌─                                   │
│  └─  Enabled      Input    0 Hz       │
│  ┌─                                   │
│  └─  IRQ          Loaded   00         │
│  ┌─                                   │
│  └─  Autoreload   Actual   00         │
└───────────────────────────────────────┘
```

Fig. 4-2 Timer parameters

The following parameters are defined for the 8-bit timer module.

Parameter	Signification
Enabled	The symbol ☒ signifies the timer is enabled and running. The symbol ☐ signifies the timer is inactive. (Corresponds to bit3 in register 07EH).
IRQ	The symbol ☒ signifies that an IRQ will be generated when the counter counts down to 0 (irq mask bit 3 register 07DH). The symbol ☐ signifies that the IRQ function is inactive.
Autoreload	The symbol ☒ signifies that the autoreload function (bit 3 register 063H) is active and the symbol ☐ that it is inactive.
Input	Specifies the counting frequency used by the counter.
Loaded	The value with which the counter was initialised and which will be used for reloading if the autoreload function is active.
Actual	The actual counter value.

Table 4-4 Timer parameters values

4.9. Supply level detection and PE1 Comparator
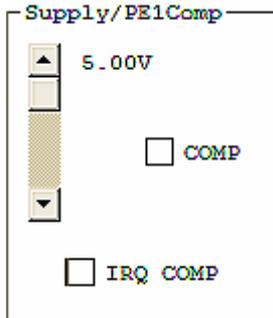
fig. 4-3 SVLD comparator parameters

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V will not stop the software simulation of the microcontroller.

The following parameters are defined for the PE1 comparator module.

Parameter	Signification
COMP	The symbol ☒ signifies the comparator is enabled. The symbol ☐ signifies the comparator is not running. (Corresponds to bit1 in register 07BH).
IRQ COMP	The symbol ☒ signifies that an IRQ will be generated when the comparator reached the detection level. (irq mask bit 3 register 06DH). The symbol ☐ signifies that the IRQ function is inactive.

The scroll bar is used to simulate the level of the input of PE1. The detection level is fixed by the SVLD selected bits.

5. EM6517

The peripheral interface module for the EM6517 is shown below.
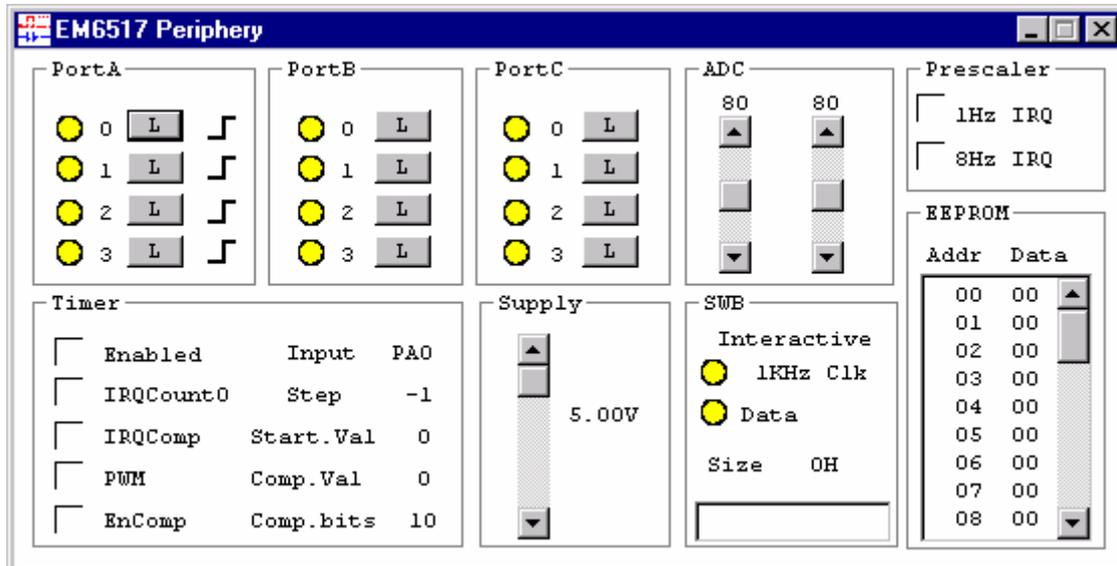
Fig. 5-1 Peripheral interface for the EM6517 microcontroller

5.1. Port A

Port A is an input port with individual maskable interrupts. The circular LED's as well as the push buttons indicates the state of port A (address 050H) and can have values shown in Table 5-1.

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 5-1 Possible input Table port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 065H) is shown next to the button. The possible interrupt mask states are shown in Table .

Symbol	State
⌐_	Interrupt generated on the falling edge of the port
_⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 5-2 Representation of possible IRQ mask states

5.2. Port B

Port B is a bitwise select input output port. The state of port B (address 052H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 051H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 5-3.

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 5-3 Possible states for input output port B.

5.3. Port C

Port C can be configured as either a 4 bit input port or a 4 bit output port (bit 1 address 053H). The possible combinations for the LED and button states are the same as for port B and are shown in Table 5-3.

5.4. ADC

The EM6517 has two 8bit ADC channels. Using the scroll bars can modify the value of each channel. The actual value of each channel is shown.

5.5. Prescaler

The state of the 1Hz and 32Hz interrupt masks are shown. The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive

5.6. Supply level detection

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V would not stop the software simulation of the microcontroller.

5.7. 10 bit timer

The parameters for the 10 bit counter and there possible values are the following:

```
┌Timer─────────────────────────┐
│  ☐  Enabled      Input    PA0 │
│  ☐  IRQCount0    Step      -1 │
│  ☐  IRQComp      Start.Val  0 │
│  ☐  PWM          Comp.Val   0 │
│  ☐  EnComp       Comp.bits 10 │
└───────────────────────────────┘
```

Parameter	Signification	Possible values
Enabled	Timer enabled (bit 3 reg. 05CH)	☐ Disabled ☒ Enabled
IRQ	IRQ mask active for limit event (bit 1 reg. 06AH)	☐ Disabled ☒ Enabled
Comp.IRQ	IRQ mask active for compare event (bit 0 reg. 06AH)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port B3 (bit 3 reg. 06DH)	☐ Disabled ☒ Enabled
Comp. Enabled	Compare function enabled (bit 1 reg. 05CH)	☐ Disabled ☒ Enabled
Input	Input event for counter	Port A0 Port A3 16KHz 2048Hz 512Hz 128Hz 8Hz 1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits

Table 5-4 10 bit Timer parameters and states

5.8. Serial write buffer

The serials write buffer parameters are shown. The output frequency is indicated next to the representation of the serial clock. The CLK LED indicates the state of the clock and the state of the data line is indicated by the state of the data LED. The data transmitted as well as the size of the data transmitted is indicated at the bottom of the window.

```
┌─ SWB ──────────┐
│   Interactive  │
│  ◯  1KHz Clk   │
│  ◯ Data        │
│                │
│  Size    OH    │
│ ┌────────────┐ │
│ └────────────┘ │
└────────────────┘
```
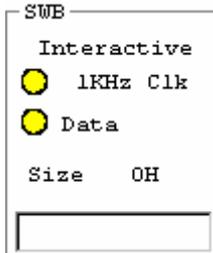
5.9. EEPROM

The contents of the internal EEPROM of the EM6517 are shown in the EEPROM section. The data is in hexadecimal format and is read only.
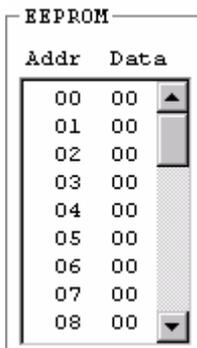
```
┌─ EEPROM ──────┐
│  Addr   Data  │
│ ┌──────────┬─┐│
│ │ 00   00  │▲││
│ │ 01   00  │ ││
│ │ 02   00  │ ││
│ │ 03   00  │ ││
│ │ 04   00  │ ││
│ │ 05   00  │ ││
│ │ 06   00  │ ││
│ │ 07   00  │ ││
│ │ 08   00  │▼││
│ └──────────┴─┘│
└───────────────┘
```

6. EM6520

The peripheral interface module for the EM6520 is shown below.
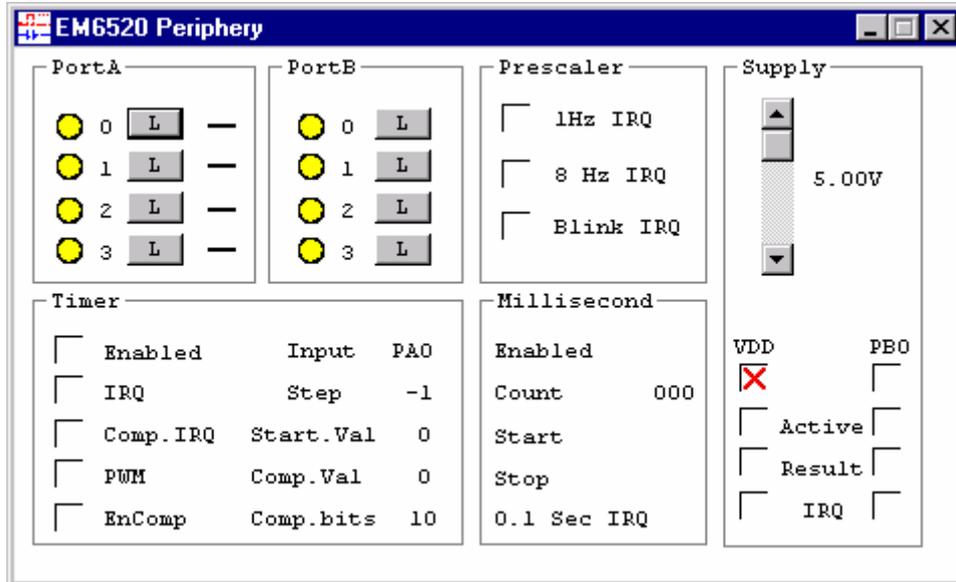
Fig. 6-1 Peripheral interface for the EM6520 microcontroller

6.1. Port A

Port A is an input port with individual maskable interrupts. The circular LED's as well as the push buttons indicates the state of port A (address 050H) and can have values shown in Table 6-1.

LED state	Button	State
Red	H	Port high
Yellow	L	Port low

Table 6-1 Possible states of input port A

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 065H) is shown next to the button. The possible interrupt mask states are shown in Table 6-2.

Symbol	State
⌐	Interrupt generated on the falling edge of the port
⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 6-2 Representation of possible IRQ mask states.

6.2. Port B

Port B is a bitwise select input output port. The state of port B (address 052H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 051H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 6-3.

LED state	Button	State
Red	H	Port input and high
Yellow	L	Port input and low
Red	NC	Port output and high
Yellow	NC	Port output and low

Table 6-3 Possible states for input output port B.

6.3. Prescaler

The state of the 1Hz, 32Hz and Blink interrupt masks are shown. The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive.

6.4. 10 bit timer

The parameters for the 10 bit counter and there possible values are the following:

Parameter	Signification	Possible values
Enabled	Timer enabled (bit 3 reg. 05CH)	☐ Disabled ☒ Enabled
IRQ	IRQ mask active for limit event (bit 1 reg. 06AH)	☐ Disabled ☒ Enabled
Comp.IRQ	IRQ mask active for compare event (bit 0 reg. 06AH)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port B3 (bit 3 reg. 06DH)	☐ Disabled ☒ Enabled
Comp. Enabled	Compare function enabled (bit 1 reg. 05CH)	☐ Disabled ☒ Enabled
Input	Input event for counter	Port A0 Port A3 16KHz 2048Hz 512Hz 128Hz 8Hz 1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits

Table 6-4 10 bit Timer parameters and states

6.5. Supply level detection

The state of the supply voltage level detector parameters is indicted in the section supply. The source used of detection as well as the Measurement State and result are shown.

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps of 0.1V can be

obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V would not stop the software simulation of the microcontroller.

6.6. LCD output

If no custom LCD configuration display is defined for a project the default display configuration shown in Fig. 6-2 is used.
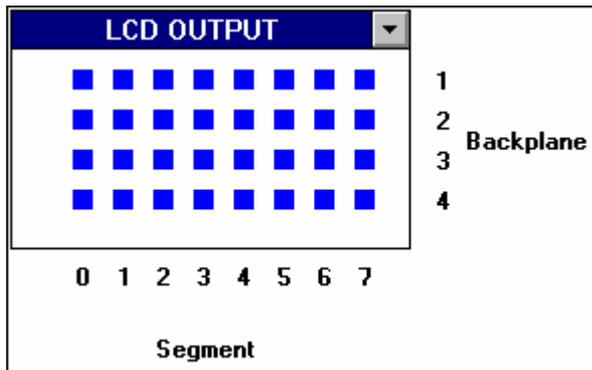
Fig. 6-2 EM6620 default LCD output

The backplane represents the bits of the LCD address space and the segment represents the address of the LCD register (segment 0 is equal address 040 hex and segment is equal to address 047 hex).

7. EM6521 & EM6522

The peripheral interface module for the EM6522 is shown below.
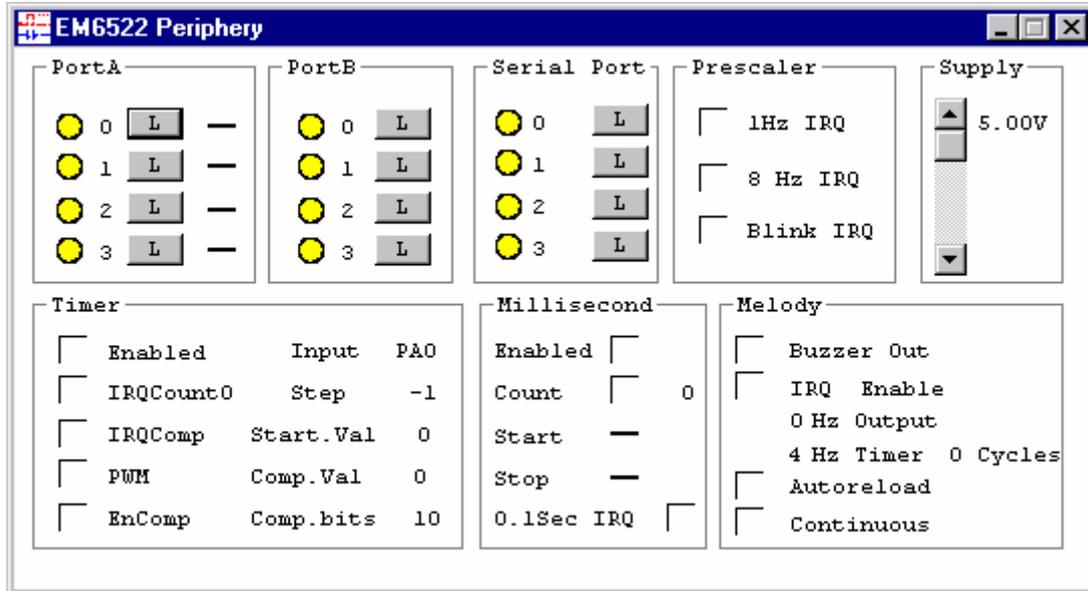
Fig. 7-1 Peripheral interface for the EM6522 microcontroller

7.1. Port A

Port A is an input port with individual maskable interrupts. The circular LED's as well as the push buttons indicates the state of port A (address 050H) and can have values shown in Table 7-1.

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 7-1 Possible input Table port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 065H) is shown next to the button. The possible interrupt mask states are shown in Table 7-2.

Symbol	State
⌐_	Interrupt generated on the falling edge of the port
_⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 7-2 Representation of possible IRQ mask states.

7.2. Port B

Port B is a bitwise select input output port. The state of port B (address 052H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 051H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 7-3.

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 7-3 Possible states for an input output port.

7.3. Serial Port

The serial port can be defined as a parallel input port or a parallel output port. When in either of these modes the possible combinations of the LED states and button states are the same as those for port B (see Table 7-3 Possible states for an input output port.).

When in serial mode the ports have the following attributions

Port	Function	Possible states
In	Serial Data In	Input
Clk	Serial Clock	Output in master mode Input in slave mode
Out	Serial Data Out	Output
S	Status - Chip select (bit 2 reg.054H)	Output

Table 7-4 Serial port states

7.4. Prescaler

The state of the 1Hz, 8Hz and Blink interrupt masks are shown. The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive.

7.5. Millisecond Counter

The parameters for the millisecond counter and there possible values are the following:

Parameter	Signification	Possible values
Enabled	Millisecond counter enabled	☐ Disabled ☒ Enabled
Count	Counter running and current value shown is shown.	☐ Disabled ☒ Enabled
Start	Start event	— CPU controller ⌐ Rising edge PA3 ⌐ Falling edge PA3
Stop	End event	— CPU controller ⌐ Rising edge PA3 ⌐ Falling edge PA3
0.1Sec IRQ	100 ms IRQ active	☐ Disabled ☒ Enabled

Table 7-5 Millisecond counter parameter states

7.6. 10 bit Timer

The parameters for the 10 bit counter and there possible values are the following:

Parameter	Meaning	Possible values
Enabled	Timer enabled (bit 3 reg. 05CH)	☐ Disabled ☒ Enabled
IRQCount0	IRQ mask active for Count 0 event (bit 1 reg. 06AH)	☐ Disabled ☒ Enabled
IRQComp	IRQ mask active for compare event (bit 0 reg. 06AH)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port B3 (bit 3 reg. 06DH)	☐ Disabled ☒ Enabled
EnComp	Compare function enabled (bit 1 reg. 05CH)	☐ Disabled ☒ Enabled
Input	Input event for counter	Port A0 Port A3 16KHz 2048Hz 512Hz 128Hz 8Hz 1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Start Val	Initial Value	0 - 3FF
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits

Table 7-6 10 bit Timer parameters and states

7.7. Melody Generator

The parameters for the 7-tone melody generator and there possible values are the following:

Parameter	Signification	Possible values
Buzzer Out	Buzzer output activate	☐ Disabled ☒ Enabled
IRQ Enabled	IRQ mask active count down event	☐ Disabled ☒ Enabled
XX Hz Output	Output frequency	0 Hz 1365 Hz 1638 Hz 2048 Hz 2340 Hz 2730 Hz 3276 Hz 4096 Hz
Timer frequency and counter	Base frequency for the timer and the number of events before an IRQ	1, 4,16, 64 Hz Clk events 0-0FH
Autoreload	The state of the autoreload function	☐ Disabled ☒ Enabled
Continuous	Frequency is continuously output and is independent of the timer	☐ Disabled ☒ Enabled

Table 7-7 Melody generator parameters and possible states

7.8. Supply level detection

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V will not stop the software simulation of the microcontroller.

7.9. LCD Output

If no custom LCD configuration display is defined for a project the default display configuration shown in Fig. 7-2 is used.
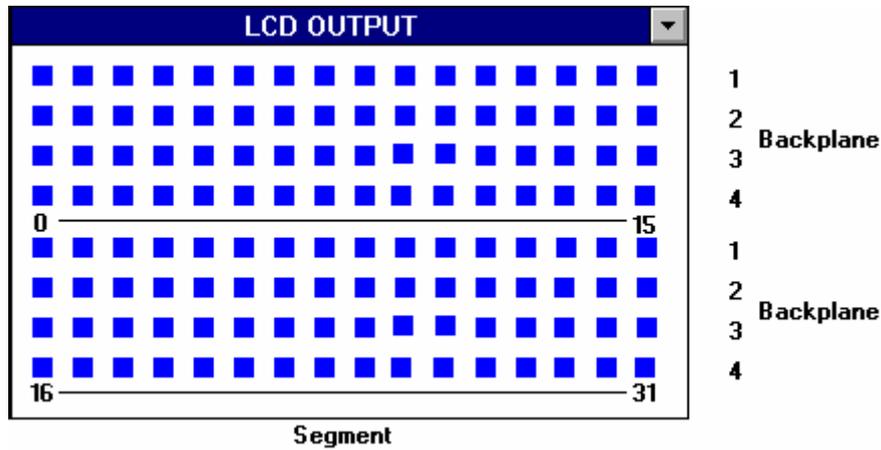
Fig. 7-2 EM6622 default LCD output

The backplane represents the bits of the LCD registers. Segments 0 to 15 map to the address space of the register LCD_1 and segments 16 to 31map to the address space of LCD_2.

8.   EM6525 & EM6526

The peripheral interface module for the EM6526 is shown below.
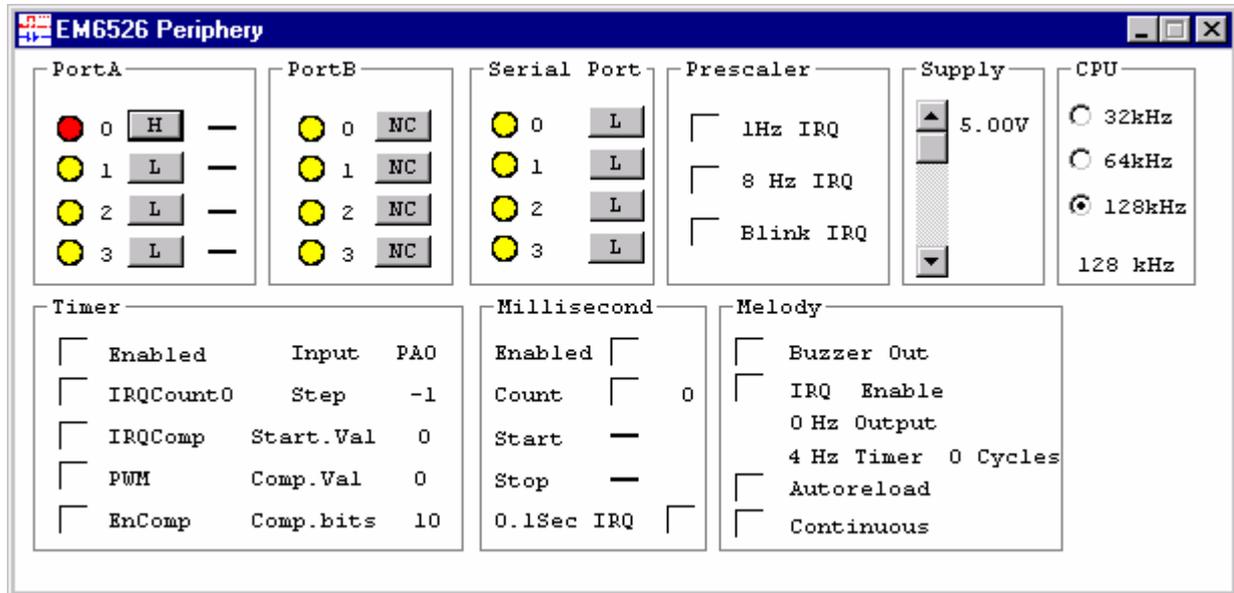
Fig. 8-1 Peripheral interface for the EM6526 microcontroller

8.1.   Port A

Port A is an input port with individual maskable interrupts. The circular LED's as well as the push buttons indicates the state of port A (address 050H) and can have values shown in Table 8-1.

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 8-1 Possible input Table port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 065H) is shown next to the button. The possible interrupt mask states are shown in Table 9-2.

Symbol	State
⌐	Interrupt generated on the falling edge of the port
_⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 8-2 Representation of possible IRQ mask states.

8.2. Port B

Port B is a bitwise select input output port. The state of port B (address 052H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 051H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 8-3.

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 8-3 Possible states for an input output port.

8.3. Serial Port

The serial port can be defined as a parallel input port or a parallel output port. When in either of these modes the possible combinations of the LED states and button states are the same as those for port B (see Table 8-3 Possible states for an input output port.).

When in serial mode the ports have the following attributions

Port	Function	Possible states
In	Serial Data In	Input
Clk	Serial Clock	Output in master mode Input in slave mode
Out	Serial Data Out	Output
S	Status - Chip select (bit 2 reg.054H)	Output

Table 8-4 Serial port states

8.4. Prescaler

The state of the 1Hz, 8Hz and Blink interrupt masks are shown. The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive.

8.5. Millisecond Counter

The parameters for the millisecond counter and there possible values are the following:

Parameter	Signification	Possible values
Enabled	Millisecond counter enabled	☐ Disabled ☒ Enabled
Count	Counter running and current value shown is shown.	☐ Disabled ☒ Enabled
Start	Start event	— CPU controller ⎍ Rising edge PA3 ⎍ Falling edge PA3
Stop	End event	— CPU controller ⎍ Rising edge PA3 ⎍ Falling edge PA3
0.1Sec IRQ	100 ms IRQ active	☐ Disabled ☒ Enabled

Table 8-5 Millisecond counter parameter states

8.6. 10 bit Timer

The parameters for the 10 bit counter and there possible values are the following:

Parameter	Meaning	Possible values
Enabled	Timer enabled (bit 3 reg. 05CH)	☐ Disabled ☒ Enabled
IRQCount0	IRQ mask active for Count 0 event (bit 1 reg. 06AH)	☐ Disabled ☒ Enabled
IRQComp	IRQ mask active for compare event (bit 0 reg. 06AH)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port B3 (bit 3 reg. 06DH)	☐ Disabled ☒ Enabled
EnComp	Compare function enabled (bit 1 reg. 05CH)	☐ Disabled ☒ Enabled
Input	Input event for counter	Port A0 Port A3 16KHz 2048Hz 512Hz 128Hz 8Hz 1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Start Val	Initial Value	0 - 3FF
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits

Table 8-6 10 bit Timer parameters and states

8.7. Melody Generator

The parameters for the 7-tone melody generator and there possible values are the following:

Parameter	Signification	Possible values
Buzzer Out	Buzzer output activate	☐ Disabled ☒ Enabled
IRQ Enabled	IRQ mask active count down event	☐ Disabled ☒ Enabled
XX Hz Output	Output frequency	0 Hz 1365 Hz 1638 Hz 2048 Hz 2340 Hz 2730 Hz 3276 Hz 4096 Hz
Timer frequency and counter	Base frequency for the timer and the number of events before an IRQ	1, 4,16, 64 Hz Clk events 0-0FH
Autoreload	The state of the autoreload function	☐ Disabled ☒ Enabled
Continuous	Frequency is continuously output and is independent of the timer	☐ Disabled ☒ Enabled

Table 8-7 Melody generator parameters and possible states

8.8. Supply level detection

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V will not stop the software simulation of the microcontroller.

8.9. CPU Frequency level

The equivalent ROM version allows the user to select by metal option 3 different system operating clock for the CPU 32kHz, 64kHz or 128kHz.

The clock for the peripherals is always 32 kHz.

Pressing the appropriate button on the simulator can simulate these frequencies.

8.10. LCD Output

If no custom LCD configuration display is defined for a project the default display configuration shown in Fig. 8-2 is used.
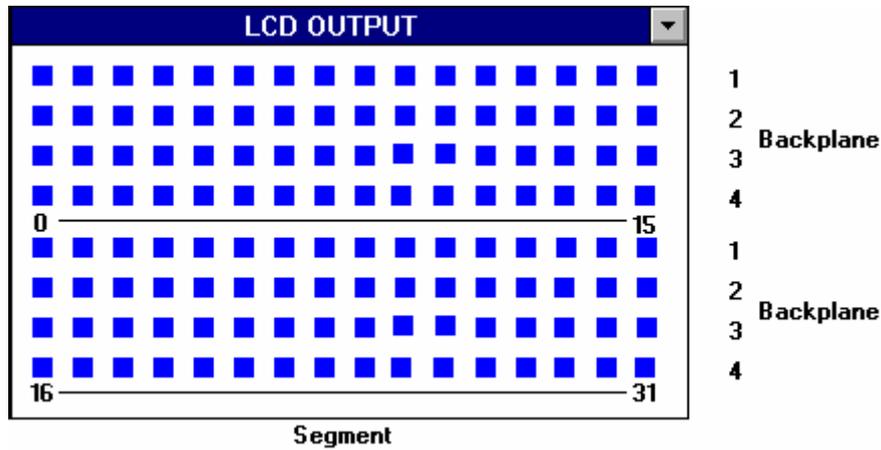
Fig. 8-2 EM6626 default LCD output

The backplane represents the bits of the LCD registers. Segments 0 to 15 map to the address space of the register LCD_1 and segments 16 to 31 map to the address space of LCD_2.

9. EM6533

9.1. General Concepts

9.1.1. Paged RAM

The paged RAM concept of the 6633 is treated as follows:
the IDE (Integrated Development Environment) can only display one Page at a time, so that, whenever the RAM page is switched, the complete new page is displayed in the RAM Window; any changes made on the RAM Window (or on the Watch window) affect only the actual page. As there is no way to declare different Symbols for the different pages, their names remain the same, independently of the page actually selected.

9.1.2. Debouncers and Pulldowns/Pullups

These features are not available on the simulator

9.1.3. Prescaler Inhibition

This feature is not available on the simulator

9.1.4. Emulation restriction

The fact that the Interrupt status registers are reset when read becomes a problem at emulation time. Whenever the emulator stops (breaks), the IDE reads and uploads the contents of all registers and RAM positions from the Emulator Module and so the contents of all IRQ Registers are always rested under this condition!

This fact could cause you problems if you have to debug the Interrupt Routine; the right thing to do to avoid these problems is:

a) Save the contents of the IRQ Registers at the beginning of the Interrupt Routine or as soon as possible.

b) don't put breakpoints before this Save has been done

9.2. Detailed Description
The peripheral interface module for the EM6533 is shown below.
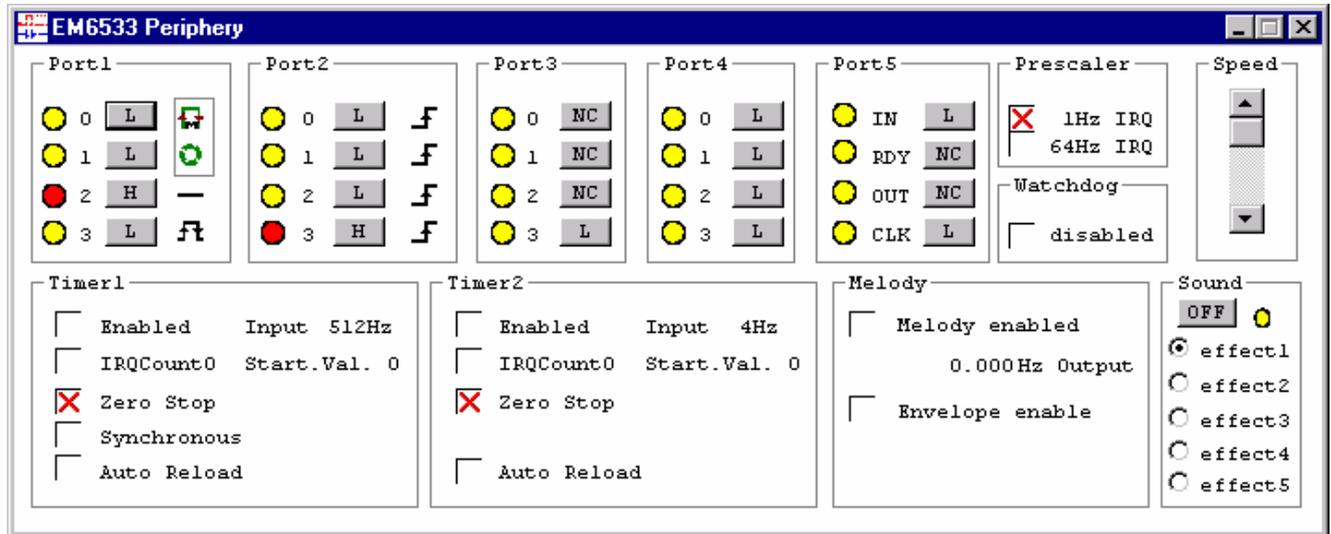
Fig. 9-1. Peripheral interface for the EM6533 microcontroller

9.2.1. Port 1

Port 1 is an input port with special interrupt sources. The state of port 1 is indicated by the circular LED's as well as the push buttons and can have values as shown in Table 9-1.

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 9-1 Possible input port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port 1 is shown next to the button.

The possible interrupt mask states (for any ports with interrupt capabilities) are shown in Table 9-2.

Symbol	State
⅂	Interrupt generated on the falling edge of the port
⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled
⅃⅂	Interrupt generated on both edges
◯	Interrupt generated by change in rotation direction
🔲	Interrupt generated by M-signal

Table 9-2 Representation of possible IRQ mask states.

Rotation detection and M-Signal interrupts are special ones in that they are generated as a combined result of bits 0 and 1 and this is shown by means of the rectangular frame surrounding the IRQ symbols for these bits
(see 2).

9.2.2. Port 2

Port 2 is an input port with only one general Interrupt (Positive edge « ored »). The state of port 1 is shown by the LED's and can be changed by depressing the Buttons. See Table 9-1.

9.2.3. Port 3

Port 3 is a block select input/output Port with no Interrupt generating capabilities. Bits 0 to 2 are block select as Input or Output (**RegPIOCntl**-0) and Bit 3 is undependable select as Input or Output (**RegPIOCntl**-2). The possible states of Port 3 are shown in Table 9-3.

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 9-3 Possible states for input output port.

9.2.4. Port 4

Port 4 is very similar to Port 3. Bits 0 to 2 are block select as Input or Output (**RegPIOCntl**-1) and Bit 3 is undependable select as Input or Output (**RegPIOCntl**-3). The possible states of Port 4 are shown in Table 9-3

9.2.5. Port 5

Port 5 is a general, bit select, input/output Port with no Interrupt capabilities. The direction of each bit is programmable by means of **RegP5Cntl**.). The possible states of Port 5 are shown in Table 9-3.

Port 5 can also function as Serial Port.

9.2.6. Serial Interface

If **RegSIOCntl**-1 is « 1 », then Port 5 will function as Serial Interface (SIO). In this case, the names of the bits change to those shown in

Fig. **9-2** and explained in Table 9-4.
Note: For a more detailed description of the different Port Serial modes, consult your Controller Manual
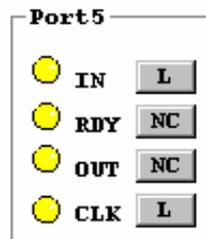
Fig. 9-2. Port 5 configured as Serial Port

Port-bit	Function	Possible states
IN	Serial Data In	Input
RDY	Status - Chip select (bit 2 reg.054H)	Output
OUT	Serial Data Out	Output
CLK	Serial Clock	Output in master mode Input in slave mode

Table 9-4 Serial port states

9.2.7. Prescaler

The different IRQ possibilities generated by the Prescaler are shown on the IO-Panel (see 2). The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive.

9.2.8. Watchdog

Function	Possible states
Watchdog	☐ Disabled ☒ Enabled

Table 9-5 Watchdog configuration

9.2.9. Timer 1

The different functioning modes of Timer 1 are shown in Table 9-6

Parameter	Signification	Possible values
Enabled	the timer is active (counting)	☐ Disabled ☒ Enabled
IRQCount0	Interrupt when count arrives to 0	☐ Disabled ☒ Enabled
Zero Stop	Zero Stop Mode	☐ Disabled ☒ Enabled
Synchronous	Synchronous Mode	☐ Disabled ☒ Enabled
Auto Reload	Auto Reload Mode	☐ Disabled ☒ Enabled
Input XXHz	Input Clock selection	*Value in Hz*
Start Val.	Start Value of the timer	*Any 4 Bit value*

Table 9-6 Timer1 parameters and possible states

9.2.10.   Timer 2

Timer 2 is very similar to Timer 1, except that Timer 2 has no Synchronous Mode.

9.2.11. Melody Generator

The different settings for melody generation are shown on Table 9-7. :

Parameter	Signification	Possible values
Melody enabled	Buzzer output activated	☐ Disabled ☒ Enabled
XX Hz Output	Output frequency value in Hz	practically any value, calculated as $65536/(^1Data+1)$
Envelope enable	Tone decay envelope	☐ Disabled ☒ Enabled
	if no Envelope selected, the picture will be	
	if long Envelope selected, the picture will be	
	if short Envelope selected, the picture will be	

Table 9-7. Melody generator parameters and possible states

[1] Data is the contents of RegFGLo and RegFGHi together

10. EM6540

The peripheral interface module for the EM6540 is shown below.
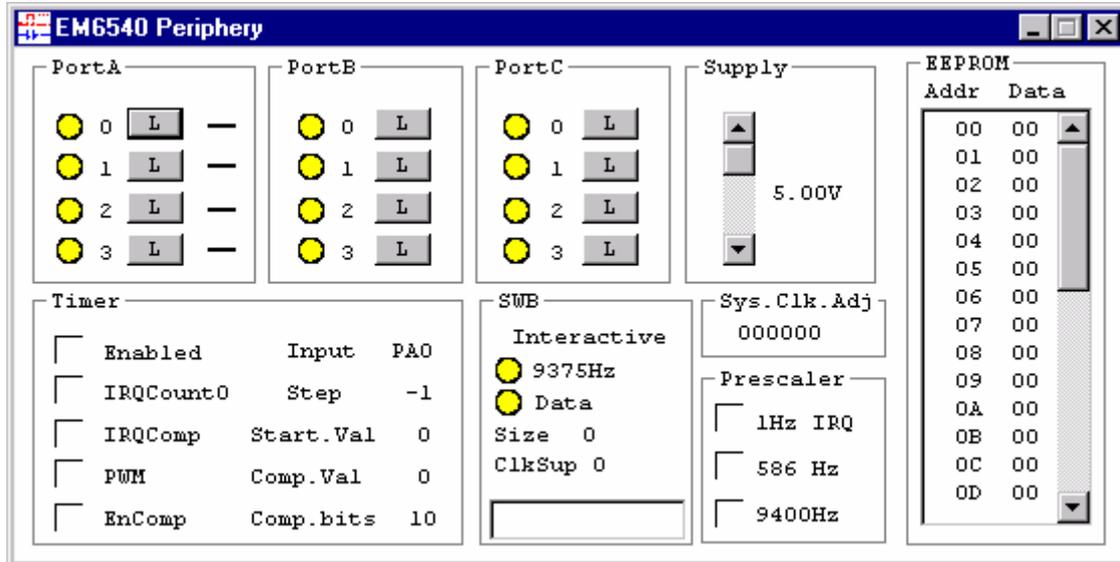
10.1. Port A

Port A is an input port with individual maskable interrupts. The state of port A (address 054H) is indicated by the by the circular LED's as well as the push buttons and can have values shown in Table 11-1.

LED state	Button	State
🔴 Red	H	Port high
🟡 Yellow	L	Port low

Table 10-1 Possible input port states

to change the state of the port click on the corresponding button.

The state of the interrupt mask for port A (address 062H) is shown next to the button. The possible interrupt mask states are shown in Table 11-2.

Symbol	State
⅃	Interrupt generated on the falling edge of the port
⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 10-2 Representation of possible IRQ mask states.

10.2. Port B

Port B is a bitwise select input output port. The state of port B (address 056H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 076H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 11-3.

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 10-3 Possible states for input output port.

10.3. Port C

Port C is similar to Port B when used as general I/O Port, but it can also overtake the signals of the SWB, as explained below.

10.4. Serial Write Buffer (SWB)
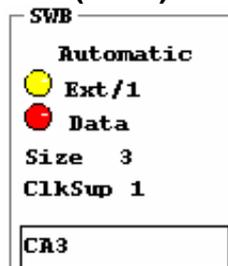
Fig. 10-2. Representation of the SWB on the IO-Panel

When the Serial Write Buffer is enabled, Port C-bit 0 overtakes the Clock Signal and Port C-bit 1 the Data Signal of the SWB.
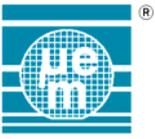
On the IO-Panel, there is a region dedicated specially to the SWB, where the Clock and Data Signals are represented once more (duplicated against the normal Port C representation).

Additional information available on the IO-Panel is the following:

- Mode (Automatic/Interactive)

- Clock (external/internal and division factor)

- Clock suppression (how many bits are suppressed)

- The data being sent: each nibble already sent will appear (from left to right) on the text box (in hex representation).

Note : it is important to emphasise that this represented value reflects the suppression of clocks ; for instance, in the example shown on fig 10-2, although the buffered data was 'C,A,B' , the transmitted data is actually considered to be 'C,A,3', because the last clock is suppressed !

10.5. Prescaler

The selected prescaler interrupt frequency is indicated. This corresponds to the value written to bits 1- 3 in the register 062H.

10.6. Timer

Fig. 10-3 Timer parameters

The following parameters are defined for the 8-bit timer module.

Parameter	Meaning	Possible values
Enabled	Timer enabled (bit 3 reg. 05EH)	☐ Disabled ☒ Enabled
IRQCount0	IRQ mask active for Count 0 event (bit 1 reg. 064H)	☐ Disabled ☒ Enabled
IRQComp	IRQ mask active for compare event (bit 0 reg. 064H)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port B3 (bit 3 reg. 06BH)	☐ Disabled ☒ Enabled
EnComp	Compare function enabled (bit 1 reg. 05EH)	☐ Disabled ☒ Enabled
Input	Input clock for counter	Port A0 Port A3 300KHz 37.5KHz 4688Hz 586Hz 9.2Hz 1.1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Start Val	Initial Value	0 - 3FF
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. Bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits

Table 10-4 Timer parameters values
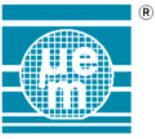
10.7. Supply level detection

It is possible to simulate changes in the supply level from 0V to 5V with the use of the scroll bar. The minimum precision is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. It should be noted that reducing the supply voltage to 0V would not stop the software simulation of the microcontroller.

10.8. EEPROM

The contents of the internal EEPROM of the EM6640 are shown in the EEPROM section. The data is in hexadecimal format and is read only.

10.9. Sys.Clk.Adj.

Shows the 6-bit configuration (OscAdj [0] to OscAdj [5]) contained in register OPTOscAdj and OPTPaRST. These bits are used to « trim » the chip's RC oscillator. The simulated controller only displays the status of this trimming, but, as it is not « real time », its speed is not affected.

11. EM6580

The peripheral interface module for the EM6580 is shown below.
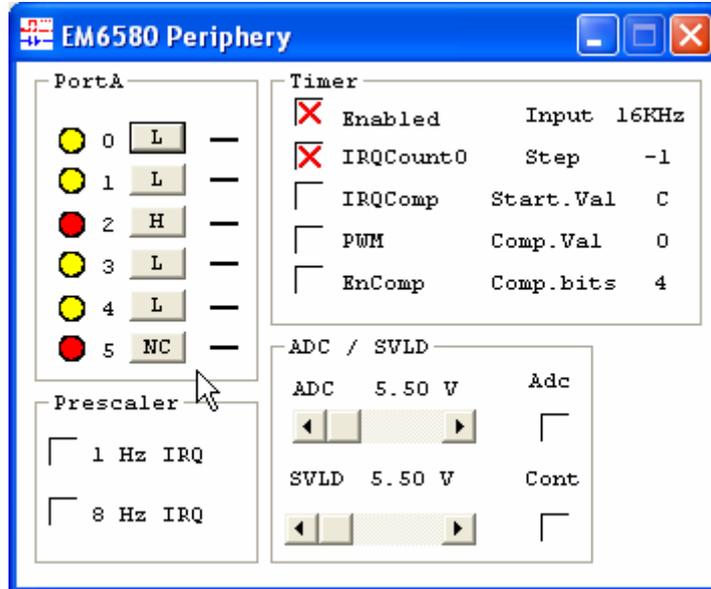
Fig. 11-1 Peripheral interface for the EM6580 microcontroller

11.1. Port A

Port A is a bitwise select input output port excepted PA4 always in input mode. The state of port A (address 050H and address 060H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 051H and address 060H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table11-1.

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 11-1 Possible states for input output port.

Port A ,when configured in input, has individual maskable interrupts on PA0, PA3, PA4, PA5. The state of the interrupt mask for port A (address 067H) is shown next to the button. The possible interrupt mask states are shown in Table 11-2.

Symbol	State
⌐_	Interrupt generated on the falling edge of the port
_⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 11-2 Representation of possible IRQ mask states.

11.2. Prescaler

The states of the 1Hz, 8Hz or 64Hz interrupt masks are shown in Fig The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive. The interrupt frequency 8Hz or 64 Hz is display based on the selection (address 06CH) Bit 1.

```
┌Prescaler─────────┐
│ ┌─                │
│ │  1 Hz IRQ       │
│                   │
│ ┌─                │
│ │  8 Hz IRQ       │
│                   │
└───────────────────┘
```
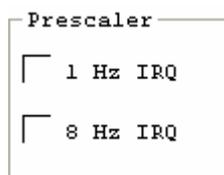
Fig. 11-2 Prescaler

11.3. Timer

```
┌Timer──────────────────────────┐
│ ┌─                            │
│ │  Enabled      Input    PA0  │
│ ┌─                            │
│ │  IRQCount0    Step      -1  │
│ ┌─                            │
│ │  IRQComp      Start.Val  0  │
│ ┌─                            │
│ │  PWM          Comp.Val   0  │
│ ┌─                            │
│ │  EnComp       Comp.bits  10 │
└────────────────────────────────┘
```

Fig. 11-3 Timer parameters

The following parameters are defined for the 8-bit timer module.

Parameter	Meaning	Possible values
Enabled	Timer enabled (bit 3 reg. 05CH)	☐ Disabled ☒ Enabled
IRQCount0	IRQ mask active for Count 0 event (bit 3 reg. 065H)	☐ Disabled ☒ Enabled
IRQComp	IRQ mask active for compare event (bit 2 reg. 065H)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port A 0 or 1 (bit 0 and1 reg. 61H)	☐ Disabled ☒ Enabled
EnComp	Compare function enabled (bit 1 reg. 05CH)	☐ Disabled ☒ Enabled
Input	Input clock for counter	Port A1 Port A3/4 RC/2 RC/4 16KHz 2KHz 512Hz 128Hz 8Hz 1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Start Val	Initial Value	0 - 3FF
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. Bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits

Table 11-3 Timer parameters values

11.4. Port A in serial mode

The Port A can be defined as a serial port. For a serial mode configuration, the ports have the following attributions

Port	Function		Possible states
In	Serial Data In	**PA0**	Input
Clk	Serial Clock	**PA1**	Output in master mode
			Input in slave mode
Out	Serial Data Out	**PA2**	Output
S	Status - Chip select	**PA3**	Output

Table 11-4 Serial port states

11.5. Supply voltage level detector and ADC mode
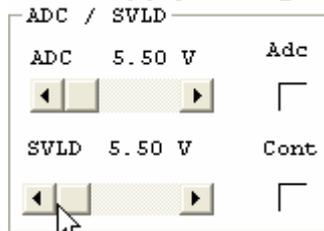
Fig. 11-4 SVLD and ADC selection

It is possible to simulate changes in the supply level or pin level of PortA bit 4 which becomes marked ADC (bit 3 address073H). These changes will be operated from 0V to 5.5V with the use of the appropriate scroll bar. The minimum accuracy is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. Reducing the supply voltage under 1.5V would stop the software simulation of the microcontroller as **POR** emulation**,** Level must be increased above 2V as **Power Check Level** emulation. The symbol ☒ indicates that the ADC or the continuous mode is active (bit 1 address 073H for continuous mode) and the symbol ☐ indicates that the ADC or the continuous mode is inactive.

11.6. Option registers

Please refer to the following table for the registers implementation:

Register **RegOscTrimH**   @ addr. 117 decimal = 75 hex.

Bit	Name	Reset	R/W	Description
3	RegOscTrim[7]	0	R/W	**Not implemented**
2	RegOscTrim[6]	1	R/W	**Not implemented**
1	RegOscTrim[5]	1	R/W	**Not implemented**
0	RegOscTrim[4]	1	R/W	**Not implemented**

Table 11-5 RegOscTrimH

Register **RegOscTrimL**   @ addr. 118 decimal = 76 hex.

Bit	Name	Reset	R/W	Description
3	RegOscTrim[3]	1	R/W	**Not implemented**
2	RegOscTrim[2]	1	R/W	**Not implemented**
1	RegOscTrim[1]	1	R/W	**Not implemented**
0	RegOscTrim[0]	1	R/W	**Not implemented**

Table 11-6 RegOscTrimL

Register **RegMFP0**   @ addr. 121 decimal = 79 hex.

Bit	Name	Reset	R/W	Description
3	Opt[3]	0	R/W	**Not implemented**
2	Opt[2]	0	R/W	Must be kept to "0"
1	Opt[1]	0	R/W	Counter Clock source 7 selection "0" PA3/PA4, "1" RCclk/2
0	Opt[0]	0	R/W	**Not implemented**

Table 11-7 RegMFP0

Note1. bit1: When Opt[1]=1 the clock of the counter is no more ck16 (32kHz/50kHZ) but the CPU clock in order to achieve a correct sampling of the RCclk/2 signal.

Register **RegMFP1**   @ addr. 122 decimal = 7A hex.

Bit	Name	Reset	R/W	Description
3	Opt[7]	0	R/W	RC base frequency (32/50 kHz) selection.
2	Opt[6]	0	R/W	RC frequency selection bit 2
1	Opt[5]	0	R/W	RC frequency selection bit 1
0	Opt[4]	0	R/W	RC frequency selection bit 0

Table 11-8 RegMFP1

Register **RegMFP2**   @ addr. 123 decimal = 7D hex.

Bit	Name	Reset	R/W	Description
3	Opt[11]	?	R/W	
2	Opt[10]	?	R/W	
1	Opt[9]	?	R/W	
0	Opt[8]	0	R/W	ADC/SVLD voltage level#15: "0" 2.75v, "1" 3v

Table 11-9 RegMFP2

12. EM6680

The peripheral interface module for the EM6680 is shown below.
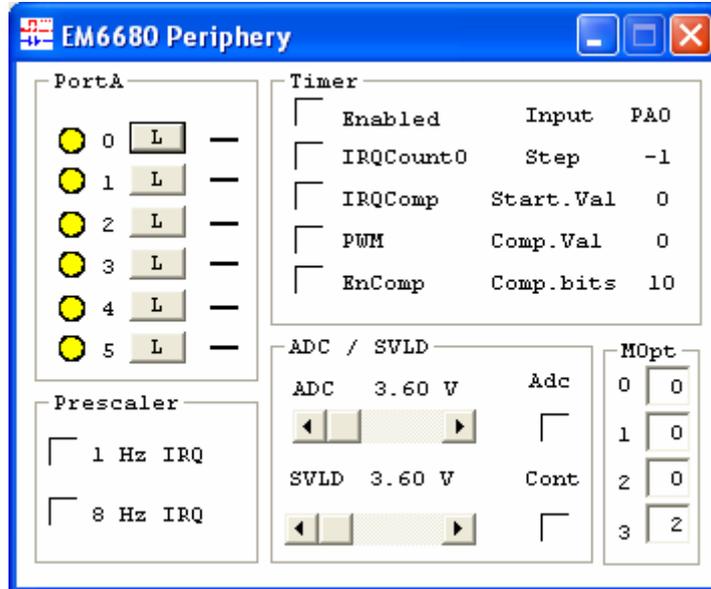
Fig. 12-1 Peripheral interface for the EM6680 microcontroller

12.1. Port A

Port A is a bitwise select input output port excepted PA4 always in input mode. The state of port A (address 050H and address 060H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 051H and address 060H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 12-1.

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 12-1 Possible states for input output port.

Port A ,when configured in input, has individual maskable interrupts on PA0, PA3, PA4, PA5. The state of the interrupt mask for port A (address 067H) is shown next to the button. The possible interrupt mask states are shown in Table 12-2.

Symbol	State
⌐_	Interrupt generated on the falling edge of the port
_⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 12-2 Representation of possible IRQ mask states.

12.2. Prescaler

The states of the 1Hz, 8Hz or 64Hz interrupt masks are shown in Fig12-2 The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive. The interrupt frequency 8Hz or 64 Hz is display based on the selection (address 06CH) Bit 1.

```
┌─Prescaler──────┐
│ ┌─┐            │
│ │ │  1 Hz IRQ  │
│ └─┘            │
│ ┌─┐            │
│ │ │  8 Hz IRQ  │
│ └─┘            │
└────────────────┘
```
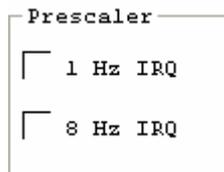
Fig. 12-2 Prescaler

12.3. Timer

```
┌─Timer─────────────────────────┐
│ ┌─┐                           │
│ │ │  Enabled    Input    PA0  │
│ ┌─┐                           │
│ │ │  IRQCount0  Step      -1  │
│ ┌─┐                           │
│ │ │  IRQComp    Start.Val  0  │
│ ┌─┐                           │
│ │ │  PWM        Comp.Val   0  │
│ ┌─┐                           │
│ │ │  EnComp     Comp.bits 10  │
└───────────────────────────────┘
```

Fig. 12-3 Timer parameters

The following parameters are defined for the 8-bit timer module.

Parameter	Meaning	Possible values
Enabled	Timer enabled (bit 3 reg. 05CH)	☐ Disabled ☒ Enabled
IRQCount0	IRQ mask active for Count 0 event (bit 3 reg. 065H)	☐ Disabled ☒ Enabled
IRQComp	IRQ mask active for compare event (bit 2 reg. 065H)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port A 0 or 1 (bit 0 and1 reg. 61H)	☐ Disabled ☒ Enabled
EnComp	Compare function enabled (bit 1 reg. 05CH)	☐ Disabled ☒ Enabled
Input	Input clock for counter	Port A1 Port A3/4 RC/2 RC/4 16KHz 2KHz 512Hz 128Hz 8Hz 1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Start Val	Initial Value	0 - 3FF
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. Bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits

Table 12-3Timer parameters values

12.4. Port A in serial mode

The Port A could be defined as a serial port. For a serial mode configuration, the ports have the following attributions

Port	Function		Possible states
In	Serial Data In	**PA0**	Input
Clk	Serial Clock	**PA1**	Output in master mode
			Input in slave mode
Out	Serial Data Out	**PA2**	Output
S	Status - Chip select	**PA3**	Output

Table 12-4 Serial port states

12.5. Supply voltage level detector and ADC mode

Fig. 12-4 SVLD and ADC selection

It is possible to simulate changes in the supply level or pin level of PortA bit 4 which becomes marked ADC (bit 3 address073H). These changes will be operated from 0V to 3.6V with the use of the appropriate scroll bar. The minimum accuracy is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. Reducing the supply voltage under 1.05V would stop the software simulation of the microcontroller as **POR** emulation, Level must be increase above 1.25V / 1.85V as **Power Check Level** emulation. The symbol ☒ indicates that the ADC or the continuous mode is active (bit 1 address 073H for continuous mode) and the symbol ☐ indicates that the ADC or the continuous mode is inactive.

12.6. Metal Mask Options

Fig. 12-5 MOpt display

The Metal Mask Options are displayed on the MOpt display, (Periphery interface window). These options can only be modified by editing the .prj file of the project. During the modification of the .prj file, the project and the simulator must be closed.

12.6.1. View of an edited .prj file
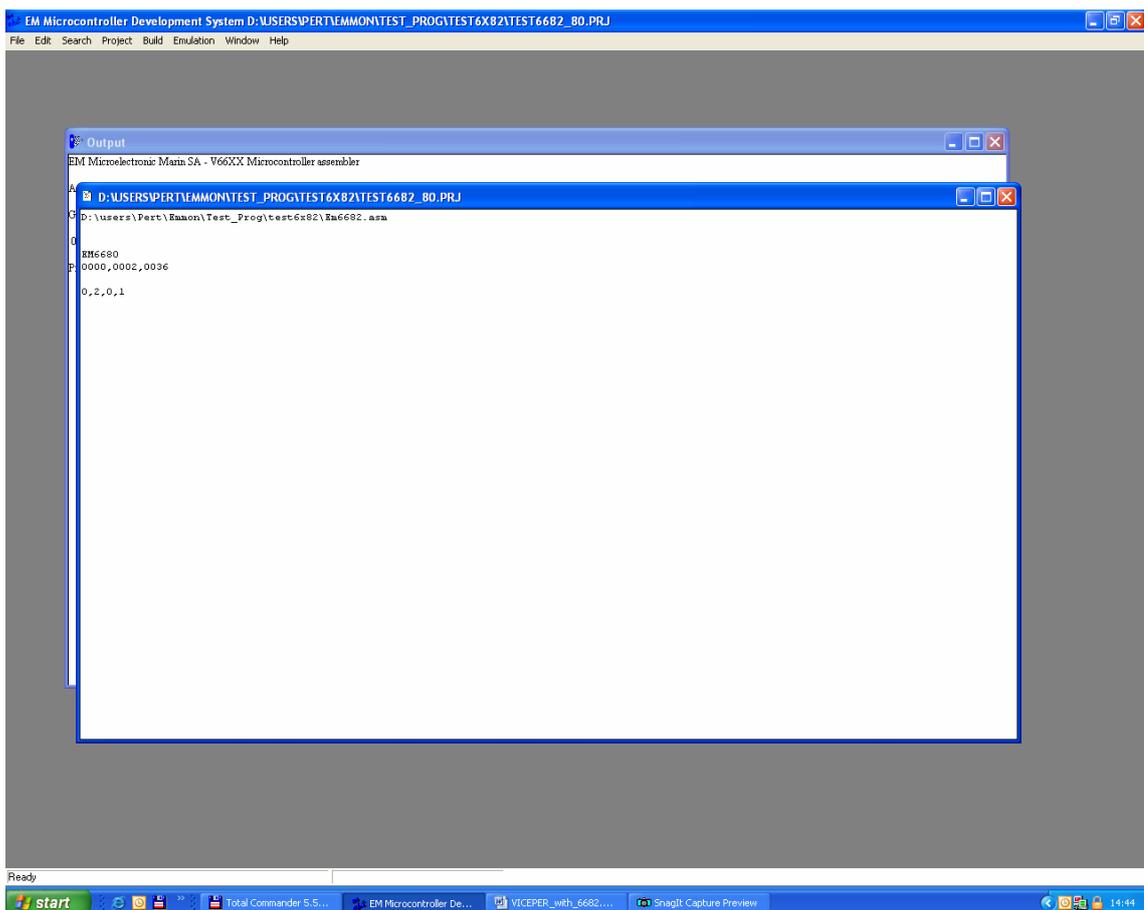
The last line corresponds to the Metal Mask Options; the different options are defined in the following table:

12.6.2. Options table

Bit Field	BIT 7 to 4	Bit 3	Bit 2	Bit 1	Bit 0
MOpt 0	No Effect	No Effect	No Effect	Counter Clock source 7 selection 0 = PA3/PA4 1 = RC/2	No Effect
MOpt 1	No Effect	RC base selection 0 = 32kHz 1 = 50kHz	RC Frequency selection 2 Refer to chapter 15.1 of the EM6580 datasheet	RC Frequency selection 1 Refer to chapter 15.1 of the EM6580 datasheet	RC Frequency selection 0 Refer to chapter 15.1 of the EM6580 datasheet
MOpt 2	No Effect	No Effect	No Effect	No Effect	SVLD level15 selection 0 = 2.75V 1 = 3V
MOpt 3	No Effect	No Effect	No Effect	No Effect	Power check selection 0 = level 5 as 2.75v 1 = level 9 as 3V

Table 12-5Serial port states

After options definition, the .prj file must be saved and the projects re-opened. All the selected options will be updated into the MOpt display.

12.6.3. Metal mask options example

On the fig. 12.6, the selected options are :

MOpt0 = 0x00 => Counter Clock source 7 as PA3/PA4.

MOpt1 = 0x02 => Rc frequency selected is 128kHz.

MOpt2 = 0x00 => ADC/SVLD LEVEL 15 as 2.75V

MOpt3 = 0x01 => Power Check level 9 as 1.85V

13. EM6682

The peripheral interface module for the EM6682 is shown below.
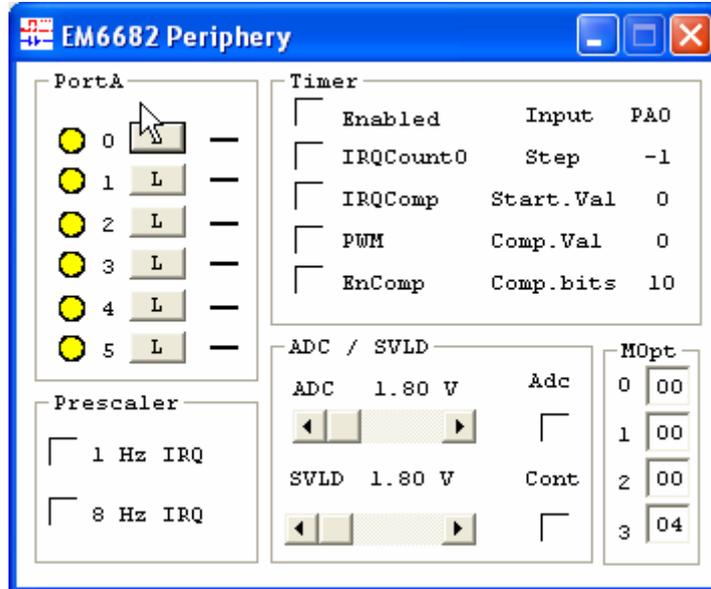
Fig. 13-1 Peripheral interface for the EM6682 microcontroller

13.1. Port A

Port A is a bitwise select input output port excepted PA4 always in input mode. The state of port A (address 050H and address 060H) is shown by the LED and buttons. When the port is input the state can be changed using the corresponding push buttons. When the port is set to output (address 051H and address 060H) the button is set to NC (not connected) and the LED indicates the state of the output port. Consequently the possible combinations for the LED and button states are shown in Table 13.1

LED state	Button	State
🔴 Red	H	Port input and high
🟡 Yellow	L	Port input and low
🔴 Red	NC	Port output and high
🟡 Yellow	NC	Port output and low

Table 13-1 Possible states for input output port.

Port A ,when configured in input, has individual maskable interrupts on PA0, PA3, PA4, PA5. The state of the interrupt mask for port A (address 067H) is shown next to the button. The possible interrupt mask states are shown in Table 13.2.

Symbol	State
⌐_	Interrupt generated on the falling edge of the port
_⌐	Interrupt generated on the rising edge of the port
—	Interrupt disabled

Table 13-2 Representation of possible IRQ mask states.

13.2. Prescaler

The states of the 1Hz, 8Hz or 64Hz interrupt masks are shown in Fig 13-2. The symbol ☒ indicates that the interrupt mask is active and the symbol ☐ indicates that the interrupt mask is inactive. The interrupt frequency 8Hz or 64 Hz is display based on the selection (address 06CH) Bit 1.
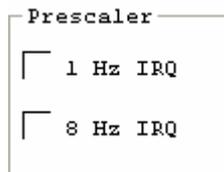
Fig. 13-2 Prescaler

13.3. Timer

Fig. 13-3 Timer parameters

The following parameters are defined for the 8-bit timer module.

Parameter	Meaning	Possible values
Enabled	Timer enabled (bit 3 reg. 05CH)	☐ Disabled ☒ Enabled
IRQCount0	IRQ mask active for Count 0 event (bit 3 reg. 065H)	☐ Disabled ☒ Enabled
IRQComp	IRQ mask active for compare event (bit 2 reg. 065H)	☐ Disabled ☒ Enabled
PWM	Pulse width modulation output to port A 0 or 1 (bit 0 and1 reg. 61H)	☐ Disabled ☒ Enabled
EnComp	Compare function enabled (bit 1 reg. 05CH)	☐ Disabled ☒ Enabled
Input	Input clock for counter	Port A1 Port A3/4 RC/2 or RC RC/4 or RC/2 16KHz 2KHz 512Hz 128Hz 8Hz 1Hz
Step	Direction of counting	+1 = Upward counting -1 = downward counting
Start Val	Initial Value	0 - 3FF
Comp. Val	Reference value used in compare function	0 - 3FF
Comp. Bits	Number of bits to be used in the compare function.	10, 8, 6, 4 bits or 9,7,5,3 bits

Table 13-3Timer parameters values

13.4. Port A in serial mode

The Port A could be defined as a serial port. For a serial mode configuration, the ports have the following attributions

Port	Function		Possible states
In	Serial Data In	**PA0**	Input
Clk	Serial Clock	**PA1**	Output in master mode
			Input in slave mode
Out	Serial Data Out	**PA2**	Output
S	Status - Chip select	**PA3**	Output

Table 13-4 Serial port states

13.5. Supply voltage level detector and ADC mode

Fig. 13-4 SVLD and ADC selection

It is possible to simulate changes in the supply level or pin level of PortA bit 4 which becomes marked ADC (bit 3 address073H). These changes will be operated from 0V to 3.6V with the use of the appropriate scroll bar. The minimum accuracy is 0.01V, although steps of 0.1V can be obtained using the page-up and page down function keys. Reducing the supply voltage under 1.05V would stop the software simulation of the microcontroller as **POR** emulation, Level must be increase above 1.25V / 1.85V as **Power Check Level** emulation. The symbol ☒ indicates that the ADC or the continuous mode is active (bit 1 address 073H for continuous mode) and the symbol ☐ indicates that the ADC or the continuous mode is inactive.

13.6. Option registers

Please refer to the following table for the registers implementation:

Register **RegOscTrimH** @ addr. 117 decimal = 75 hex.

Bit	Name	Reset	R/W	Description
3	RegOscTrim[7]	0	R/W	**Not implemented**
2	RegOscTrim[6]	1	R/W	**Not implemented**
1	RegOscTrim[5]	1	R/W	**Not implemented**
0	RegOscTrim[4]	1	R/W	**Not implemented**

Table 13-5 RegOscTrimH

Register **RegOscTrimL** @ addr. 118 decimal = 76 hex.

Bit	Name	Reset	R/W	Description
3	RegOscTrim[3]	1	R/W	**Not implemented**
2	RegOscTrim[2]	1	R/W	**Not implemented**
1	RegOscTrim[1]	1	R/W	**Not implemented**
0	RegOscTrim[0]	1	R/W	**Not implemented**

Table 13-6 RegOscTrimL

Register **RegMFP0** @ addr. 121 decimal = 79 hex.

Bit	Name	Reset	R/W	Description
3	Opt[3]	0	R/W	**Not implemented**
2	Opt[2]	0	R/W	Must be kept to "0"
1	Opt[1]	0	R/W	Counter Clock source 7 selection "0" PA3/PA4, "1" RCclk/2
0	Opt[0]	0	R/W	**Not implemented**

Table 13-7 RegMFP0

Note1. bit1: When Opt[1]=1 the clock of the counter is no more ck16 (32kHz/50kHZ) but the CPU clock in order to achieve a correct sampling of the RCclk/2 signal.

Register **RegMFP1** @ addr. 122 decimal = 7A hex.

Bit	Name	Reset	R/W	Description
3	Opt[7]	0	R/W	RC base frequency (32/50 kHz) selection.
2	Opt[6]	0	R/W	RC frequency selection bit 2
1	Opt[5]	0	R/W	RC frequency selection bit 1
0	Opt[4]	0	R/W	RC frequency selection bit 0

Table 13-8 RegMFP1

Register **RegMFP2** @ addr. 123 decimal = 7D hex.

Bit	Name	Reset	R/W	Description
3	Opt[11]	?	R/W	
2	Opt[10]	?	R/W	
1	Opt[9]	?	R/W	
0	Opt[8]	0	R/W	ADC/SVLD voltage level#15: "0" 2.75v, "1" 3v

Table 13-9 RegMFP2

13.7. Metal Mask Options

Fig. 13-5 MOpt display

The Metal Mask Options are displayed on the MOpt display, (Periphery interface window). These options can only be modified by editing the .prj file of the project. During the modification of the .prj file, the project and the simulator must be closed.

13.7.1. View of an edited .prj file
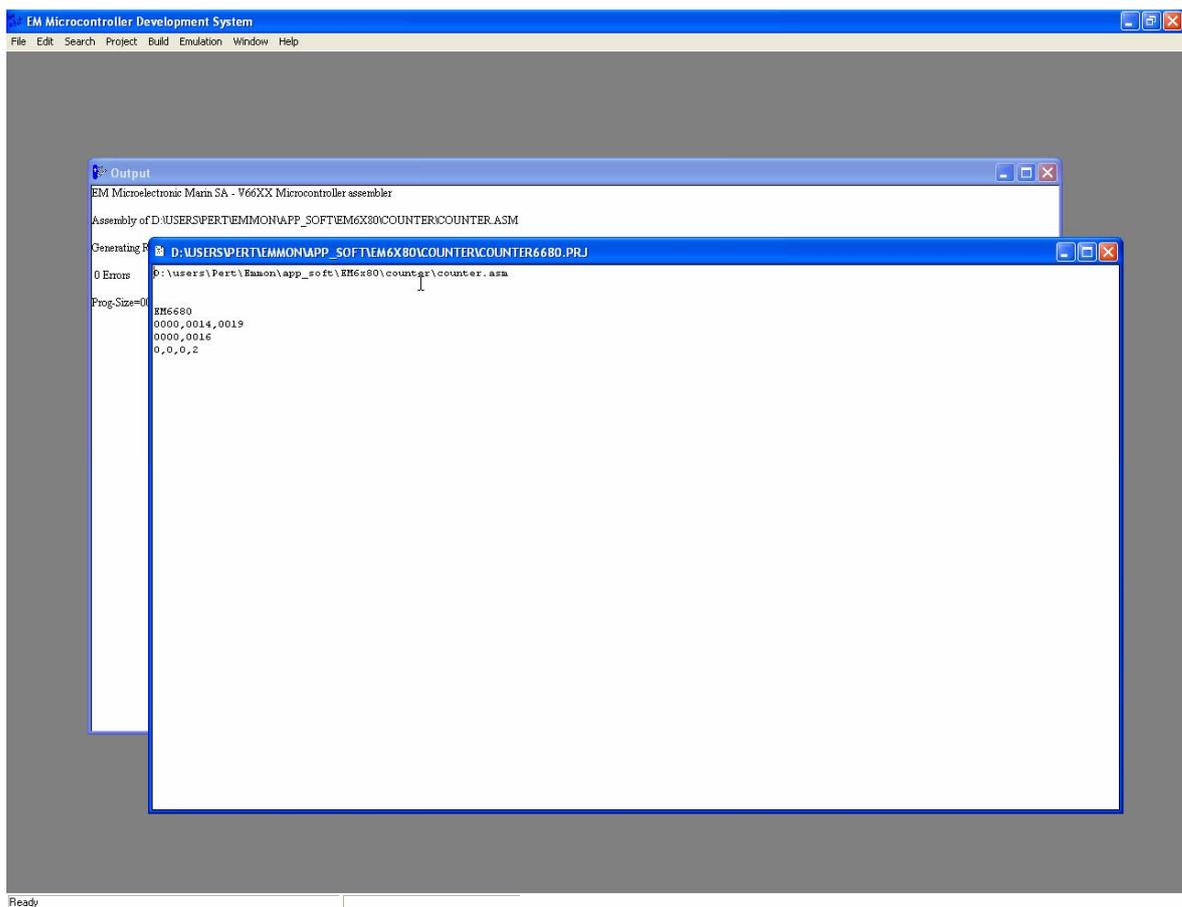
The last line corresponds to the Metal Mask Options; the different options are defined in the following table:

13.7.2. Options table

Bit Field	BIT 7 to 4	Bit 3	Bit 2	Bit 1	Bit 0
MOpt 3	Counter clock update 0 = RC / 2 1 = RC	Power check actif 0 = only at POR 1 = All Reset	SVLD Level selection 0 = EM6680 1 = EM6681	Timer bit 0 don't care 0 = 10 bits counter 1 = 9 bits counter	Power check selection 0 = level 5 as 2.75v 1 = level 9 as 3V

Table 13-10Serial port states

After options definition, the .prj file must be saved and the projects re-opened. All the selected options will be updated into the MOpt display.

13.7.3. Metal mask options example

MOpt3 = 0x11 => Power Check level 9 as 1.85V, counter clock update as RC

EM66xx Microcontroller

LCD Editor Module

CONTENTS

1. LCD Simulation

The application LCDEDIT allows the definition of virtual LCD displays for use with the EM 4bit Microcontroller Development System simulation tools. These virtual LCD displays can be customised to the specific needs of the applications and consequently allows the complete testing of the software with the simulation tools. In addition this tool allows the evaluation of LCD prototypes as well as the development of software before the existence of the actual display.

When simulating a microcontroller that has an LCD the simulation module of the microcontroller searches the current project directory for a LCD configuration file (*.CFG) that has the same name as the project definition file. If this LCD configuration file cannot be found then the default LCD configuration file for the target microcontroller will be used. The LCD configuration file contains the information necessary for the display of the LCD. This includes parameters such as the initial position of the display on the screen, the size of the display and the size and position of each segment within the display. Each defined segment is assigned a segment address and data bit, as well as a bitmap, which defines the active state of the segment. Segment address/data bit and Segment/ backplane (COM) indexes have a one to one mapping (except that segment addresses are zero based while segment indexes are 1 based) for default segment allocation. For free segment allocation, each Segment/backplane index can be allocated freely whitin the μController's LCD address space (refer to μC specification). One bitmap may be assigned to several segments thus reducing the amount of time spent editing the bitmaps. In addition several segments may be assigned identical segment and backplane indexes.

The following document describes how to use the application LCDEDIT to define a virtual LCD display.

2. Defining an LCD configuration

2.1. Defining the LCD size and position

When the LCDEDIT application is started the default configuration is loaded. The default display defines only the initial and size and position of the display in the simulator and doesn't contain any segment definitions. When the LCD frame is the selected object its outline is shown in red. In this state the size and position can be modified by either dragging the display with the left mouse button down using the arrows to position the display and ctrl + arrows to resize the display (see section 3.2 for complete key definitions) .
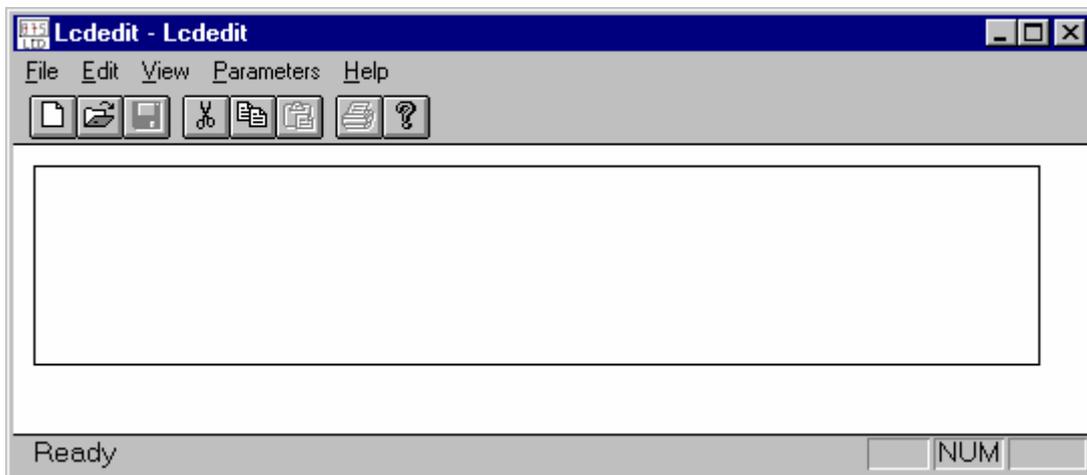
Figure 1 LCD editor view

In addition the size and position of the display can be defined by selecting the general option in the parameters menu as shown below.
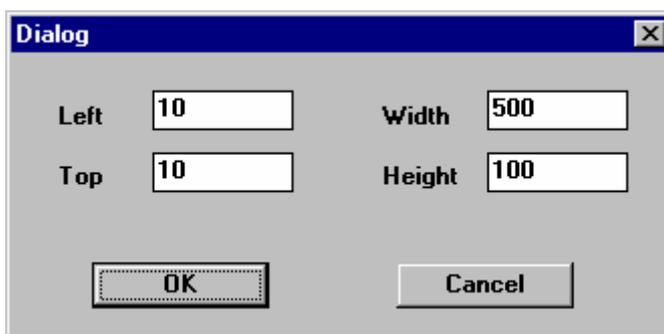
Figure 2. Definition of LCD size and position

2.2. Inserting a new segment

To insert a new LCD segment use the « Insert » key of the keyboard. This inserts a segment at position 0,0 of the display and selects the new segment as the currently active object (outline shown in green). The segment parameters can now be edited by either double clicking on the segment or by using the segments option in the parameters menu.
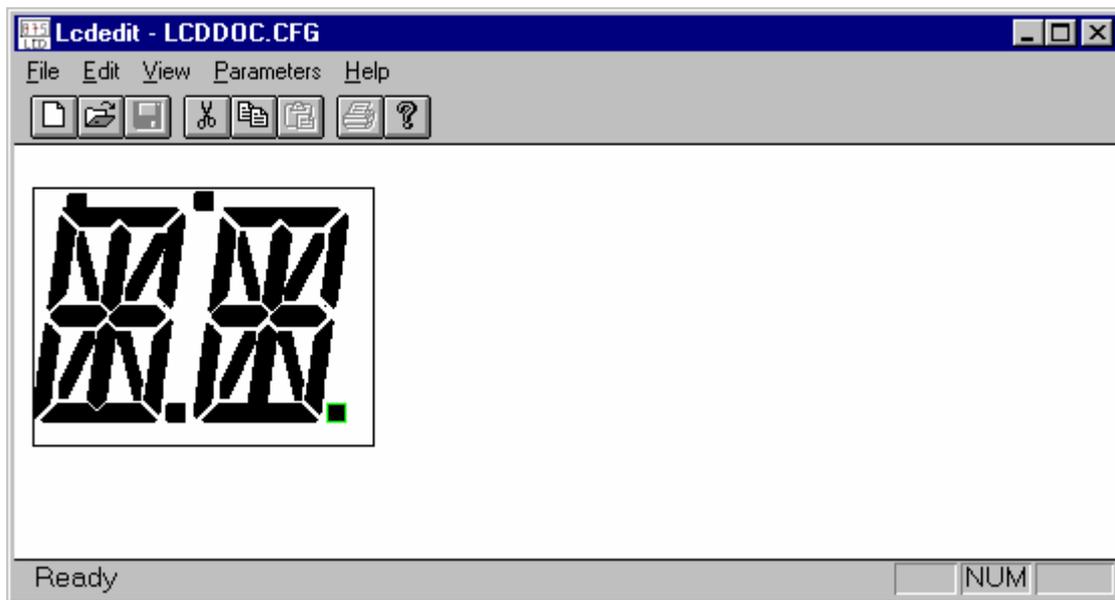
Figure 3 Editor's view with defined segments

2.3. Selecting Segments

A segment can be selected as the current object to be edited by either clicking on the segment with the left mouse button or using the «TAB » key.

2.4. Deleting an existing segment

A segment definition can be deleted by selecting the segment to be deleted with the mouse or the tab key and using the « delete » key. The segment definition as well as the reference to the bitmap is the deleted from the configuration. The bitmap file associated with the segment is however not deleted.

2.5. Editing segment parameters

When a segment is selected as the active object its outline is shown in green. The size and position the selected segment can be modified either with the mouse or by the keyboard as described in section 3.4. All the segment parameters can also be edited by either double clicking on the segment or by using the segments option in the parameters menu. The segment parameters dialogue box is shown below.
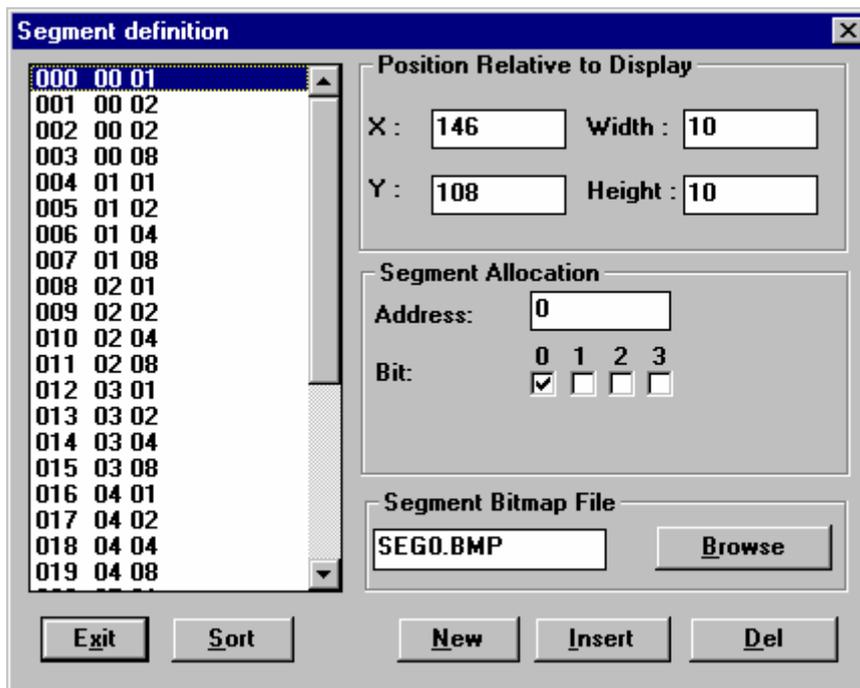
Figure 4 LCD segment parameter definition

The existing segments are displayed in the list box on the left of the dialogue box. The current selected segment is highlighted by default. Once the segment is selected in the listbox its parameters can be modified using the edit fields on the right of the dialogue box. The parameters are the following.

Position parameters

X   The X (horizontal) co-ordinate of the segment in logical screen units (X=0 is the left of the screen).

Y   The Y (vertical) co-ordinate of the segment in logical screen units Y=0 is the top of the screen.

Width   The width in logical screen units of the segment

Height   The height in logical screen units of the segment
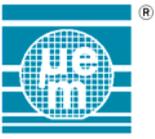
Segment Allocation

Address	The segment address (zero based)
Bit	Select the data bit associated with the segment

Segment address/data bit are used by the program to drive the LCD segment.

Segment Bitmap Enter the filename of the bitmap that will be used to represent the segment. The Browse button can be used to select the file name.

The actual bitmap used to represent the segment can be produced by any graphical editing tool. However, only monochrome or 16 colour bitmaps are supported at present. During display in the simulation tool the bitmap will be either stretched or shrunk to fill the area defined for the segment.

Functions

New	Inserts a new segment
Del	Deletes the currently selected segment
Sort	Sorts the segments in segment/backplane indexes order When using free segment allocation (see below), sorting the segments may completely change the allocation. Versions of the LCD editor prior to 1.3 sorted the segment each time the configuration was saved.
Exit	Exit the dialogue box

2.6.   Segment allocation

From version 1.3 of the LCD editor, it is possible to define free segment allocation.

For each segment/backplane an address/data-bit can be freely defined within the LCD address space to drive the segment. This is done by defining for each segment/backplane an index in the segments defined in the 'Segment definition' dialog.

The following dialog allows the segment allocations.
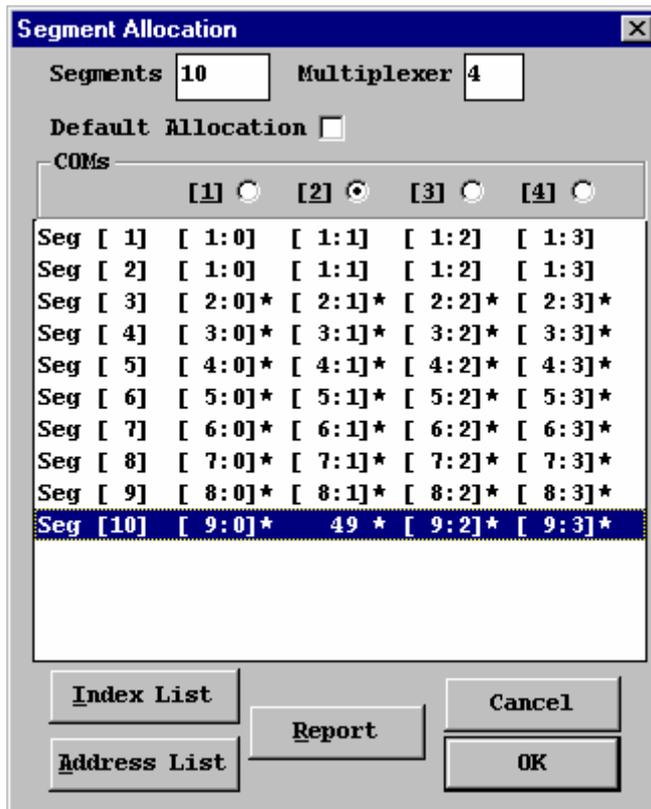
Figure 5 Segment allocation dialog

The number of segments as well as the multiplexer parameters can be defined. The 'Default Allocation' check box is unchecked for free segment allocation. The list box displays for each segment/backplane the address/data-bit which drive the segment. An asterix '*' besides the definition (as in [2:0]*) indicates that the segment/backplane was not allocated an address/data-bit and thus the default allocation is used for this segment/backplane allocation. An index as in ' 49 *' indicates that the index defined for the allocation is no more valid (ie the segment has been deleted).

To define one segment/backplane allocation: chek the appropriate [com] in the COMs group-box and double-click on a Seg [seg] line in the list box and select a segment from the 'Segment definition' dialog which is displayed then dismiss the 'Segment definition' dialog using 'Exit'. At this point the allocation is updated in the list box.

By clicking 'Index list' button, it is possible to display the segment allocation in the index view as follow:
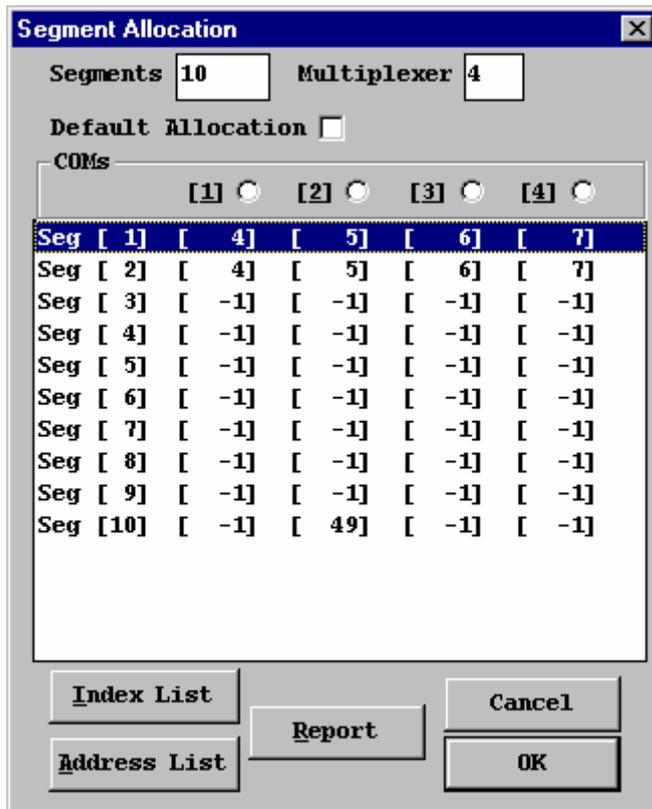
Figure 6 Segment allocation (index view)

A [-1] indicates that the segment/backplane was not allocated an address/data-bit and thus the default allocation is used for this segment/backplane allocation.

2.7. Segment allocation report

It is possible to generate a report on the segment allocation by clicking the Report button. The report is generated and displayed in a 'Segment Allocation Report' dialog, which allows to view the report and to save it. The report is saved in the application directory with filename 'app-name.txt'.

The following listing shows a report for a free segment allocation.

```
10:06:30 02/05/98
Segment allocation report for LCDDOC


Segments:   10
Multiplexer:  4
Free segment allocation

Segment allocation table (addresses)

          COM[0]  COM[1]  COM[2] COM[3]
Seg [ 1] [ 1:2]    [ 1:1] [ 0:2] [ 0:3]
Seg [ 2] [ 1:0]    [ 1:1] [ 1:2] [ 1:3]
Seg [ 3] [ 2:0] [ 2:1] [ 2:2] [ 2:3]
Seg [ 4] [ 3:0]*[ 3:1]*[ 3:2]*[ 3:3]*
Seg [ 5] [ 4:0]*[ 4:1]*[ 4:2]*[ 4:3]*
```

```
Seg [ 6] [ 5:0]*[ 5:1]*[ 5:2]*[ 5:3]*
Seg [ 7] [ 6:0]*[ 6:1]*[ 6:2]*[ 6:3]*
Seg [ 8] [ 7:0]*[ 7:1]*[ 7:2]*[ 7:3]*
Seg [ 9] [ 8:0]*[ 8:1]*[ 8:2]*[ 8:3]*
Seg [10] [ 9:0]*   49 *[ 9:2]*[ 9:3]*
```

Segment allocation table (indexes)

```
           COM[0] COM[1] COM[2] COM[3]
Seg [ 1] [  6]   [  5]   [  2]   [  3]
Seg [ 2] [  4]   [  5]   [  6]   [  7]
Seg [ 3] [  8]   [  9]   [ 10]   [ 11]
Seg [ 4] [ -1]   [ -1]   [ -1]   [ -1]
Seg [ 5] [ -1]   [ -1]   [ -1]   [ -1]
Seg [ 6] [ -1]   [ -1]   [ -1]   [ -1]
Seg [ 7] [ -1]   [ -1]   [ -1]   [ -1]
Seg [ 8] [ -1]   [ -1]   [ -1]   [ -1]
Seg [ 9] [ -1]   [ -1]   [ -1]   [ -1]
Seg [10] [ -1]   [ 49]   [ -1]   [ -1]
```

Segment definition report

Segments: 32

```
Seg   Add:Bit  X / Y   W / H   Bitmap
 0      [ 0:0]  146/108  10/ 10  SEG0.BMP
 1      [ 0:1]  135/ 66  15/ 15  SEG1.BMP
...
 31     [ 7:3]   17/ 3   10/ 10  SEG15.BMP
```

Consistency chek

```
[ 9:1] no segment defined
[ 9:2] no segment defined
[ 9:3] no segment defined
[ 9:4] no segment defined
[10:1] no segment defined
[10:2] index to segment list invalid
[10:3] no segment defined
[10:4] no segment defined

segment index:  0 not used
segment index:  1 not used
```

The consistency check reports are:

'[9:1] not segment defined':
indicates that the address specified for segment:9/com:1 ([8:0] default value) is not defined in any segment.

'[10:2] index to segment list invalid':
indicates that the specified index is not valid (the segment has been deleted).

'segment index: 0 not used':
indicates the segment 0 is not used.

A similar report can be generated for configuration with default allocation.

10:18:15 02/05/98
Segment allocation report for LCDDOC

Segments: 10
Multiplexer: 4
Default segment allocation

Segment allocation table (addresses)

	COM[0]	COM[1]	COM[2]	COM[3]
Seg [1]	[0:0]	[0:1]	[0:2]	[0:3]
Seg [2]	[1:0]	[1:1]	[1:2]	[1:3]
Seg [3]	[2:0]	[2:1]	[2:2]	[2:3]
Seg [4]	[3:0]	[3:1]	[3:2]	[3:3]
Seg [5]	[4:0]	[4:1]	[4:2]	[4:3]
Seg [6]	[5:0]	[5:1]	[5:2]	[5:3]
Seg [7]	[6:0]	[6:1]	[6:2]	[6:3]
Seg [8]	[7:0]	[7:1]	[7:2]	[7:3]
Seg [9]	[8:0]	[8:1]	[8:2]	[8:3]
Seg [10]	[9:0]	[9:1]	[9:2]	[9:3]

Segment definition report

Segments: 32

Seg	Add:Bit	X / Y	W / H	Bitmap
0	[0:0]	146/108	10/ 10	SEG0.BMP
1	[0:1]	135/ 66	15/ 15	SEG1.BMP
...				
31	[7:3]	17/ 3	10/ 10	SEG15.BMP

Consistency chek

[1:2] multiple definitions
        at index: 1
        at index: 2
[1:3] not defined
[9:1] not defined
[9:2] not defined
[9:3] not defined
[9:4] not defined
[10:1] not defined
[10:2] not defined
[10:3] not defined
[10:4] not defined

The consistency check reports are:

'[1:2] multiple definitions
        at index: 1
        at index: 2':
indicates that address/data-bit for segment/com [1:2] is defined for segments with index 1 and 2

'[1:3] not defined':
indicates that the address specified for segment:1/com:3 ([0:2] default value) is not defined in any segment.
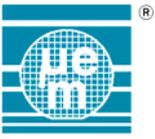
2.8. Editing Advices

Whether you use default or free segment configuration, you should start by defining you segments using the editor view and the 'Segment definition' dialog. After defining the segments, you can proceed to the segment allocation part.

If you use default allocation, you can nevertheless specify the Segments and Multiplexer parameters and generate a report.

If you use free segment allocation you have to define each segment allocation (as defined above), unless the segment can use it's default 'address/data bit'. If you need to add new segment or modify current segment definition be carefull no to sort the segment (as stated above segment allocation use an index to the segments in the 'Segment definition' dialog).

2.9. Saving LCD configurations

Once terminated the LCD configuration can be saved to permanent storage using the Save option in the File menu. An existing definition can also be saved to another name using the « Save As » option in the « File » menu. It should be noted that the configuration file should be saved to the saved directory as the project definition of the application and have the file extension CFG. This does not apply to the bitmap files associated with the segments since the complete file name can be specified in the segment definition parameters.

2.10. Loading LCD configurations

LCD configurations can be loaded for editing from either the most recently used file list in the « File » menu or by using the « Open » option in the file menu. A LCD definition file has the extension *.CFG.

3. Editing Keys

3.1. Changing selection

It is possible to change the selected item by using the TAB key.

3.2. Whole display

When the LCD frame is selected (the outline is shown in red) the following keys can be used to move and resize the display.

Function	Key
Move the display left	Left arrow
Move the display right	Right arrow
Increase the display width	Ctrl + Right arrow
Decrease the display width	Ctrl + Left arrow
Move the display up	Up arrow
Move the display down	Down arrow
Increase the display height	Ctrl + Down arrow
Decrease the display height	Ctrl + Up

3.3. Zoom functions

It is possible to zoom in while editing the display. These functions are available in the view menu or can be achieved using the following keys

Function	Key
Zoom in	Ctrl + numpad plus
Zoom out	Ctrl + numpad minus
Zoom 1:1	Ctrl + 1 key
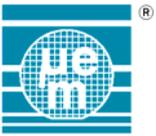
3.4. Segments

When a segment is selected (the outline is shown in green) the following keys can be used to move and resize the segment

Function	Key
Move the segment left	Left arrow
Move the segment right	Right arrow
Increase the segment width	Ctrl + Right arrow
Decrease the segment width	Ctrl + Left arrow
Move the segment up	Up arrow
Move the segment down	Down arrow
Increase the segment height	Ctrl + Down arrow
Decrease the segment height	Ctrl + Up
Go to next segment	TAB

EM MICROELECTRONIC - MARIN SA

A COMPANY OF THE SWATCH GROUP
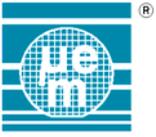
MFP – EM65xx

Programming Interface

Manual

CONTENTS

1. Introduction

The MFP programmer (called PEEPROM, too) is an EEPROM programmer developed and intended for the EM66xx microcontroller family. It has to be connected to a PC and communicates with a telegram-oriented protocol. The MFP immediately interprets each telegram coming from the PC. By means of this protocol, the PC can send orders and control the system (download data's, program device, calculate CRC). First the data have to be downloaded from the PC to the buffer of the programmer. In order to detect if a transmission error occurred, the PC asks the programmer to calculate the CRC (cyclic redundancy checksum) and checks if it fits with its own CRC. After that, the PC can either control the programming cycles or switch the programmer in a stand-alone mode. In this state, the MFP can be disconnected and programming is controlled by means of the programming button. The status LED's indicate what the MFP is doing and if programming cycle has been successfully terminated.

2. Overview

Top View

Socket

Error
Busy
Ok
Standalone
Power

Status LED's

Programming button

Back pannel view

Power supply

Additionnal connector

1 Vbat
2 Vreg
3 Qout
4 Test
5 Qin
6 GROUND

② ④ ⑥
① ③ ⑤

Serial interface
Connection PC

Reset
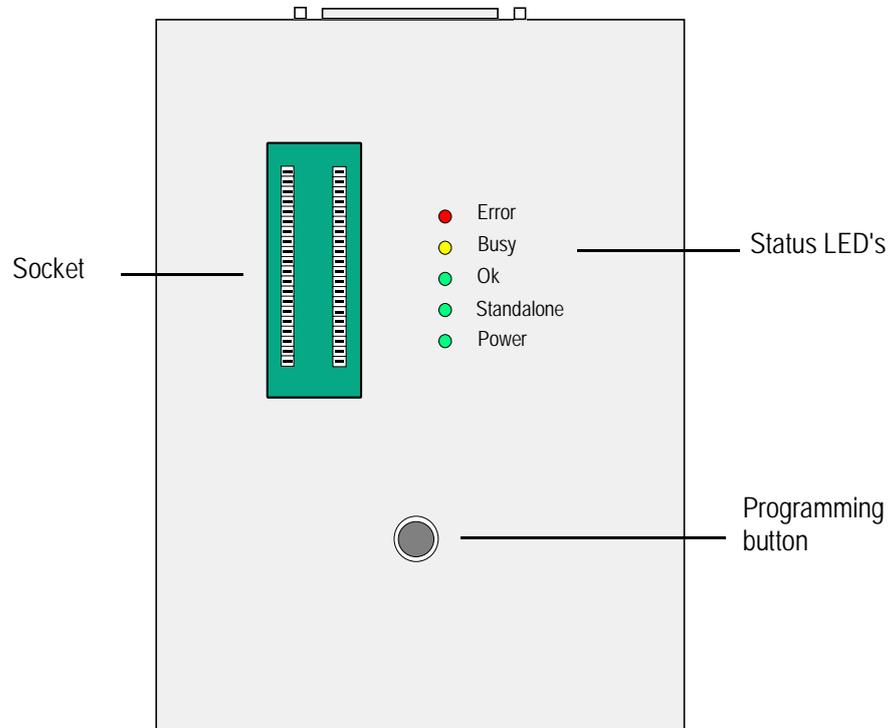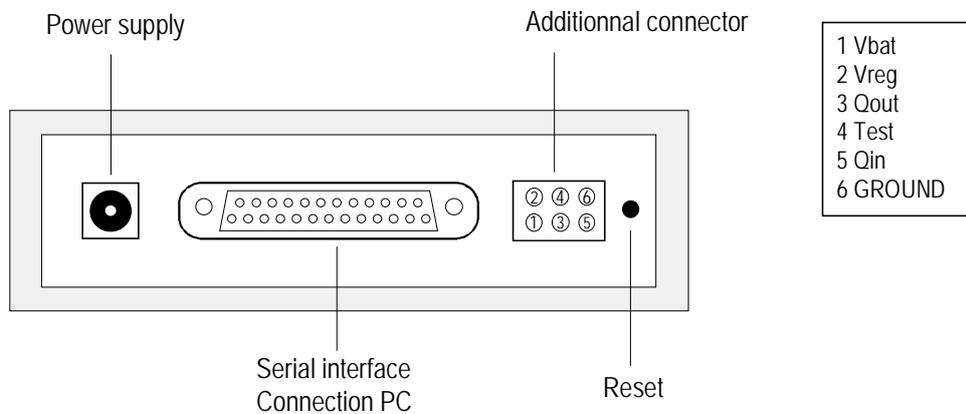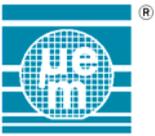
3. Getting Started

1) Connect the programmer to the PC with a point-to-point cable.
2) Connect the AC Adapter (Control that the positive pole is in the center of the connector)
3) Put the device on the socket.

3.1. Programming from the PC

1) Open the MFP Interface Window in the EM microcontroller development system..
2) Set up the parameters in the **Options** menu.

PORT	COM2
BAUD	19200
DTR	X
RTS	X
BUFFERED	OPTIONAL

3) Choose the correct file (**Browse)**.
4) press **Connect**
5) press **Download**
6) If checksum master and checksum programmer are identical press **Program**
7) The programming is correct if the three checksums fit and no errors have been detected.

3.1.1. Status LED's

The busy LED works during each transmission *from PC to programmer* and *from programmer to device.*
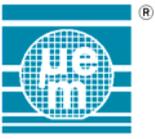
3.2. Programming in standalone mode

1) Open the MFP Interface Window in the EM microcontroller development system.
2) Set up the parameters in the **Options** menu.

PORT	COM2
BAUD	19200
DTR	X
RTS	X
BUFFERED	OPTIONAL

3) Choose the correct file (**Browse)**.
4) press **Connect**
5) press **Download**
6) If checksum master and checksum programmer are identical select **Standalone** in operating mode.
7) The programmer is ready to work alone if the standalone LED's is switch on.
8) Press programming button.

3.2.1. Status LED's

Busy LED :   Works during each transmission *from programmer to device*.
Correct LED : Indicates that the CRC of the master fits with the CRC of the
               device. No error occurred during the programming.
Error LED :   Error occurred during the programming or chip not present.

3.3. MFP programming box with extension cable.

To ensure the correct programming of an MFP μcontroller using the extension cable (either from side connector or from adapter socket), the following schematic has to be used.
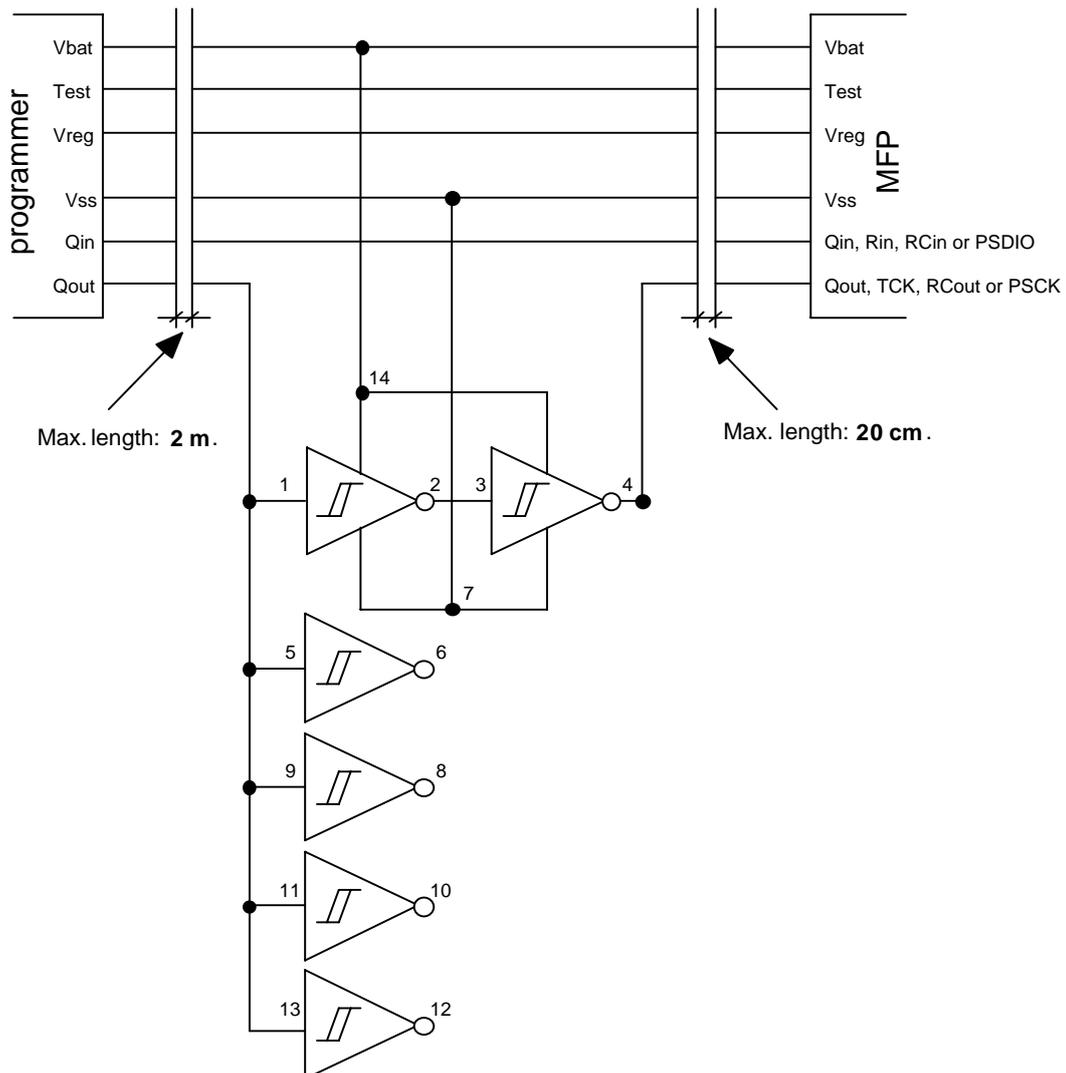
Use the Schmitt Trigger IC : HEF40106B

3.3.1 EMPC65 pinout.
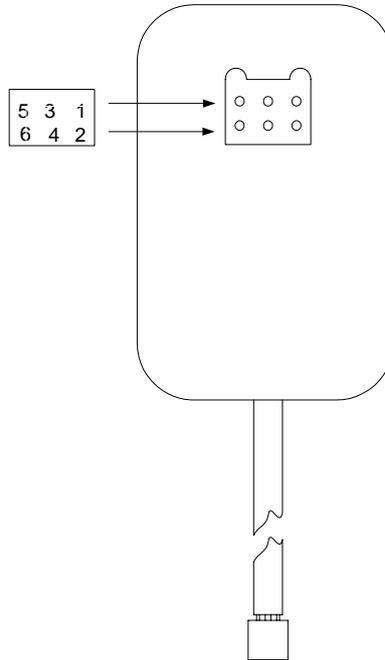
3.4. Some additional Info on MFP programmer.

40 pin DIL socket is for EM6520 only. Use EMPA65XX adapter for other devices.

Pinout for P6520 DIL40:

Vbat	:	39
Vreg	:	40
Vss	:	5
Qin	:	3
Qout	:	1
Test	:	29

Pin used for programming are displayed on last sheet and on additional connector of programmer.

Correspondence for RC oscillators: QIN: RIN, RCIN or PSDIO

                                                       QOUT: TCK, RCOUT or PSCK

3.5. Global Ordering Reference Number

Designator	Reference Numbers for ordering
Programmer or Programming box	EMPB65xx
Programming adaptor for EM6503	EMPA6503 SO24
Programming adaptor for EM6504	EMPA6504 SO24
Programming adaptor for EM6505	EMPA6505 SO24
Programming adaptor for EM6517	EMPA6517 SO28
Programming adaptor for EM6520	EMPA6520 TQFP44
Programming adaptor for EM6521	EMPA6521 TQFP52
Programming adaptor for EM6522	EMPA6522 TQFP64
Programming adaptor for EM6540	EMPA6540 SO18
In Circuit Programming Cable	EMPC65
In-Circuit Emulator for EM66xx	EME6600 (Please also specify the circuit number)
Demo Board for EM6503	EMDB6503
Demo Board for EM6504	EMDB6504
Demo Board for EM6517	EMDB6517
Demo Board for EM6520	EMDB6520
Demo Board for EM6521	EMDB6521
Demo Board for EM6522	EMDB6522
Demo Board for EM6540	EMDB6540
Demo Board for EM6580	EMDB6580

MFP PROGAMMING WIRING SCHEMATIC DIAGRAM

Progamming Box

max. length : 2 meters

max. length : 20 centimeters
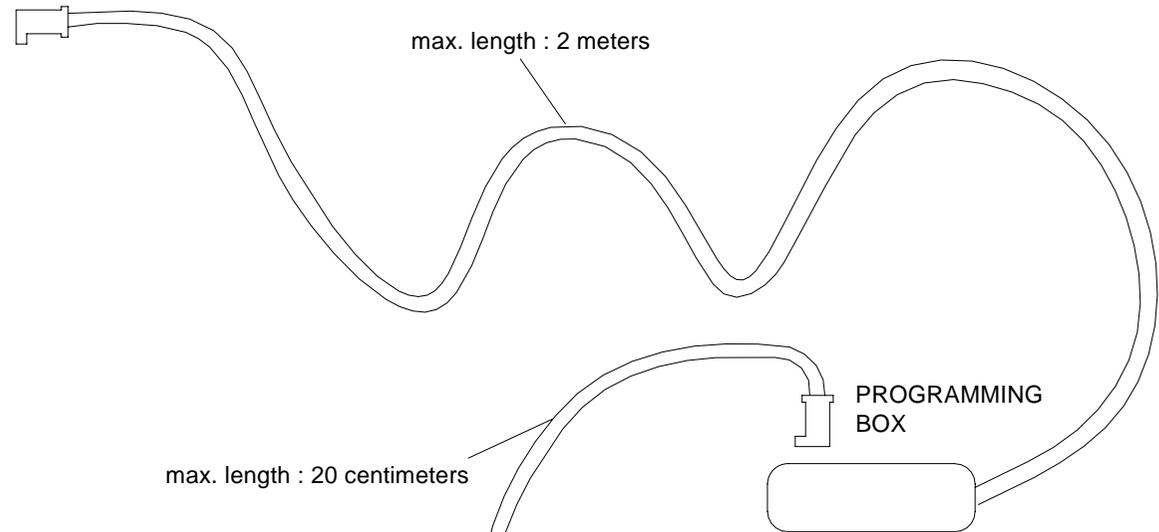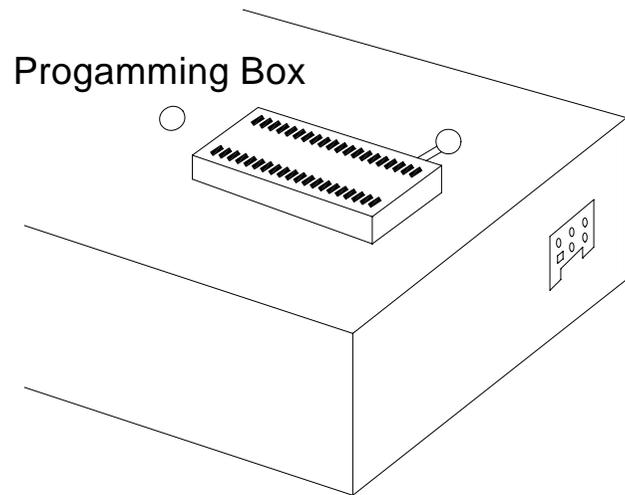
PROGRAMMING
BOX

- The MFP's can be programmed by
  putting it into the socket located on
  the programmer.

- Use the adequate header

- To program in-curcuit, use the socket
  at the rear of the progammer

- Wire the application as shown

Qin   Qout   Test   Vreg

Vdd   *EM65xx*   Vss

your application