BETA Release

Tutorial - Editor - Tools Manual

E-LAB AVRco

Pascal Multi-Tasking for Single Chips

Version for

AVR

Blaise Pascal Mathematician 1623-1662

3-Jul-2009

Autor Rolf Hofmann
Editor Gunter Baab

E-LAB Computers
Grombacherstr. 27
D74906 Bad Rappenau
Tel 07268/9124-0
Fax 07268/9124-24
http://www.e-lab.de
info@e-lab.de

E-LAB
Computers

Mikroprozessor-Technik
Industrie-Elektronik
Hard + Software
8-Bit • 16-Bit • 32-Bit

Important information

Everybody tries to write Software without bugs. The emphasis is on tries, because everybody knows that the more complex a Software is, the more likely it is to produce bugs.

We have the opinion, that this shouldn't have to be norm, and that we do not have to live with the problems and mistakes (although some Software giants think like that ☺).

If you should find any errors, we would be thankful for any information. We will try to solve any problems as quickly as possible.

It is also a normal international agreement that the software producer does not accept liability for any costs arising out of errors in software, unless otherwise agreed.

E-LAB Computers do not accept liability for costs resulting out of errors in the software. It is a condition of use of this Software you agree with these terms. If you do not agree, you are not permitted to use the software.

As we have said, before this exclusion of liability is international standard.

This user guide and the software is intellectual property from E-LAB Computers and therefore copyright protected.

This document and the software it relates to are solely for the use of the purchaser. The purchaser is not permitted to give give, sell or distribute these products. Distributing copies of these products to a third party is strictly prohibited.

We like to think that you as user of the software can make money from it and therefore also expect maintenance of the product. Illegal copies would make it impossible for us to be able to maintain this service.

As you see it is also in the interest of you, the user, to observe the copyright.

That´s it the author

Table of Contents

1 Overview

1.1 AVRco Versions

All AVRco Versions support all AVR Controllers with an internal RAM (for the stack). That means in practice the whole range.

AVRco Profi Version:
The Profi Version contains all available drivers, including very complex ones like e.g. a FAT16 file system and an extensive library for graphic LCDs.
The professional program development is furthermore assisted by the full support of Units.

AVRco Standard Version:
The Standard Version omits only the most complex drivers, and does not support units.

AVRco Demo Version:
The Demo Version supports all controllers and all drivers of the Standard Version.
The **only restriction** is the limitation of the generated code to max. 4 kByte size.

1.2 Manual Versions

Chapters marked with the attribute (*P*) are only available in ***AVRco Profi Revision.***
Chapters marked with the attribute (*4*) are only available in ***AVRco Revision 4.***

1.3 Structure of the Documentation

..\E-Lab\DOCs\DocuCompiler.pdf:
contains the Pascal language description and the enhancements compared with Standard Pascal

..\E-Lab\DOCs\DocuStdDriver.pdf:
contains the description of the drivers contained as well in the Standard, as in the Profi Version.

..E-Lab\DOCs\DocuProfiDriver.pdf:
contains the description of the drivers contained only in the Profi Version.

..E-Lab\DOCs\DocuReference.pdf:
contains a Short Reference (the the same as the online help)

..\E-Lab\DOCs\DocuTools.pdf:
contains the description of the IDE, the simulator, a tutorial etc.

..\ E-LAB\IDE\DataSheets\Release-News.txt:
lists the enhancements in chronological order.
The enhancements are documented in the above mentioned .pdf files (DocuXXX.pdf)

..\E-Lab\AVRco\Demos\ :
contains many test and demo programs

..\E-Lab\DOCs\ :
contains the documentation and further schematics and data sheets

2 Tutorial

by Gunter Baab

2.1 Introduction

All you need to work through this tutorial is the free Demo-Version of the AVRco.
The AVRco allows you to develop applications for all popular AVR controller types.
You can debug your software with the AVRco Simulator and check the behaviour
on a graphical screen (see the screenshots in the chapters concerning the simulator) !

If you want to go beyond the goals of this tutorial to build and program a working
micro controller circuit of yourself you need knowledge of at least four scopes:

1. the programming language Pascal in general and
   the differences, specialities and enhancements of the AVRco Pascal

2. the internal structure of the used controller type

3. the handling of the AVRco package
   to create, compile and debug your program

4. more or less hardware knowledge depending on whether you build the circuit
   and the programming hardware yourself or you buy commercial parts

This tutorial covers only part 3: *the handling of the AVRco*

Other parts are only touched.

The first chapter bescribes the installation the AVRco Demo Version.
The only restriction of the demo is the limitation to a code size of 4k.

Maybe you ask (like me at the very beginning):
OK. 4k code size. But what does that mean to a real application of my own?
What can I do with no more than 4k code size and (the High-Level-Language) Pascal?
Is this enough to scan some switches and control some LEDs – but not more?
Or is it even enough to control a LCD and a matrix keyboard?
What overhead implies the use of such a compiler?

The answer I got was not very helpfull: "*don't care – the compiler is very efficient*".

If you should worry about this limitation have a quick look to the chapter "*Multitasking*"!
This demo uses a LCD, 8 (debounced) switches, a 6-digit 7-segment display, 6 LEDs and the MultiTasking Kernel of the AVRco and takes 85% of the available (flash-) ressources.
So it approuches the limits of the Demo.
But your first applications will seldom need this amount of drivers and every driver enables
you to do complex functions with a single call. This reduces the required space for your
own coding to a minimum (maybe some hundred bytes).

I hope, this gives you a little feeling to recognize the limits of the Demo.
If you start with small applications (maybe 2..3 drivers) the remaining program space of the demo takes at least one weekend (or two) to be filled with reasonable code of your own.

The documentation of the AVRco can be found in **C:\E-Lab\DOCs\DocuCompiler.pdf**.
Especially if you are working through the 2nd example of this tutorial, I strongly recommend to have a printout of this file. There are many cross references to the manual.

The drivers contained in the demo are documented in **C:\E-Lab\DOCs\DocuStdDriver.pdf**.

2.2 Quick Start – Build And Test An Application - A Step-By-Step Introduction

If you are new to the AVRco, you should work through this chapter.
The goal is to create as fast as possible a working environment and give you a rapid impression of what the AVRco is able to do. You do not need to enter a lot of Pascal statements. There is a pre-defined source to be easily inserted via cut & paste.

What is the application for?
8 Keys select various messages to be displayed on a 2x16 LCD.
Keys and LCD are connected to a Mega8 controller.
The hardware is simulated using the AVRco Simulator.

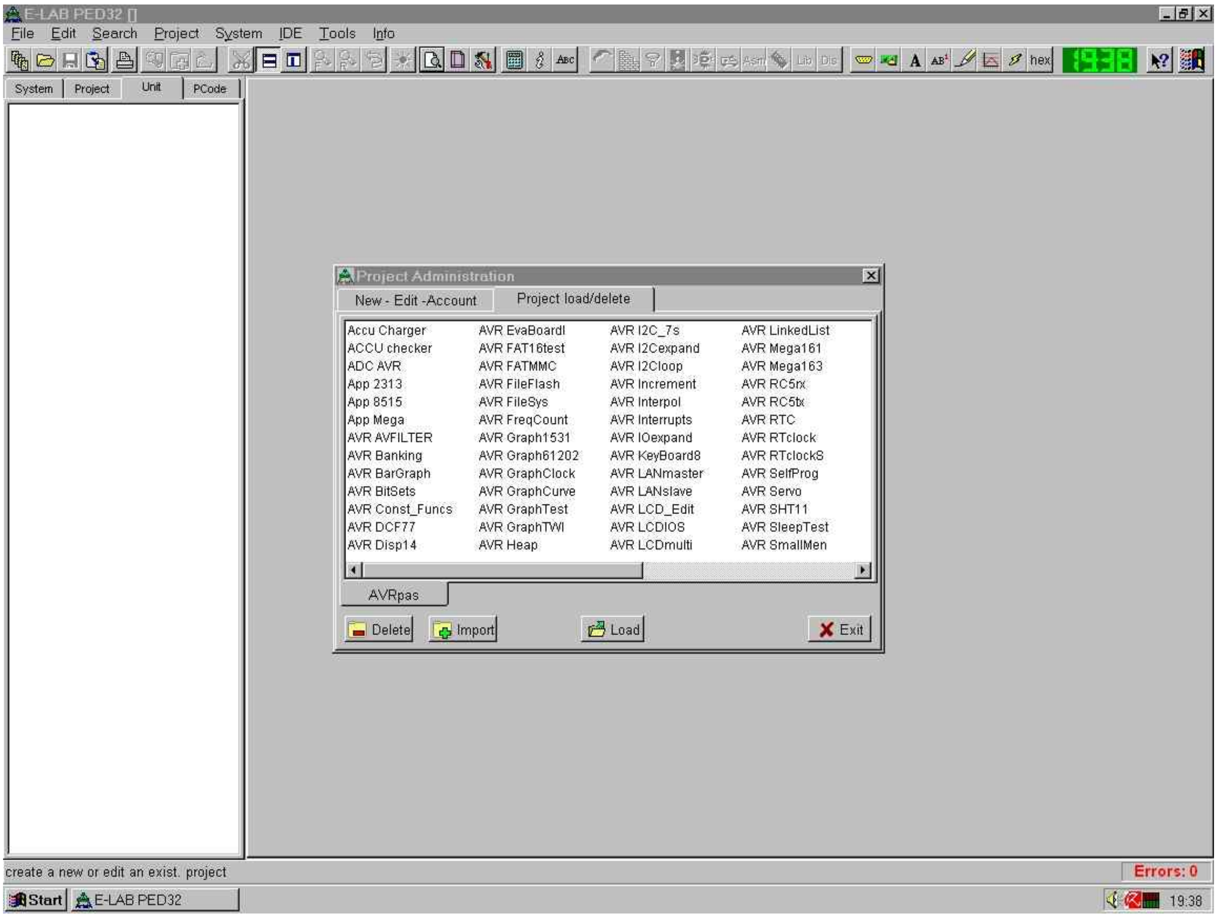
2.2.1 Download, Install And Start The AVRco (Demo Version)

Download the Demo-Version of the AVRco from http://www.e-lab.de and start the AVRdemo.exe.
This self-extracting file will guide you through the installation process, as usual (1,2).

Click *Install* and select the desired language for the documentation.
Do **not** change the default *target directory* from *C:\* to install the compiler to *C:\E-Lab* and click *OK* to create a program group *AVRco* and then *OK* to close the message window.

Close all windows and click *Start – Programs – AVRco – E-LAB PED-32* to start the the IDE (Integrated Development Environment). The PED32 (Programmers Editor for WIN32), that serves as a platform to access the numerous applications of the AVRco (compiler, assembler, simulator and much more) comes up.

notes
(1): do not worry how to get rid of the AVRco. It comes with an un-installer that removes all changes made to your PC.
(2): see the Appendix for the differences/restrictions of the AVRco versions



Accu Charger
ACCU checker
ADC AVR
App 2313
App 8515
App Mega
AVR AVFILTER
AVR Banking
AVR BarGraph
AVR BitSets
AVR Const_Funcs
AVR DCF77
AVR Disp14
AVR EvaBoardI
AVR FAT16test
AVR FATMMC
AVR FileFlash
AVR FileSys
AVR FreqCount
AVR Graph1531
AVR Graph61202
AVR GraphClock
AVR GraphCurve
AVR GraphTest
AVR GraphTWI
AVR Heap
AVR I2C_7s
AVR I2Cexpand
AVR I2Cloop
AVR Increment
AVR Interpol
AVR Interrupts
AVR IOexpand
AVR KeyBoard8
AVR LANmaster
AVR LANslave
AVR LCD_Edit
AVR LCDIOS
AVR LCDmulti
AVR LinkedList
AVR Mega161
AVR Mega163
AVR RC5rx
AVR RC5tx
AVR RTC
AVR RTclock
AVR RTclockS
AVR SelfProg
AVR Servo
AVR SHT11
AVR SleepTest
AVR SmallMen

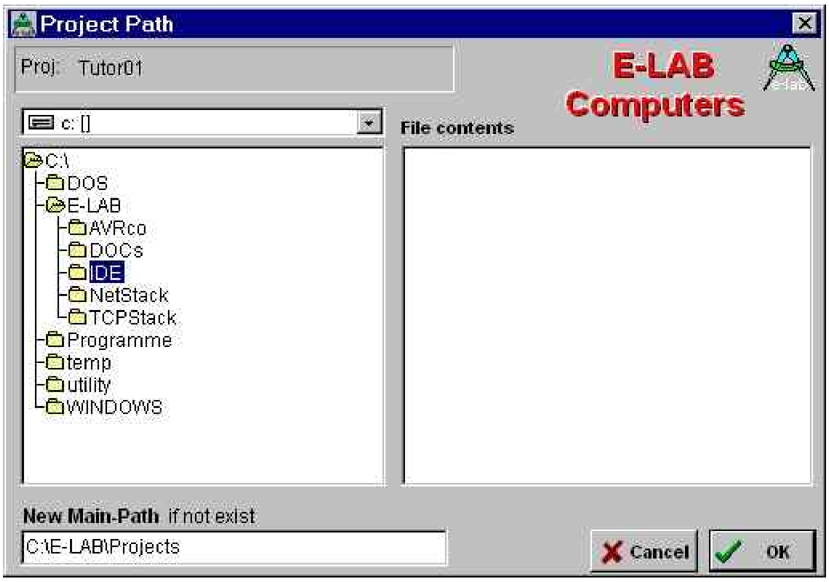
create a new or edit an exist. project

Errors: 0

2.2.2 Create Your First Projekt

Click in the *Project Administration* window the *New – Edit – Account* tab.
In the next window

- enter *Tutor01* in the *Name* field
- click the three dots in front of the *Directory* field: the *Project Path* window comes up
- edit the ***New Main-Path*** *if not exist* field according to the next screenshot

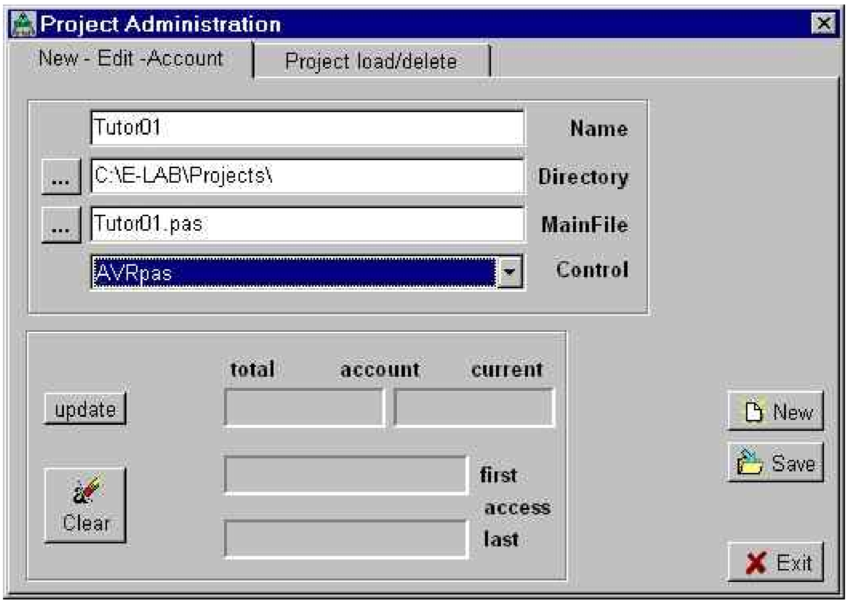


Project Path

Proj: Tutor01

New Main-Path if not exist: C:\E-LAB\Projects

- click *OK* and *Yes* in the *Confirm* window that comes up
- enter the *MainFile* Name and select the *Control*



Project Administration

Name: Tutor01

Directory: C:\E-LAB\Projects\

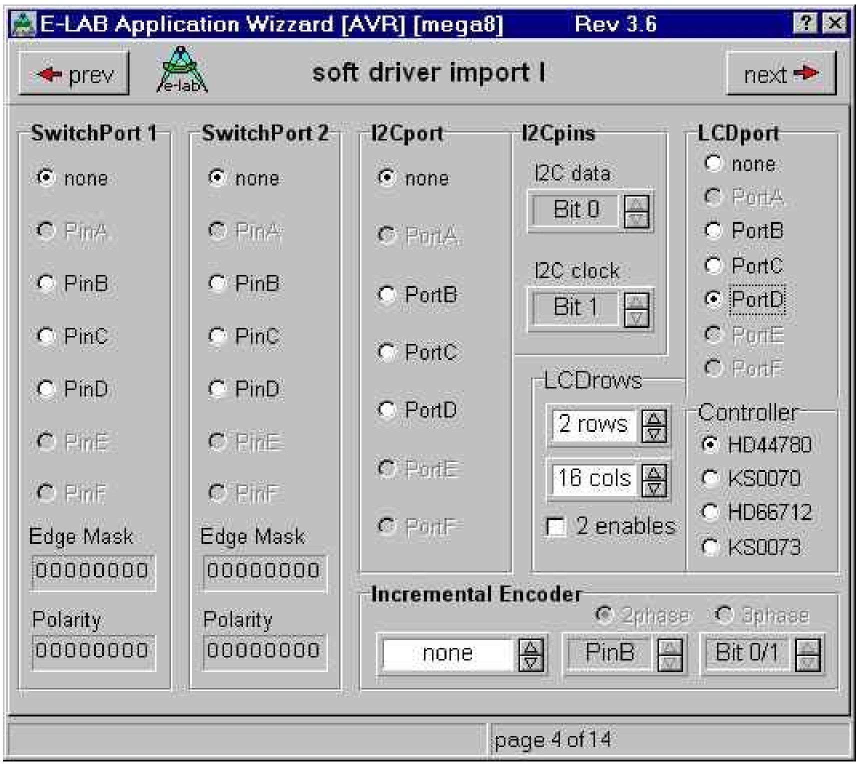
MainFile: Tutor01.pas

Control: AVRpas

- click *Save* and *Exit*

2.2.2.1 Create A Program Frame

- click the speed button *Projekt*
- search (*in the tab Project load/delete*) for the *Tutor01* project, select it and click load
- it will take some time to load the *Application Wizard*
- on the 1st page select *mega8* and a *Frequency* of 8 MHz. Do **not** change other settings
- click repeatedly *next* until you see the *soft driver import* window that contains the *LCDport*
- enable the *LCDport* at *PortD* and chose *2 rows* and 16 columns (*cols*)



E-LAB Application Wizzard [AVR] [mega8] Rev 3.6

soft driver import I

SwitchPort 1: ● none ○ PinA ○ PinB ○ PinC ○ PinD ○ PinE ○ PinF
Edge Mask: 00000000
Polarity: 00000000

SwitchPort 2: ● none ○ PinA ○ PinB ○ PinC ○ PinD ○ PinE ○ PinF
Edge Mask: 00000000
Polarity: 00000000

I2Cport: ● none ○ PortA ○ PortB ○ PortC ○ PortD ○ PortE ○ PortF

I2Cpins: I2C data: Bit 0; I2C clock: Bit 1

LCDport: ○ none ○ PortA ○ PortB ○ PortC ● PortD ○ PortE ○ PortF

LCDrows: 2 rows; 16 cols; □ 2 enables

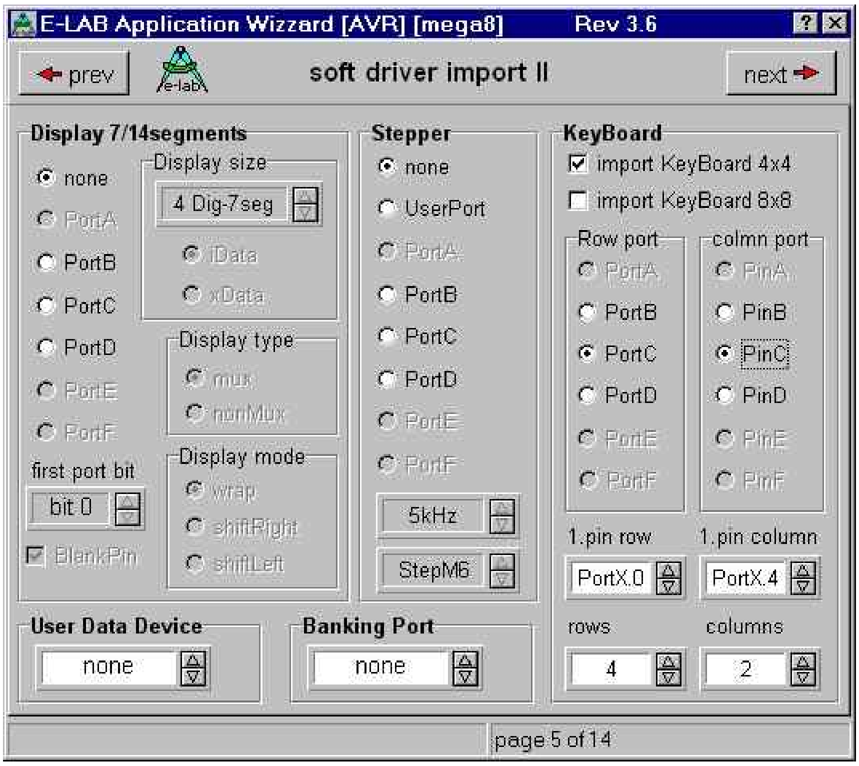
Controller: ● HD44780 ○ KS0070 ○ HD66712 ○ KS0073

Incremental Encoder: none; ● 2phase ○ 3phase; PinB; Bit 0/1

page 4 of 14

note: the page number (above 4 of 14) may be different. It depends on the compiler version.

- click *next* until you see the *soft driver import* window that contains the *KeyBoard* (refer to the screenshot on the next page)
- enable the *import KeyBoard 4x4* check box
- select *PortC* and *PinC* as *Row port* or *column port*
- chose *PortX.0* as *1.pin row* or *PortX.4* as *1.pin column*
- chose 4 rows and 2 columns



E-LAB Application Wizzard [AVR] [mega8] Rev 3.6

soft driver import II

Display 7/14segments: ● none; ○ PortA; ○ PortB; ○ PortC; ○ PortD; ○ PortE; ○ PortF
first port bit: bit 0
☑ BlankPin
Display size: 4 Dig-7seg; ● iData; ○ xData
Display type: ● mux; ○ nonMux
Display mode: ● wrap; ○ shiftRight; ○ shiftLeft

Stepper: ● none; ○ UserPort; ○ PortA; ○ PortB; ○ PortC; ○ PortD; ○ PortE; ○ PortF
5kHz
StepM6

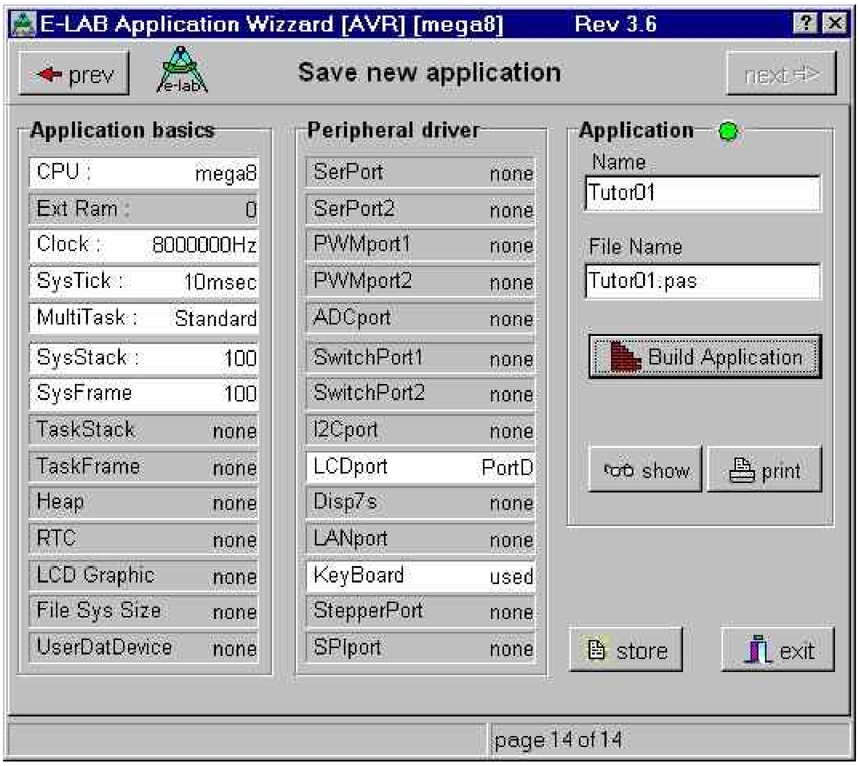
KeyBoard
☑ import KeyBoard 4x4
□ import KeyBoard 8x8
Row port: ○ PortA; ○ PortB; ● PortC; ○ PortD; ○ PortE; ○ PortF
colmn port: ○ PinA; ○ PinB; ● PinC; ○ PinD; ○ PinE; ○ PinF
1.pin row: PortX.0
1.pin column: PortX.4
rows: 4
columns: 2

User Data Device: none
Banking Port: none

page 5 of 14

note: the page number (above 5 of 14) may be different. It depends on the compiler version.

- click repeatedly *next* until the last window



E-LAB Application Wizzard [AVR] [mega8] Rev 3.6

Save new application

Application basics

CPU :	mega8
Ext Ram :	0
Clock :	8000000Hz
SysTick :	10msec
MultiTask :	Standard
SysStack :	100
SysFrame	100
TaskStack	none
TaskFrame	none
Heap	none
RTC	none
LCD Graphic	none
File Sys Size	none
UserDatDevice	none

Peripheral driver

SerPort	none
SerPort2	none
PWMport1	none
PWMport2	none
ADCport	none
SwitchPort1	none
SwitchPort2	none
I2Cport	none
LCDport	PortD
Disp7s	none
LANport	none
KeyBoard	used
StepperPort	none
SPIport	none

Application
Name: Tutor01
File Name: Tutor01.pas

page 14 of 14

note: the page number (above 14 of 14) may be different. It depends on the compiler version.

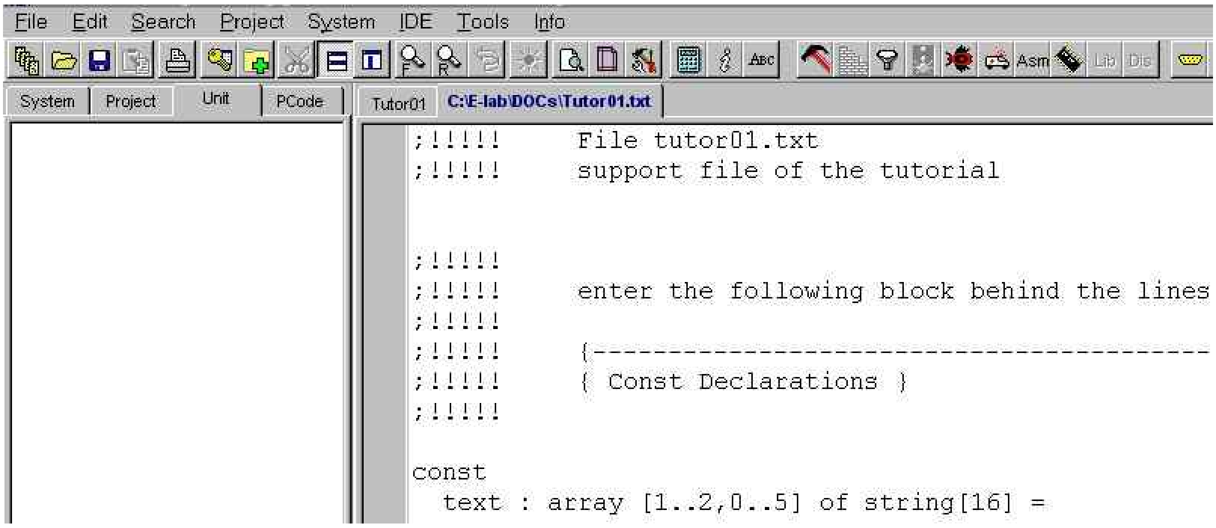
- click *Build Application*, *store* and finally *exit*
- the *Project Administration* is closed an the created frame is loaded into the editor

You should see now the program frame:

```
program Tutor01;

{ $BOOTRST $00C00}         {Reset Jump to $00C00}
{$NOSHADOW}
{ $W+ Warnings}                     {Warnings off}

Device = mega8, VCC=5;

Import SysTick, LCDport, MatrixPort;

From System Import ;

Define
        ProcClock   = 8000000;        {Hertz}
        SysTick     = 10;             {msec}
        StackSize   = $0064, iData;
        FrameSize   = $0064, iData;
        LCDport     = PortD;
        LCDtype     = 44780;
        LCDrows     = 2;              {rows}
        LCDcolumns  = 16;             {columns per line}
        MatrixRow   = PortC, 0;       {use PortC, start with bit0}
        MatrixCol   = PinC, 4;        {use PinC, start with bit4}
        MatrixType  = 4, 2;           {4 Rows at PortC, 2 Columns at PinC}

Implementation

{$IDATA}

{--------------------------------------------------------------}
{ Type Declarations }

type


{--------------------------------------------------------------}
{ Const Declarations }


{--------------------------------------------------------------}
{ Var Declarations }
{$IDATA}


{--------------------------------------------------------------}
{ functions }



{--------------------------------------------------------------}
{ Main Program }
{$IDATA}

begin


  EnableInts;
  loop


  endloop;
end Tutor01.
```

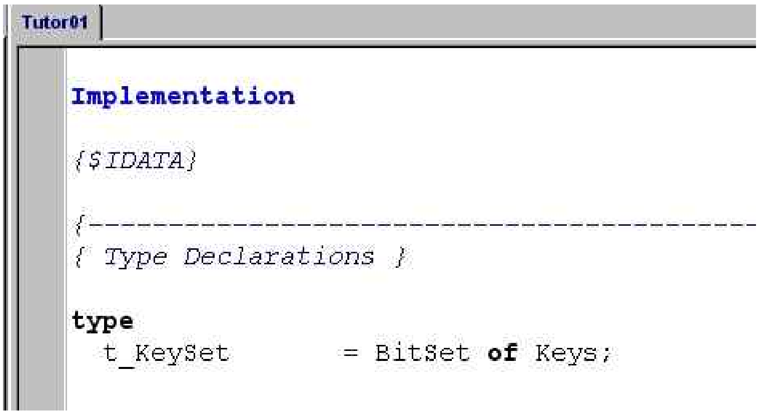
2.2.2.2 Enter The Program

- for your convenience there is a .txt file, that contains the program
- click the *Open* speed button
- change the default file type from *PASCAL/ASM Source* to *Text*
- search the file *tutor01.txt* in *C:\E-Lab\DOCs* and open it


File Edit Search Project System IDE Tools Info
Asm
Lib
Dis
System
Project
Unit
PCode
Tutor01
C:\E-lab\DOCs\Tutor01.txt

```
;!!!!!     File tutor01.txt
;!!!!!     support file of the tutorial


;!!!!!
;!!!!!     enter the following block behind the lines
;!!!!!
;!!!!!     {----------------------------------------
;!!!!!     { Const Declarations }
;!!!!!

const
  text : array [1..2,0..5] of string[16] =
```

- the text file is loaded in a 2nd window
- note the 2 tabs *Tutor01* (the program frame) and *C:\E-Lab\DOCs\Tutor01.txt*
- mark the block starting with *const* (to the end-marker) and press *Ctrl+c* (copy)
- note the location where to copy this block (the info above the block)
- click the *Tutor01* tab, to open the program frame
- position the cursor to the destination line (in the*Tutor01* file) and enter *Ctrl+v* (paste)
- return to the text file and copy the 2nd and 3rd block to the correct locations
- **right**-click the *C:\E-Lab\DOCs\Tutor01.txt* tab and select *close file*
- complete the *type* section according the following screenshot
  (enter the line starting with *t_KeySet*)
- note: the "***of"*** is a keyword. The editor shows keywords automatically in **bold** style
- make sure to enter the line exactly as shown and do not forget the semicolon ";"
  (the number of blanks does not matter)



```
Implementation

{$IDATA}

{-------------------------------------------
{ Type Declarations }

type
  t_KeySet        = BitSet of Keys;
```

2.2.2.3 Compile And Assemble ("Make") The Program

- click the *make* speed button 🔨 to compile and assemble the program
- if you hear, after a short while, a beep and see the sand-glass disappear, your program was succesfully compiled and assembled. **Continue with the next chapter!**

- if you hear a "chord" and an error window comes up, there is still something wrong

  <u>if you got errors:</u>

- click *OK* to close the error window

- do **not** try to ignore an error. The compiler/assembler has not yet created the necessary files to continue

- look at the (first) yellow highlighted line. The (first) error is in this line <u>or above</u>.
  Note that the compiler is **not** always able to locate the **exact** position of an error.
  Once more: **the error is frequently located one ore more lines above!**

- do **not** try to correct several errors at the same time!
  It is not unusual, that one error produces several error messages

- very often there are spelling errors or forgotten semicolons

- try –step by step- to correct the error(s) and "make" the program again

- if you do not succeed print the source file 🖨 and the *tutor.txt* file.
  Check the blocks and their location

- **you should absolutely try to correct the errors by yourself!**
  Even if this takes a lot of tries and some time!
  This is a typical challange of software development and you need these experiences for your own projects

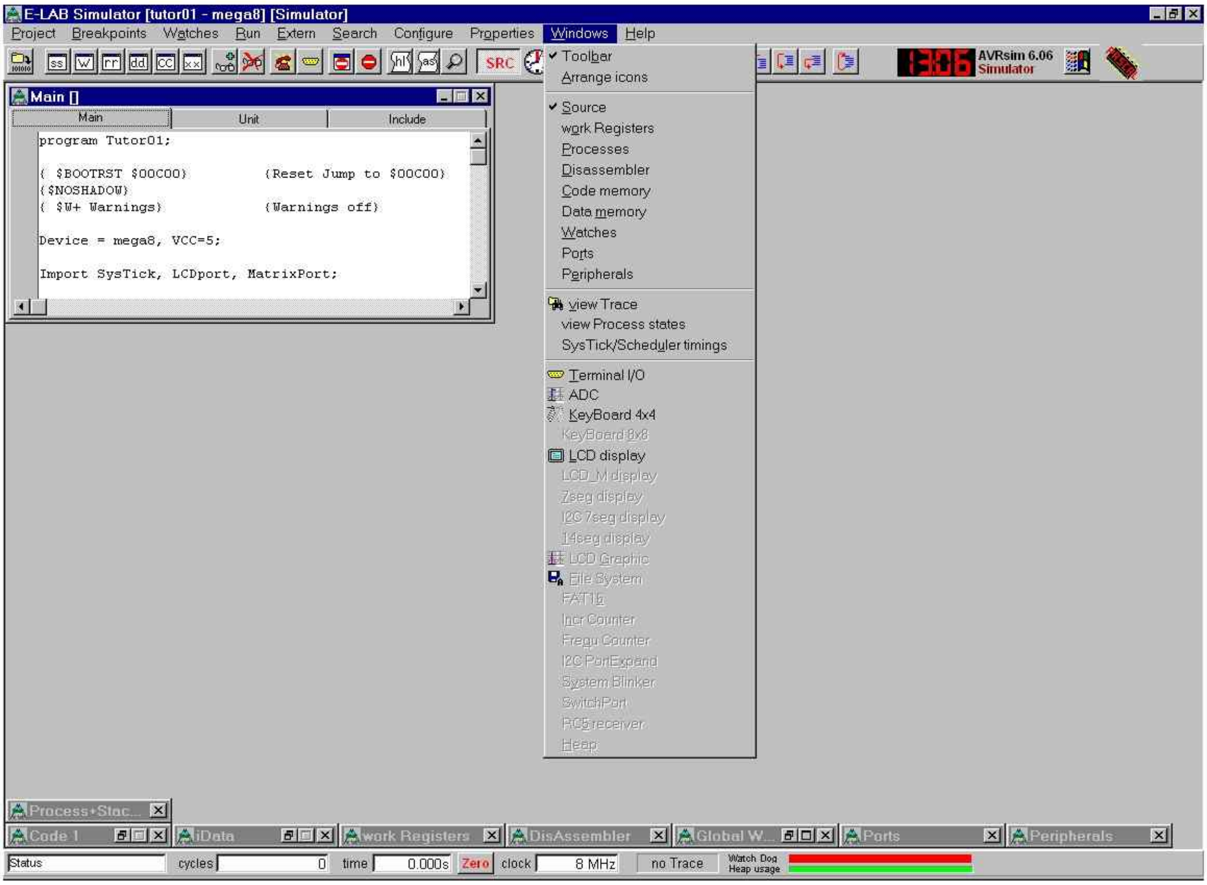
when all else fails:
make a printout of the "*tutor01.pas*" part of the appendix and compare it with your source. Take special care on the blocks that you copied:

- are these blocks complete ?
- are they at the correct location?
- is the "type" block OK?

the last rescue
delete the whole content of the source file and insert a copy of the corresponding part of the appendix via cut&paste

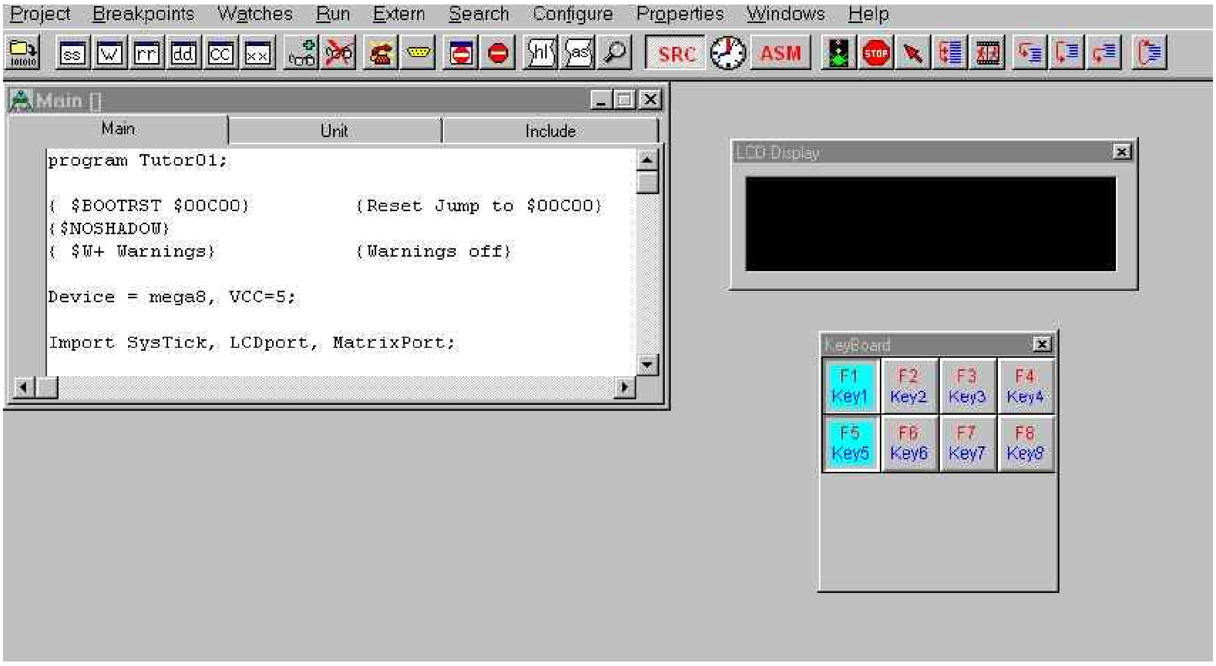
2.2.3 Check The Program Using The Simulator

- start the simulator with the speed button
- the simulator comes up with a lot of open windows.
  As we do not want to debug the program (in this example), we do not need any of these windows
- close all windows exept *Main[]* that contains the source code you already know
- press *Alt+w* to open the *Windows* menu
- your screen should look like the next screenshot



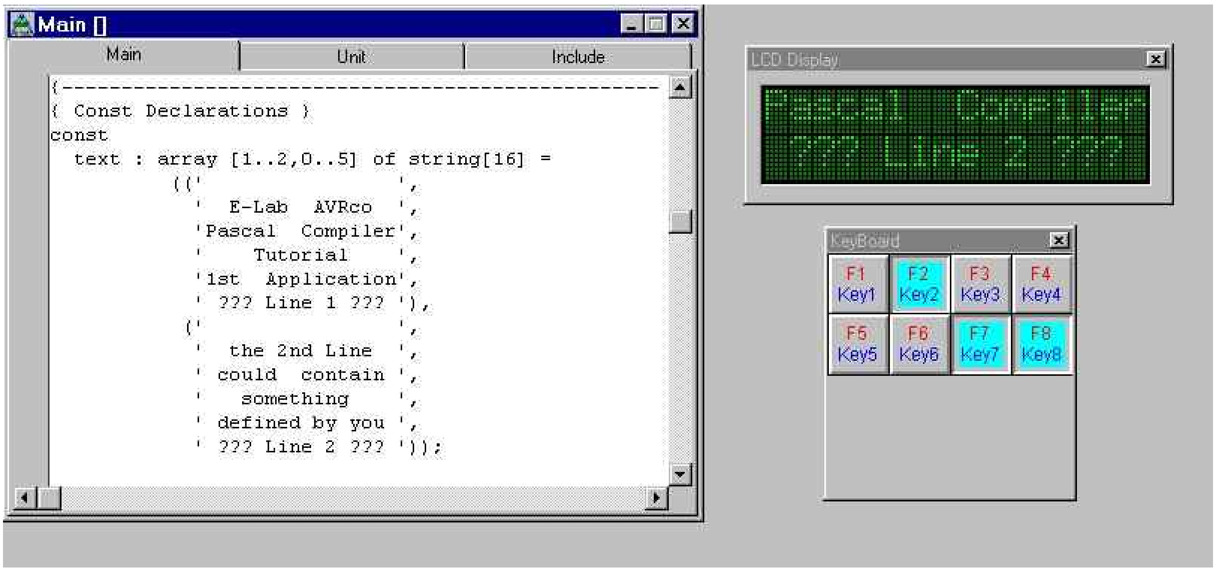
```
program Tutor01;

{ $BOOTRST $00C00}          {Reset Jump to $00C00}
{$NOSHADOW}
{ $W+ Warnings}             {Warnings off}

Device = mega8, VCC=5;

Import SysTick, LCDport, MatrixPort;
```

- select the *KeyBoard 4x4*
- press *Alt+w* again and select the *LCD Display*
- position the windows according the next screenshot
- click some keys and notice how to switch one or more on and off
- switch *F1/Key1* and *F5/Key5* on, all others off



```
program Tutor01;

{ $BOOTRST $00C00}         {Reset Jump to $00C00}
{$NOSHADOW}
{ $W+ Warnings}            {Warnings off}

Device = mega8, VCC=5;

Import SysTick, LCDport, MatrixPort;
```

- press the *run* key and watch the LCD simulation
- switch both keys (*Key1* and *Key5*) off and try the other keys (only one active key per row!)
- look what happens if more then one key per row is depressed
- increase the size of the *Main[]* window and scroll to the *Const Declarations*
- play with the keys and note the relationship between the keys, the display and the source file



```
{--------------------------------------------------
{ Const Declarations }
const
  text : array [1..2,0..5] of string[16] =
          (('                ',
            '  E-Lab  AVRco  ',
            'Pascal  Compiler',
            '    Tutorial    ',
            '1st  Application',
            ' ??? Line 1 ??? '),
           ('                ',
            '  the 2nd Line  ',
            ' could  contain ',
            '   something    ',
            ' defined by you ',
            ' ??? Line 2 ??? '));
```

```
Pascal  Compiler
 ??? Line 2 ???
```

2.2.4 Enter Some Own Messages

- close the simulator and you are back in the editor
- scroll to the definitions of the 2nd display line starting with

```
  ('                ',
   '  the 2nd Line  ',
```

- note:
  - each message is enclosed in apostrophs: **'** xxxxxx **'**
  - the first message contains 16 blanks and is used to clear the display line
  - the following 4 messages relate to *Key5* ... *Key8*
  - the last message is displayed if more then one key per line is depressed
  - the length of all messages is exactly 16 characters to make sure, that a long message is completely overwritten by a short one

- edit **one** line and enter a message of your own
- **still be careful ! Do not change too much !**
- I recommend to keep the fixed line length of 16 for this try
- "make" the program as explained above and correct any errors
- start the simulator again and run it (press F9)
- note that the simulator remembers the number, size and position of your windows

- return to the editor, enter an intentional error (e.g. "forget" (delete) an apostrophe) and compile the program to see how the compiler displays error messages
- correct the error and compile again

- mix shorter and longer messages (than 16) and check the behaviour of the program

- if you end up with errors, that you are not (yet) able to correct, see above: "*if you got errors*"

2.3 Build An Application Of Your Own – Take a Deeper Look

This chapter should encourage you to program and verify a very simple
application for a real hardware of your own. The main intention is to demonstrate
how to create new projects and use some important functions of the IDE.
Furthermore, you should get a deeper understanding of the Simulator features
that help you to find logical erors in your software.

What is the application for?
4 LEDs are used to display several patterns. The patterns are selected with a switch.
The switch and the LEDs are connected to a Mega8 controller: the switch is connected
to PortD-0, the LEDs to PortC-0 ... PortC-3.
To minimize the hardware requirements, the internal oszillator of the Mega8 is used (at 1 MHz).
This is the default setting of a new controller and you do not need to change any
fuse bytes. (1)
In this chapter the hardware is simulated using the AVRco Simulator.
In following chapters you will find all you need to build a real hardware and download the
program to it.

2.3.1 Create A New Project And A Program Frame

If you are insecure about the proceeding refer to the correspondig parts above and have a look at the screenshots!

Start the PED32.exe. Note that the editor remembers your last project and open file(s)
and comes up with Tutor01 loaded

There are several ways to create a new project:

e.g. the speed button *project*
the menu *File – New Project*
the menu *Project – New Project*
a right-click on the tab *Tutor01* and selection of *Load Project*
and probably more ...

No matter what way you use – the *Project Administration* comes up (2)

in the tab *New – Edit - Account* window enter

Name:	Tutor02
Directory:	C:\E-LAB\Projects\
MainFile:	Tutor02.pas
Control:	AVRpas (3)

Click *save* and *exit* to start the *Application Wizard*

notes:
(1): for more informations about the fuse bytes download the Mega8 manual (see links near the end of the tutorial). Be careful and do not play around with the fuse bytes! There are some bits that lock the controller and prevent a further programming!
(2): use the *Project Administration* whenever you want to load / create / delete / import a project.
(3): for more informations about "Controls" and "Projects" see the chapter *"PED32"* !

The *Application Wizard* is used to create a basic program frame for your Pascal source file. There are a number of pages that allow you to specify your system and select the desired features and options like the used data types, the on chip options and additional drivers you want to use. Furthermore, you may define the I/O-ports you need.
The more informations you specify the more complete is the created frame. The wizard creates the statements to import the drivers, the necessary defines, the initalization of the ports and a location for the main program.

Important:

use always the *Project Administration* to create a new application!
You can not create a new application by starting the *Application Wizard*!

As far as the AVRco can not find the main program file it starts the Wizard automatically.
A manual start of the Wizard is only for special purposes and usually never needed.

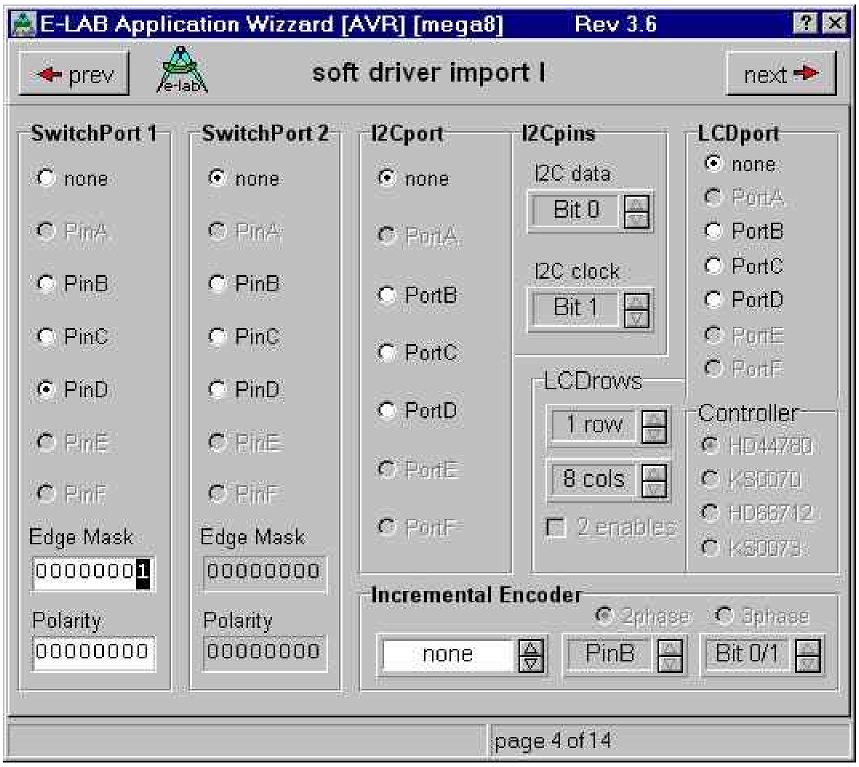
In this tutorial it is by far not possible to cover all items. I will describe only the needed parts. All features are by default disabled. So you can concentrate on the needed ones and enable them selectively.
Do not worry if you forgot something to specify or made a mistake:
the wizard is "only" an aid and you can add, delete and modify the concerning parts later with the editor.

Select the *mega8* and a *Frequency* of *1 MHz*. Leave the other settings at default (1)
click next.

The following windows depend on the compiler version: the functions of the AVRco are constantly extended and the number of pages increases. Actually the 2nd window is *"System and Types Import"*. These items are explained in the AVRco Manual.
The 3rd window is *"OnChip driver import I"*. At the time all "OnCip drivers" fit on one page. Depending on the the future development of the contollers and the available drivers there will –maybe- soon a further page *"OnChip driver import II"* be inserted. These items are covered in the AVRco- and the controller manual.

Do the next input in the *soft driver import* page containing the *Switch Port 1:*
select PinD and edit the Edge Mask according to the following screenshot

notes:
(1): some options are grayed out, depending on the builtin features of the selected processor type and the selected options. This is also true for the following pages



E-LAB Application Wizzard [AVR] [mega8] Rev 3.6

soft driver import I

SwitchPort 1: ○ none, ○ PinA, ○ PinB, ○ PinC, ● PinD, ○ PinE, ○ PinF
Edge Mask 00000001
Polarity 00000000

SwitchPort 2: ● none, ○ PinA, ○ PinB, ○ PinC, ○ PinD, ○ PinE, ○ PinF
Edge Mask 00000000
Polarity 00000000

I2Cport: ● none, ○ PortA, ○ PortB, ○ PortC, ○ PortD, ○ PortE, ○ PortF

I2Cpins: I2C data Bit 0; I2C clock Bit 1

LCDrows: 1 row; 8 cols; □ 2 enables

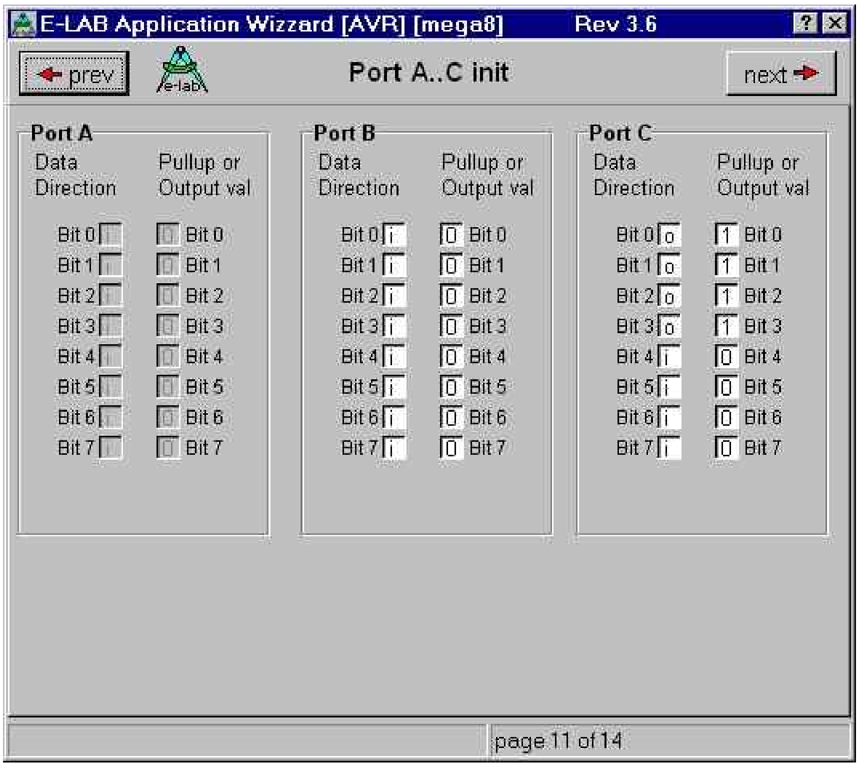
LCDport: ● none, ○ PortA, ○ PortB, ○ PortC, ○ PortD, ○ PortE, ○ PortF

Controller: ● HD44780, ○ KS0070, ○ HD66712, ○ KS0073

Incremental Encoder: ● 2phase ○ 3phase; none; PinB; Bit 0/1

page 4 of 14

The next pages are called *special driver import.* The description of these drivers can be found in the *"DocuStdDriver"* manual
click repeatedly *next* until the *Port A..C init* window
change the *Data Direction* and *Pullup or Output val* of *Bit 0* to *Bit 3* of *Port C*



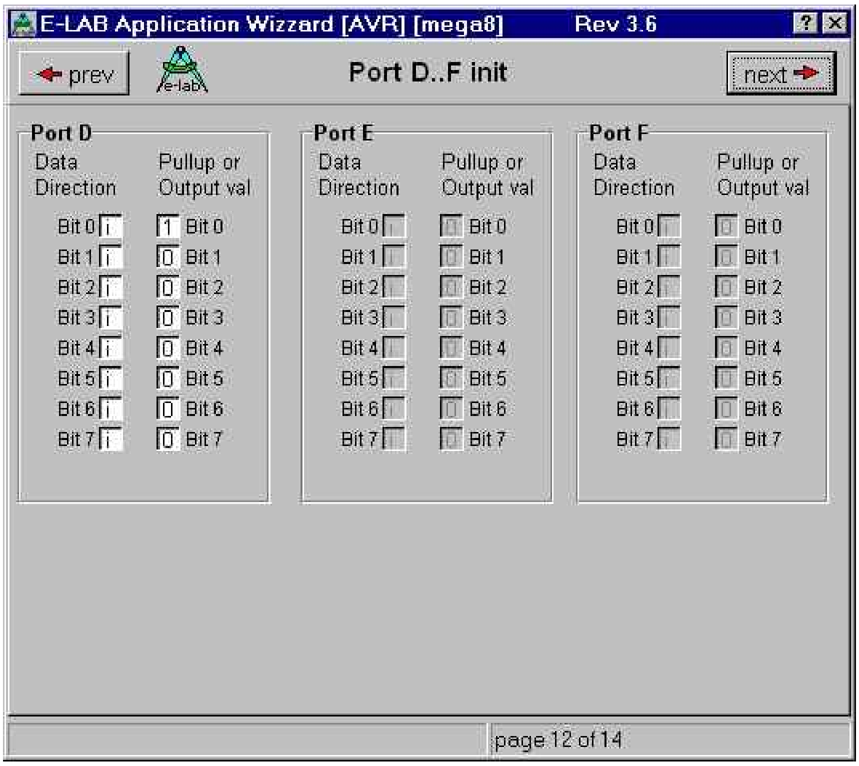
E-LAB Application Wizzard [AVR] [mega8] Rev 3.6

Port A..C init

	Port A Data Direction	Port A Pullup or Output val	Port B Data Direction	Port B Pullup or Output val	Port C Data Direction	Port C Pullup or Output val
Bit 0			i	0	o	1
Bit 1			i	0	o	1
Bit 2			i	0	o	1
Bit 3			i	0	o	1
Bit 4			i	0	i	0
Bit 5			i	0	i	0
Bit 6			i	0	i	0
Bit 7			i	0	i	0

page 11 of 14

click *next* to the *Port D..F init* window and edit *Port D Bit 0* as follows



Port D

Data Direction		Pullup or Output val	
Bit 0	i	1	Bit 0
Bit 1	i	0	Bit 1
Bit 2	i	0	Bit 2
Bit 3	i	0	Bit 3
Bit 4	i	0	Bit 4
Bit 5	i	0	Bit 5
Bit 6	i	0	Bit 6
Bit 7	i	0	Bit 7

Port E

Data Direction		Pullup or Output val	
Bit 0		0	Bit 0
Bit 1		0	Bit 1
Bit 2		0	Bit 2
Bit 3		0	Bit 3
Bit 4		0	Bit 4
Bit 5		0	Bit 5
Bit 6		0	Bit 6
Bit 7		0	Bit 7

Port F

Data Direction		Pullup or Output val	
Bit 0		0	Bit 0
Bit 1		0	Bit 1
Bit 2		0	Bit 2
Bit 3		0	Bit 3
Bit 4		0	Bit 4
Bit 5		0	Bit 5
Bit 6		0	Bit 6
Bit 7		0	Bit 7

The meaning of the *Data Direction* and *Pullup or Output val* entries can be found in the controller manual.

Click repeatedly *next* until the last page of the Application Wizard
Click *Build Application*, *Store* and *Exit*

You are should now be in the editor with the new program frame for Tutor02 loaded.

press Ctrl+F9 to "make" (compile and assemble) the program frame.
This should not produce any errors

2.3.2 Enter And "Make" The Program

Enter the definitions for the constant "pattern", the variable "sel" and the main program between the keywords *loop* and *endloop* according the following listing.
Note that most of the program is done by the *Application Wizard*. You have to complete only the blue lines.

Enter the programm step by step!
Start with the "const" block and "make" the program again. Correct any errors before proceeding.
Continue with the "var" definition and let the compiler check the syntax (press Ctrl+F9).
The next step should be the **bold blue** block below the keyword *loop.*
The last lines can be entered (and checked) one by onother.

Remember that the compiler is frequently **not** able to locate the exact location of an error.
If you change too much at once it may be hard to find the error(s).

Do **not** try to correct several errors at once (unless the same type at the same location, e.g. several forgotten commas). Mostly your statements behind the first error are misintepreted by the compiler.
If you got several errors a click on any error message moves the cursor to the related line in the source file.

If the "make" succeeds try the following:

- remove the braket at the **end** of { Const Declarations **}**
- "make" the programm and notice the error location the compiler assumes
- do not forget to correct your source again

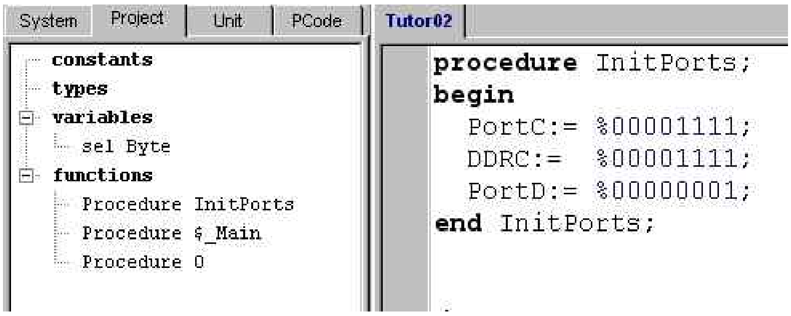
You see: the less you change (or add) at once the easier is the debugging.

```
program Tutor02;

{ $BOOTRST $00C00}         {Reset Jump to $00C00}
{$NOSHADOW}
{ $W+ Warnings}                  {Warnings off}

Device = mega8, VCC = 5;

Import SysTick, SwitchPort1;

From System Import;

Define
  ProcClock    =   1000000;        {Hertz}
  SysTick      =  10;              {msec}
  StackSize    =  $0064, iData;
  FrameSize    =  $0064, iData;
  SwitchPort1  =  PinD, $01;
  PolarityP1   =  $00;              // polarity
```

```
Implementation

{$IDATA}

{---------------------------------------------------------------}
{ Type Declarations }

type


{---------------------------------------------------------------}
{ Const Declarations }

const
  pattern        : array [0..3] of byte =
  (
  %00000000,
  %11000011,
  %10100101,
  %10010110);

{---------------------------------------------------------------}
{ Var Declarations }
{$IDATA}

var
  sel            : byte;

{---------------------------------------------------------------}
{ functions }

procedure InitPorts;
begin
  PortC:= %00001111;
  DDRC:=  %00001111;
  PortD:= %00000001;
end InitPorts;


{---------------------------------------------------------------}
{ Main Program }
{$IDATA}

begin
  InitPorts;

  EnableInts;

  loop

  if Inp_Raise1(0)
  then
    inc (sel);
    sel := sel mod 4;
  endif;

  PortC := not (Pattern[sel] shr 4);
  mDelay(250);
  PortC := not (Pattern[sel] and $0F);
  mDelay(250);

  endloop;

end Tutor02.
```

2.3.3 Some Nice And Useful Feature Of The Editor

- click on the left pane (in the editor) the *Project* tab and expand all items



```
constants
types
variables
   sel Byte
functions
   Procedure InitPorts
   Procedure $_Main
   Procedure 0
```

```
procedure InitPorts;
begin
  PortC:= %00001111;
  DDRC:=  %00001111;
  PortD:= %00000001;
end InitPorts;
```

- double click *sel* and the procedures. The cursor moves at once to the corresponding definition. Especially in large source files this function is very useful.

- position the cursor on a keyword (e.g. behind the "c" of the above "procedure InitPorts;")
- press F1. Note the possiblity to copy the examples via the clipboard to your source.

- destroy the formatting of some lines by deleting leading blanks and "make" the source

  e.g:

```
procedure InitPorts;
begin
PortC:=%00001111;
DDRC:=%00001111;
PortD:= %00000001;
end InitPorts;
```

- click the *beautifier* speed button

- position the cursor behind the "*end Tutor02.*" - line
  (all lines below this statement are ignored by the compiler. This is a save place to play around with the editor).
  enter the first characters of a statement, e.g. *md*
  press Ctrl + <blank> and <enter> to accept the mDelay procedure

- t ry to enter a single character and press Ctrl + <blank>

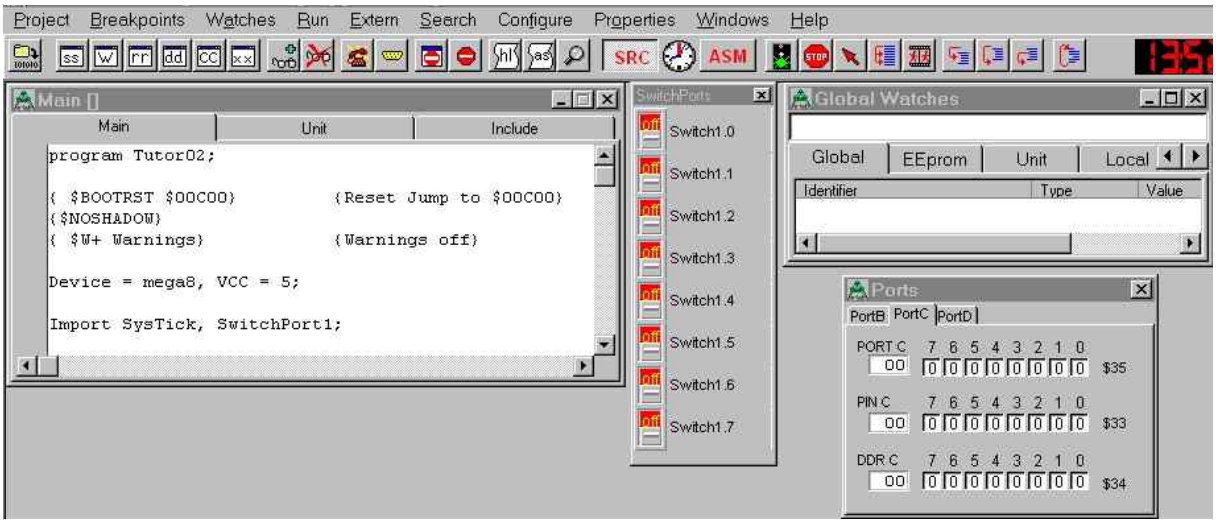
- press Shift + Ctrl + p. This is a keyboard macro and creates an empty frame for a procedure.

- open the menu *IDE – Edit Keyboard macros* to see all predefined macros

- remove all test lines and "make" the program again

2.3.4 The Simulator – The Only Way To Success

if the "make" was successful open the simulator
to have a better survey close all windows but *main[]*, *Global Watches* and *Ports*
enable the *SwitchPort* in the *Windows* menu
select the *PortC* tab at the *Ports* windows. Arrange and size the windows

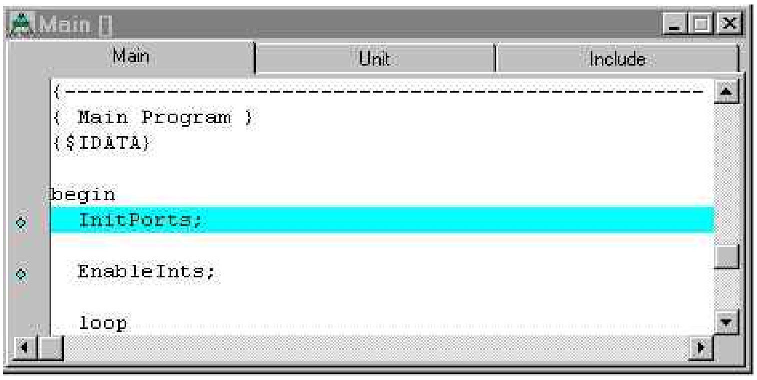


```
program Tutor02;

{ $BOOTRST $00C00}          {Reset Jump to $00C00}
{$NOSHADOW}
{ $W+ Warnings}             {Warnings off}

Device = mega8, VCC = 5;

Import SysTick, SwitchPort1;
```

```
PORT C   7 6 5 4 3 2 1 0
  00     0 0 0 0 0 0 0 0   $35

PIN C    7 6 5 4 3 2 1 0
  00     0 0 0 0 0 0 0 0   $33

DDR C    7 6 5 4 3 2 1 0
  00     0 0 0 0 0 0 0 0   $34
```

press once F7 to execute a single step (see the menu *Run* and notice the 3 speed buttons for single stepping)

This executes the initialization of the variables and the imported drivers and places the cursor on the first line of your program: the call of the procedure *InitPorts*.



```
{------------------------------------------------
{ Main Program }
{$IDATA}

begin
  InitPorts;

  EnableInts;

  loop
```

- to step through this procedure press F7
- execute the next command: *PortC:= %00001111;* (use again F7)
- watch the 1st line in the *Ports* window (make sure, you have selected the *PortC* tab)

- execute the next command and watch how the register *DDR C* changes (the 3rd line)
- as the next command changes a register of PortD, click the *PortD* tab in the *Ports* window
- execute the command and press twice F7. You are again in the main program at the 2nd line: *EnableInts;*
- before you proceed to step through the program play with the switches in the *SwitchPorts* window and watch the display of *Pin D*. Note that an open switch is displayd as "1" and a closed as "0"
- if you missed a thing you can easily start over by *Reset processor*
- try it: *Reset* the processor and execute this time the *InitPorts* procedure with F8 (in a single step)

- position all switches to "off" and select the *PortC* tab before you continue

- press repeatedly F7 and watch the program loop

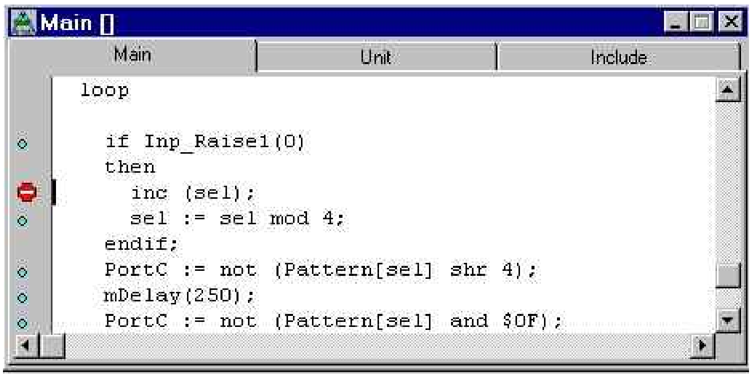
- remember: we will connect 4 LEDs to *Port C0* .. *Port C3*. These port pins stay at "1" while the program loops. A "1" means: the LED is **off** ! Hold F7 depressed (to constantly loop) while watching the *Ports* window

- note that the **if**-statement never becomes *true* and the statements inside the body of the if-statement are never executed

- the variable *sel* is used to select the display pattern. What is its value? Search for one occurence of the variable and double click it. This opens a window to edit the variable or Add it to WatchList. Click *Add to WatchList* and close the window.

- note the variable in the *Global Watches* window while you loop through the program

- click the blue circle in front of the line *inc (sel);* to set a breakpoint



```
loop

  if Inp_Raise1(0)
  then
    inc (sel);
    sel := sel mod 4;
  endif;
  PortC := not (Pattern[sel] shr 4);
  mDelay(250);
  PortC := not (Pattern[sel] and $0F);
```

- press F9 to (free) run the simulator. Note the clock icon and the grayed out speedbuttons. The program is now running until you stop it or the program flow reaches a breakpoint

- simulate a key hit by positioning the *Switch1.0* to *on* and back to *off* (not too fast!)
- the if-satement becomes true, the program flow reaches the breakpoint and the program stops
- continue with one step (F7) and see the new value of the variable *sel*
- hold F7 depressed and watch *Port C*: the first alternating pattern is displayed
- repeat the last steps (F9, key hit, F7) and watch the next pattern(s)
- stop the simulator (if it is running) STOP and remove the above breakpoint by clicking on it.
- press Ctrl+F9 to start the *Animate* function or use the speed button
- play with the switches. Note that only *Switch1.0* changes the pattern.
  The simulated controller is running at a very low speed to make the changes visible. As the *SwitchPort* is debounced it takes some time (some processor clock cycles) to accept the switch as *on* or *off*.
- stop the simulator (if it is running) and position the cursor at the line *inc (sel);* in the source window.
  Press F4 *Goto cursor pos* or the speedbutton and simulate a key hit

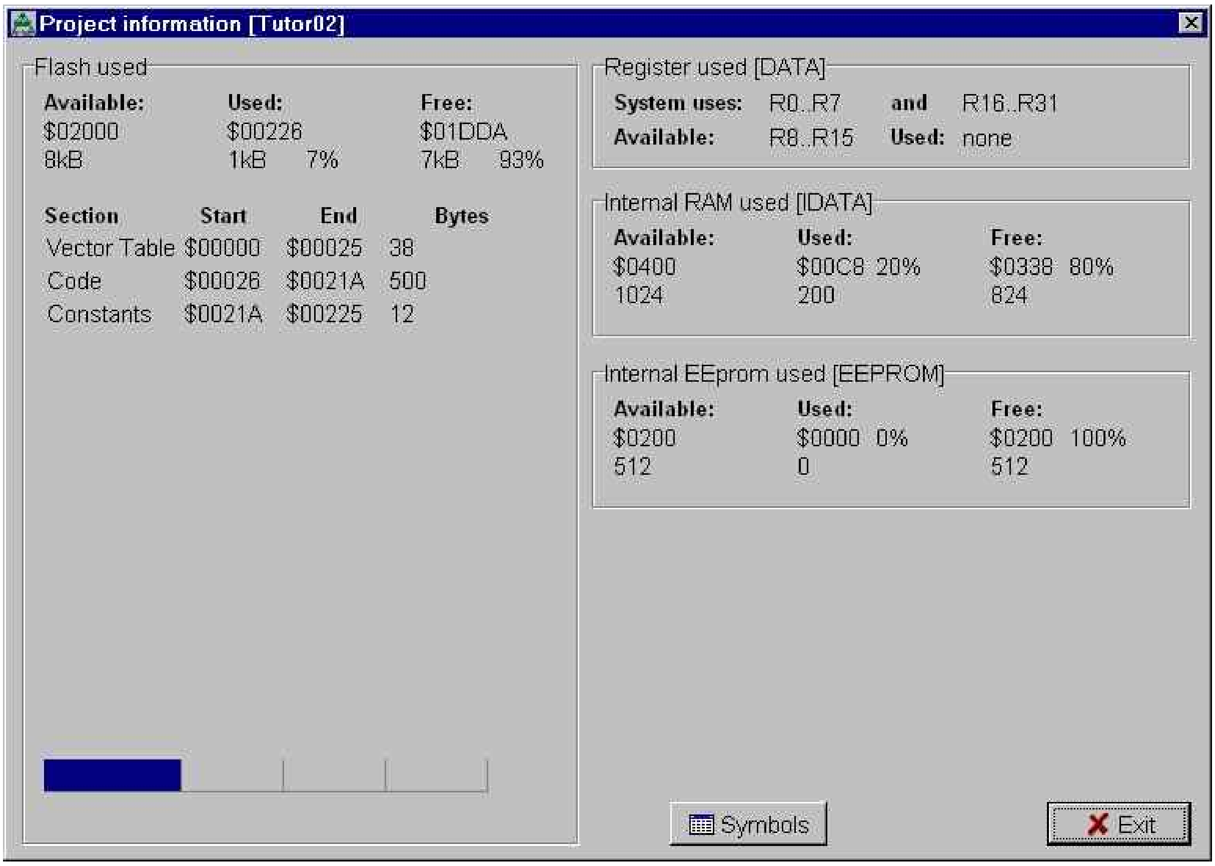
The simulator is an undispensable part of the AVRco and not only nice-to-have!
No first version of any program will ever run as desired. And if, nevertheless, it does, be twice as suspicious – the experience tells that the bugs exist but are well hidden.

The above covered techniques should be sufficient to debug many simple programs.
If you start using more complex functions like interrupts or MuliTasking there are a lot of more useful features in the simulator. Most of them are self-explaining if you understood the concerning background: e.g. the use of the internal EEProm.
Other functions, as the debugging on assembly level, should rarely be necessary.

Unfortunately the builtin help of the simulator is not yet available.
If you experience heavy problems follow the guidelines in the chapter *"Additional Ressources: The AVRco Documentation".*

2.3.5 Get More Useful Informations

- in the editor choose *Project – Project Informations* or click to open the following window



Project information [Tutor02]

Flash used

Available:	Used:	Free:
$02000	$00226	$01DDA
8kB	1kB 7%	7kB 93%

Section	Start	End	Bytes
Vector Table	$00000	$00025	38
Code	$00026	$0021A	500
Constants	$0021A	$00225	12

Register used [DATA]

System uses:	R0..R7	and	R16..R31
Available:	R8..R15	Used:	none

Internal RAM used [IDATA]

Available:	Used:	Free:
$0400	$00C8 20%	$0338 80%
1024	200	824

Internal EEprom used [EEPROM]

Available:	Used:	Free:
$0200	$0000 0%	$0200 100%
512	0	512

One handicap of the microcontrollers is the limited amount of ressources.
Use the above function to get an overview of the used ressources:

- Flash
  - -the program memory
  - -you see an *available* size of 8kB. This is true for the Mega8 controller. But keep in mind that the AVRco-Demo is restricted to 4kB
  - -pay special attention on the *Used* entry: including all system imports Tutor02 needs only about 1kB. There are still 3kB available for your program (and flash-stored constants) (1)

notes:

(1): be aware that the AVRco includes only the necessary parts of a system library:
e.g.: if you import "float" and use only a float multiply, the (assembly) subroutine for float divison is **not I**ncluded!
Check it out when you have finished the tutorial:
try a := a * 1.5; a := a / 0.5; and check the code size.
Start over and
try a := a * 1.5; a := a * 2.0; and check the size again.
Open the .lst files. Scroll down to the end and compare the section "Imported Library Routines" in both cases.

- [DATA]
  -the processor registers (1)
  -is (actually) not important. Only for advanced programming techniques.

- [IDATA]
  -the amount of Ram used for the variables (your own and variables the AVRco uses internally). As there are still 80% free: no reason to worry about

- [EEPROM]
  -only important if you use the internal EEProm

- choose (in the editor) *Info – Internet update* to see your current version of the AVRco. Check regulary the E-LAB home page for enhanced versions and pay attention to the latest *DocuAddOnV-x*

open the symbol file *C:\E-Lab\Projects\Tutor02.sym* to see the symbols the AVRco has defined. Compare them with the controller manual and note that you can address all processor registers by their names

refer to the chapters "*types – BIT"* and "*System Library – BIT" Processing* of the *"DocuCompiler"* manual to access single bits with symbolic names

e.g. to access the *Carry Flag* (bit 0) of the *Status Register (SREG)* as variable "CFlag" you may define

```
var
CFlag [@SREG,0] : Bit;
```

or (3)

```
{$PDATA}
CFlag [$5F, 0] : Bit;
{$IDATA}
```

notes:
(1): the amount of available processor registers depend on the inluded drivers. Use this info when your project is nearly finished to decrease the code size and increase the speed (see the *"DocuCompiler"* Manual)
(2): refer to the chapter *Compiler Switches* of the *"DocuCompiler"* manual for explanations of {$PDATA} and {$IDATA}.
{$PDATA} tells the compiler to address the internal processor registers.
Make sure not to forget the {$IDATA} to address the Ram area of the controller for subsequent defines

2.4 The Internal EEProm

the internal EEProm provides a non-volatile memory area that can be accessed

either at runtime

> e.g. to store user preferences, calibration values etc. that must be available
> at next power-up

or already at compile time

> e.g. to store fixed menus, lookup tables etc.

Although any value, already known at compile time, may also be stored in the flash,
an EEProm storage can free flash memory for program code.

There are several ways to access the EEProm with the AVRco.
Refer to the dedicated chapter of the *"DocoCompiler"* manual! (1)

If you specify EEProm data in your source file (see {$EEPROM} compiler switch) using the **StructConst** directive the AVRco creates an additional file with an *.eep*-extension. (2)
This file must, besides the program file for the flash memory (extension *.hex*), be downloaded to the controller.

Ususally the programming sotware has the options (3)

- read / write / verify all (Flash and EEProm)
- read / write / verify Flash
- read / write / verify EEProm
- erase all (Flash and EEProm)

notes

(1): access to EEProm is by far not so straight forward as access to RAM. It needs a specific handling and timing.
The AVRco takes this burdon: you can treat EEProm nearly the same way as RAM, but you should be aware that an EEProm access is much slower than a RAM access!

(2): note that you have to define any data to create this file. You may also define variables in the EEProm area. This reserves only the necessary bytes but does not create an .eep-file. See the STRUCTCONST statement on how to place constants in the EEProm.

(3): some programming software defaults to other extensions (e.g. .e2p for EEProm file) and you have to change the extension before browsing to the file

2.5 Additional Ressources: The AVRco Documentation

The compiler documentation is contained in ***C:\E-Lab\DOCs\DocuCompiler.pdf***.
The drivers are in ***C:\E-Lab\DOCs\DocuStdDrivers.pdf*** documented.
Complex drivers that are only supported with the AVRco Profi Version can be found in ***C:\E-Lab\DOCs\DocuProfiDrivers.pdf***
The additional programs in this manual ***C:\E-Lab\DOCs\DocuTools.pdf***

The actual enhancements are documented in ***C:\E-Lab\DOCs\DocuAddOn.pdf***.
The most recent *DocuAddOn.pdf* can also be downloaded from the E-LAB homepage.

2.5.1 How to Find all related Informations?

- the first ressource are the AVRco manuals

- new drivers are frequently published together with an example. The related files can be found in *C:\E-LAB\AVRco\Demos.*

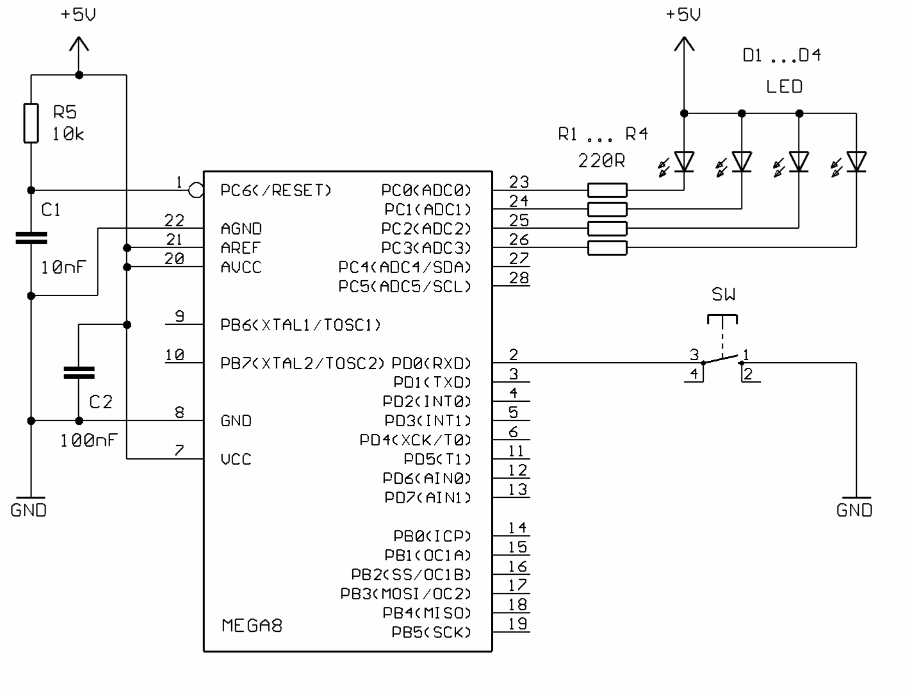
- if you have a very specific interest or problem, it can be helpful to search the C:\E-Lab directory (including subdirectories) for files containing typical keywords

- if you do not succeed search the AVRco forum on the E-LAB home page for the keywords

- the last rescue:
  poste your question in the E-LAB AVRco forum.
  Although most threads are in german do not hesitate to post in english!

2.6 What Hardware Do You Need?

As mentioned above the hardware is very simple.
All you need, besides the controller, the switch and the 4 LEDs are 5 resistors and 2 capacitors.
The current consumption is far below 50mA and depends on the on/off state of the LEDs.



Note that the LEDs are connected with a command anode. A "0" at a port will switch the LED "on".
The switch is connected to GND. PD0 becomes "0" when the switch is depressed.
LEDs and switch are "active low".
By setting the correspondig *Pullup or Output val* bits we made sure that the initial state of the LEDs is off and the internal pullup resistor at PD0 is activated.

If you want to setup a hardware by our own, the easiest way is the use of an breadboard.

Pay special attention on the power supply! A supply voltage above 5.5V may destroy the controller.
Should you not have the necessary hardware knowledge I recommend the use of a commercial mini-board and power supply (see links near the end of the tutorial).
If you use a commercial circuit edit the *ProcClock Define* and enter the correct frequency!

2.6.1 Some additinional hints concerning the hardware

1. remember: controllers with an "*internal oscillator*" option (like the Mega8) are delivered with the fuse bytes programmed to: "internal oscillator at 1 MHz". The XTAL 1/2 Pins are at delivery defined as general purpose I/O pins.

   To use an external crystal oscillator you have to change some fuse bits. Refer to the example in the appendix for a Mega8 running at 8 to 16 MHz.
   It is a good idea to document the default and new values for later reference.

   again: be extremely careful with the fuse bytes!

   before making any other changes read and understand the corresponding section in the controller manual!
   Take special care on how the bits are presented by the programming software. The controller interprets a value of "1" as unprogrammed and a value of "0" as programmed.

   Some programming software displays these bits "inverted"!
   e.g.: as check-boxes with checked=programmed="0" and unchecked=unprogrammed="1".
   Although this should be to your convenience it may be very confusing.

   The fuse bytes are accessed like the flash and EEProm memory with a programmer and a programming software.

2. take care: the newer controllers with 40pins and above (like the Mega16) come with a builtin JTAG interface. This interface uses 4 port pins and is, at delivery, enabled. These port pins can **not** be used as general I/Os without changing the fuse bytes.
   e.g.:
   the JTAG on the Mega16 uses PortC 2 to PortC 5. To use these ports as general I/O pins you have to disable (= unprogram = set to "1") the JTAGEN bit.

3. a frequently asked question is the value of the capacitors for an external crystal oscillator.
   This value depends on the oscillator itself and is specified by the manufacturer of the oscillator (not by the controller!).
   Place the crystal oscillator and the capacitors as close as possible to the controller.
   The value of the capacitors is not very critical and 22pF fits in most cases.

4. to display the clock signal on an oscilloscope use a probe with an attenuation of 1:10.
   Check only the XTAL 2 pin - **not** the XTAL 1 pin (this may stop the oscillation because of the additional load of the probe)

5. another source of confusion is often the behaviour of character LCDs that are only connected to the supply voltage. As long as the display is not properly intilialized most displays show the odd lines as black bars, the even lines stay bright
   e.g.
   -on a 2-line display the 1st line is black, the 2nd bright
   -on a 4-line display the lines 1&3 are black, 2&4 bright
   Do not worry. The LCD is **not** faulty. This is "normal". But take care: before you buy a LCD make sure, the AVRco supports the controller type the LCD uses (the display-contoller is a SMD chip (or several) mounted on the rear of the LCD board). The supported types can be found on the *LCD port* page of the *Application Wizard*.
   Unfortunately some so called "compatible" controllers are not "compatible enough" and cause problems.
   If you do not see the the bar(s) check the contrast-voltage. LCDs with an enhanced temperature range (for automotive applications) need a negative contrast-voltage (as the Grahic LCDs do). Take care to use the correct section of the datasheet:
   most LCDs are manufactured in several versions (temperature ranges) and use different ranges of the contrast voltage.

2.7 What Do You Need To Program Your Hardware?

the AVRco creates a program file (e.g. ...\Projects*TutorXX.**hex***). This file must be transferred to the internal flash memory of your controller.
Maybe it created also an EEProm file (e.g. ...\Projects*TutorXX.**eep***). This file must be transferred to the internal EEProm memory of your controller.

Until now these files reside on your harddisk. To download them to your hardware you need

1. some kind of connection between your PC and your controller hardware
2. a software that transferers the file(s) and is able to check them (via read-back)

The controllers offer several builtin functions to write/read/erase the internal memory areas.

This chapter describes the most popular one, the SPI (Serial Peripheral Interface).

The others are

- the "Parallel Programming Mode" that is very complex and
- the "JTAG Mode" that is only available on controllers with >= 40pins and has a lot of extended features (like OnChip debugging)

The SPI is a synchonous interface that allows data transfers from/to peripheral devices.
At a Reset (while the Reset pin is held low) the SPI enters a special "*Serial Programming Mode*“ that allows to access the internal memory areas.
The *Serial Programming* uses the pins SCK, MOSI and MISO for data transfer. (1)
These pins (+ Reset) must be controlled by the PC.

On the PC there are actually 3 types of interfaces to connect such an equipment:
a parallel port, a serial port (= V24 = RS232) or an USB port.

To avoid confusion:

- the programming is always done serially (also with the parallel PC port!)
- the "Parallel Programming Mode" has nothing to do with the parallel PC port (here is the "program" and "verify" done in parallel)

To connect the controller to the PC you need a programmer hardware (called ICP - In Circuit Programmer or ISP – In System Programmer).

All types of programmers can be found:
serial/parallel/USB connected, professional and do-it-youself types.

And you need a programming software that fits to your programmer. There are also commercial and free versions.

As the do-it-yourself programmers (usually published together with a free software) cause very often problems, I recommend the use of a commercial one.

Links to do-it-yourself programmers and free software can be found near the end of the tutorial.

Furthermore all programmers / programming software allow you access to the configuration bytes (fuses) of the controllers.

Note:
(1) there are exeptions like the Mega128. Check the Controller Manual!

E-LAB offers different types with serial (V24) and USB connection and the AVRco comes with a programming software for these programmers. The software is an integral part of the IDE, so you can use a speed button to download your application file(s).

<u>This is the most secure and convenient way to program your hardware.</u>

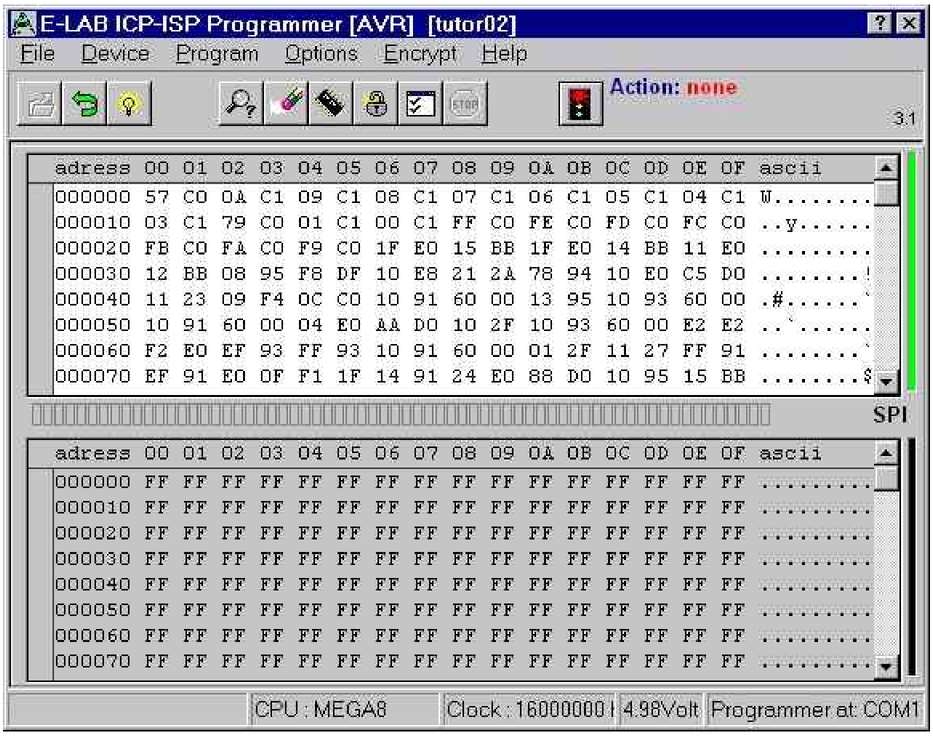
The commercial versions (standard and profi) of the AVRco come already with a programmer included. The standard version includes a V24/RS232 type “ISP-V24” and the profi version includes an USB type “ISP-USB” which also serves as the JTAG-ICE debugger.
All E-LAB programmer types support both modes, JTAG **and** SPI programming.

2.7.1 How To Use The E-LAB Programmer (serial programmer, SPI mode)

connect the programmer to your PC and your hardware and power the system on.
Refer to the programmers-manual for the different modes to provide the power.
Compile your project with the desired frequency value and frequency source. (1)

Press the *Prommer* speed button in the IDE to start the programming software: *AVRprog.exe*

Note:
(1): it does **not** matter that some new controllers come with the internal clock source at 1 MHz selected.
The first step of the programming cycle is to set the fuse bits and select the correct ferquency.
All further programming is done with that clock selection.



E-LAB ICP-ISP Programmer [AVR] [tutor02]

```
adress 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ascii
000000 57 C0 0A C1 09 C1 08 C1 07 C1 06 C1 05 C1 04 C1 W........
000010 03 C1 79 C0 01 C1 00 C1 FF C0 FE C0 FD C0 FC C0 ..y......
000020 FB C0 FA C0 F9 C0 1F E0 15 BB 1F E0 14 BB 11 E0 .........
000030 12 BB 08 95 F8 DF 10 E8 21 2A 78 94 10 E0 C5 D0 ........!
000040 11 23 09 F4 0C C0 10 91 60 00 13 95 10 93 60 00 .#......`
000050 10 91 60 00 04 E0 AA D0 10 2F 10 93 60 00 E2 E2 ..`......
000060 F2 E0 EF 93 FF 93 10 91 60 00 01 2F 11 27 FF 91 ........`
000070 EF 91 E0 0F F1 1F 14 91 24 E0 88 D0 10 95 15 BB ........$
```

```
adress 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ascii
000000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
000010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
000020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
000030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
000040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
000050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
000060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
000070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .........
```

CPU : MEGA8 | Clock : 16000000 | 4.98Volt | Programmer at: COM1

All needed informations are automatically transfered from the IDE to the AVRProg:

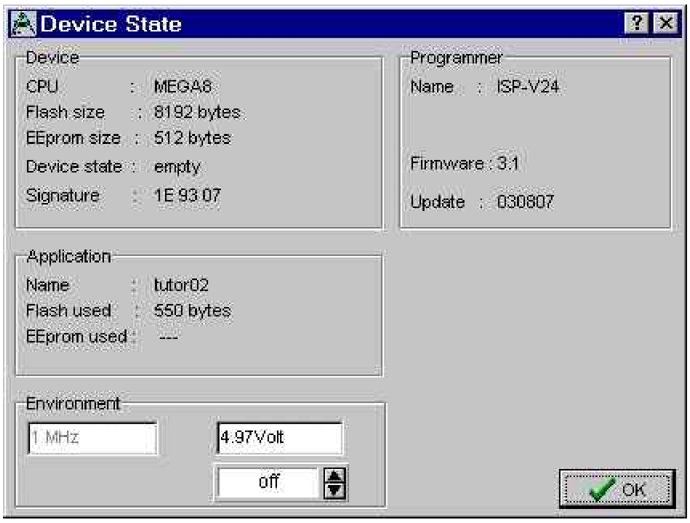
- the name of the flash file
- the name of the EEProm file (if available)
- the controller type
- the clock frequency

and the file(s) is (are) loaded.

Note: the "*Programmer at COM1*" message at the lower right corner. This indicates the programmer has been successfully detected.

Press the *Device Check* speed button

On a new Mega8 you should get a message like:



Device State

Device
CPU : MEGA8
Flash size : 8192 bytes
EEprom size : 512 bytes
Device state : empty
Signature : 1E 93 07

Programmer
Name : ISP-V24
Firmware : 3.1
Update : 030807

Application
Name : tutor02
Flash used : 550 bytes
EEprom used : ---

Environment
1 MHz
4.97Volt
off

I M P O R T A N T :

**if you start the AVRprog <u>the very first time with a new project</u> you MUST at first select the *Programmer Options*!
This has usually to be done only once (because the AVRprog stores your selections for each project).
See the following pages for guidelines!**

<u>THIS MUST DE DONE (once) FOR EACH NEW PROJECT!</u>

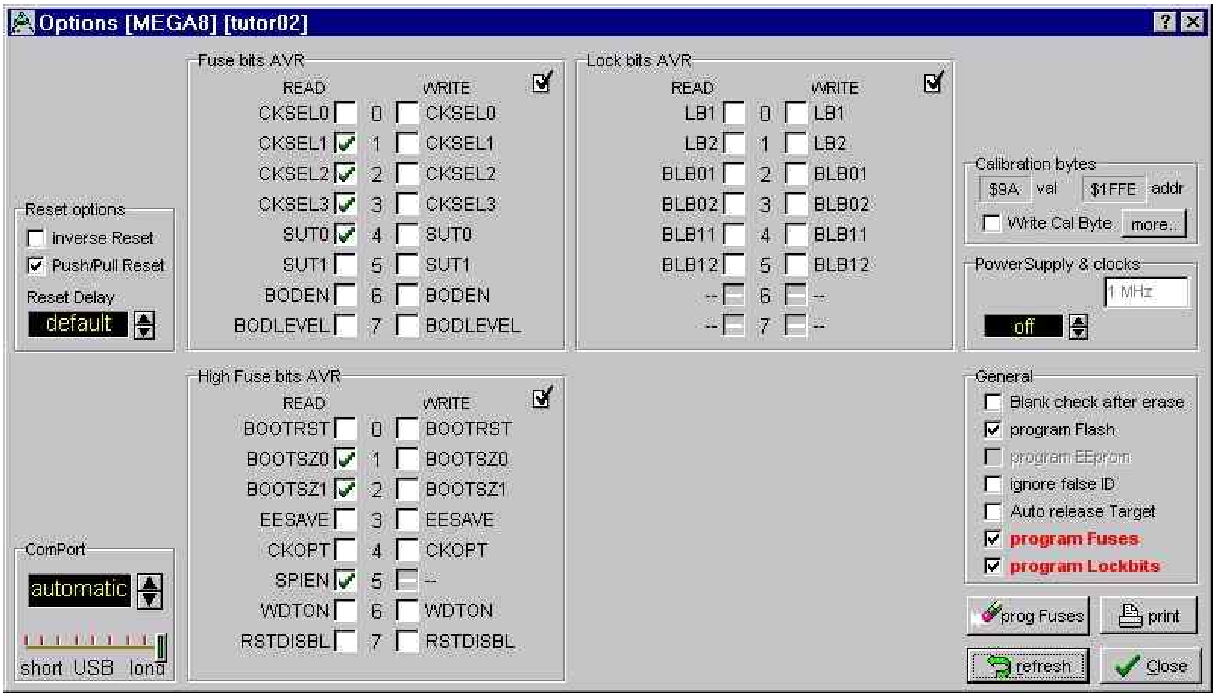
**The AVRprog defaults to "program Flash, EEProm and Fuse Bytes".
The default for all fuse bits is inactive (binary "1").**

**e.g.:
without the correct selection the first programming cycle will set a Mega8 to "External Crystal / Ceramic Resonator" (CKSEL 3..0 = "1111").
In oder to access the Mega8 again and change this selection you need to connect such a clock device (if you do not already have).**

2.7.2 How To Set The Programmer Options For a New Project

goto "Options" – "Programmer Options"

the programmer has not yet read the fuse bits and all are unchecked
press "Refresh" to read the bits from the controller (note the controller type in the header line)
you should see now the actual state of the fuses (below the factory default)



Options [MEGA8] [tutor02]

Reset options
☐ inverse Reset
☑ Push/Pull Reset
Reset Delay: default

ComPort: automatic

Fuse bits AVR

READ		WRITE
CKSEL0 ☐	0	☐ CKSEL0
CKSEL1 ☑	1	☐ CKSEL1
CKSEL2 ☑	2	☐ CKSEL2
CKSEL3 ☑	3	☐ CKSEL3
SUT0 ☑	4	☐ SUT0
SUT1 ☐	5	☐ SUT1
BODEN ☐	6	☐ BODEN
BODLEVEL ☐	7	☐ BODLEVEL

Lock bits AVR

READ		WRITE
LB1 ☐	0	☐ LB1
LB2 ☐	1	☐ LB2
BLB01 ☐	2	☐ BLB01
BLB02 ☐	3	☐ BLB02
BLB11 ☐	4	☐ BLB11
BLB12 ☐	5	☐ BLB12
-- ☐	6	☐ --
-- ☐	7	☐ --

High Fuse bits AVR

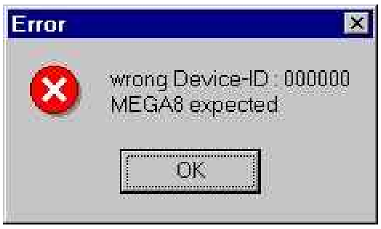
READ		WRITE
BOOTRST ☐	0	☐ BOOTRST
BOOTSZ0 ☑	1	☐ BOOTSZ0
BOOTSZ1 ☑	2	☐ BOOTSZ1
EESAVE ☐	3	☐ EESAVE
CKOPT ☐	4	☐ CKOPT
SPIEN ☑	5	☐ --
WDTON ☐	6	☐ WDTON
RSTDISBL ☐	7	☐ RSTDISBL

Calibration bytes
$9A val $1FFE addr
☐ Write Cal Byte

PowerSupply & clocks
1 MHz
off

General
☐ Blank check after erase
☑ program Flash
☐ program EEprom
☐ ignore false ID
☐ Auto release Target
☑ program Fuses
☑ program Lockbits

- if you get an error message like



Error

wrong Device-ID : 000000
MEGA8 expected

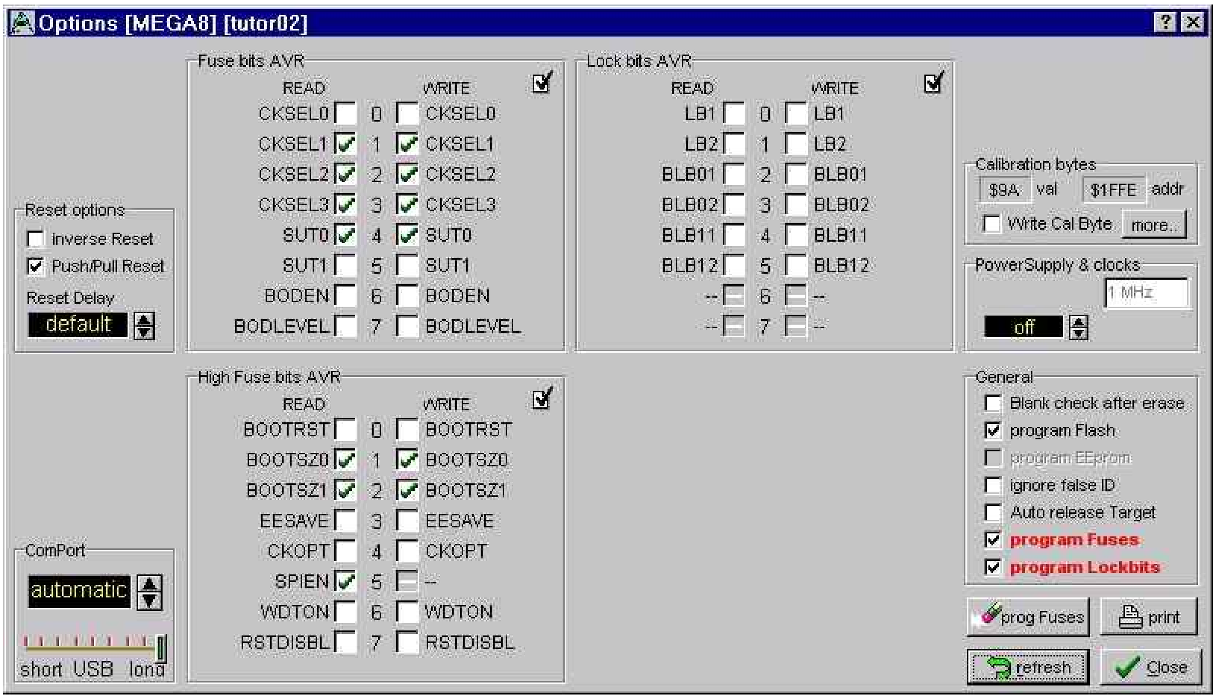
the controller fuses are (already) wrong and you need to apply the correct clock source to re-program the chip. There is no information what kind of clock source is "correct".

- If you blew your controller with the AVRprog most likely all fuses are disabled.
  A Mega8 needs an external crystal or resonator to be accessed ("CKSEL3..0" are "1111").
  For other controllers refer to the controllers manual for the meaning of "all fuses = "1".

- note that the JTAG programming mode does **not** need a clocked controller

- if you do not have a error message select the **bits to <u>WRITE</u>**.

e.g.
if you use the default (internal oscillator at 1 MHz) check the same bits



Options [MEGA8] [tutor02]

Fuse bits AVR

READ			WRITE ☑
CKSEL0	□	0	□ CKSEL0
CKSEL1	☑	1	☑ CKSEL1
CKSEL2	☑	2	☑ CKSEL2
CKSEL3	☑	3	☑ CKSEL3
SUT0	☑	4	☑ SUT0
SUT1	□	5	□ SUT1
BODEN	□	6	□ BODEN
BODLEVEL	□	7	□ BODLEVEL

Lock bits AVR

READ			WRITE ☑
LB1	□	0	□ LB1
LB2	□	1	□ LB2
BLB01	□	2	□ BLB01
BLB02	□	3	□ BLB02
BLB11	□	4	□ BLB11
BLB12	□	5	□ BLB12
--	□	6	□ --
--	□	7	□ --

High Fuse bits AVR

READ			WRITE ☑
BOOTRST	□	0	□ BOOTRST
BOOTSZ0	☑	1	☑ BOOTSZ0
BOOTSZ1	☑	2	☑ BOOTSZ1
EESAVE	□	3	□ EESAVE
CKOPT	□	4	□ CKOPT
SPIEN	☑	5	□ --
WDTON	□	6	□ WDTON
RSTDISBL	□	7	□ RSTDISBL

Reset options
□ inverse Reset
☑ Push/Pull Reset
Reset Delay: default

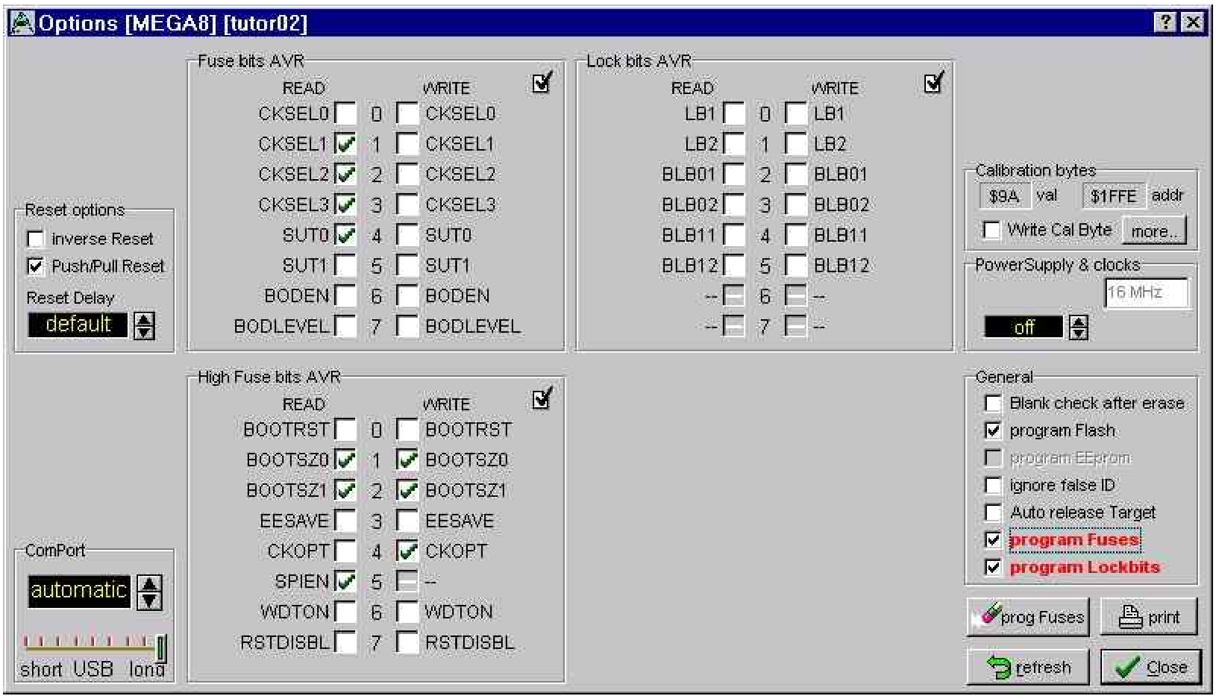
ComPort: automatic
short USB long

Calibration bytes
$9A val $1FFE addr
□ Write Cal Byte

PowerSupply & clocks
1 MHz
off

General
□ Blank check after erase
☑ program Flash
□ program EEprom
□ ignore false ID
□ Auto release Target
☑ program Fuses
☑ program Lockbits

or, if you want to select an external clock from 8 to 16 MHz (see Appendix) un-/check


Options [MEGA8] [tutor02]

Fuse bits AVR

READ			WRITE
CKSEL0 ☐	0	☐ CKSEL0
CKSEL1 ☑	1	☐ CKSEL1
CKSEL2 ☑	2	☐ CKSEL2
CKSEL3 ☑	3	☐ CKSEL3
SUT0 ☑	4	☐ SUT0
SUT1 ☐	5	☐ SUT1
BODEN ☐	6	☐ BODEN
BODLEVEL ☐	7	☐ BODLEVEL

Lock bits AVR

READ		WRITE
LB1 ☐	0	☐ LB1
LB2 ☐	1	☐ LB2
BLB01 ☐	2	☐ BLB01
BLB02 ☐	3	☐ BLB02
BLB11 ☐	4	☐ BLB11
BLB12 ☐	5	☐ BLB12
-- ☐	6	☐ --
-- ☐	7	☐ --

High Fuse bits AVR

READ		WRITE
BOOTRST ☐	0	☐ BOOTRST
BOOTSZ0 ☑	1	☑ BOOTSZ0
BOOTSZ1 ☑	2	☑ BOOTSZ1
EESAVE ☐	3	☐ EESAVE
CKOPT ☐	4	☑ CKOPT
SPIEN ☑	5	☐ --
WDTON ☐	6	☐ WDTON
RSTDISBL ☐	7	☐ RSTDISBL

Reset options
☐ inverse Reset
☑ Push/Pull Reset
Reset Delay: default

ComPort: automatic
short USB long

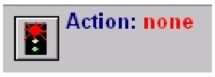
Calibration bytes: $9A val $1FFE addr
☐ Write Cal Byte

PowerSupply & clocks: 16 MHz, off

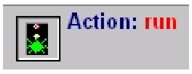
General
☐ Blank check after erase
☑ program Flash
☐ program EEprom
☐ ignore false ID
☐ Auto release Target
☑ program Fuses
☑ program Lockbits

- to download your project press the *Program* speed button of the AVRprog

after the download your hardware stays locked. That means the *Reset* of your controller is held active (low)

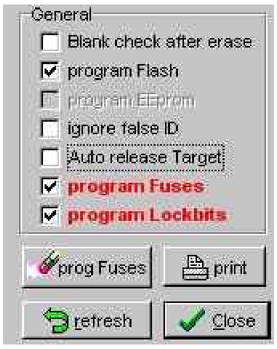

Action: none

to start the program click the traffic light


Action: run

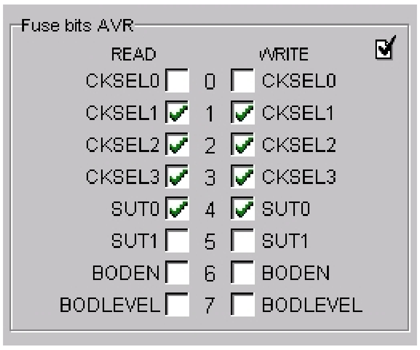
the *Reset* pin is released and your applications starts

if you prefer to start your application as far as the download ist complete select the programmer options and enable the "*Auto release Target*"


General
☐ Blank check after erase
☑ program Flash
☐ program EEprom
☐ ignore false ID
☐ Auto release Target
☑ program Fuses
☑ program Lockbits

- note that –depending on the project and the controller- some options may be "grayed out".
  e.g.:
  above the grayed out "program EEprom" selection as there is no EEProm file available

- note that *AVRprog* presents the **logical** state of the bits:

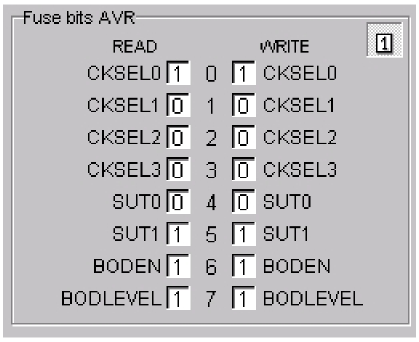


Fuse bits AVR ☑

READ			WRITE
CKSEL0 □	0	□	CKSEL0
CKSEL1 ☑	1	☑	CKSEL1
CKSEL2 ☑	2	☑	CKSEL2
CKSEL3 ☑	3	☑	CKSEL3
SUT0 ☑	4	☑	SUT0
SUT1 □	5	□	SUT1
BODEN □	6	□	BODEN
BODLEVEL □	7	□	BODLEVEL

e.g.: CKSEL0 is not activated, CKSEL1 ist activated

if you are confused or want to compare the bits with the controllers manual click the checkbox in the upper right corner to display the **binary** values of the fuses



Fuse bits AVR 1

READ			WRITE
CKSEL0 1	0	1	CKSEL0
CKSEL1 0	1	0	CKSEL1
CKSEL2 0	2	0	CKSEL2
CKSEL3 0	3	0	CKSEL3
SUT0 0	4	0	SUT0
SUT1 1	5	1	SUT1
BODEN 1	6	1	BODEN
BODLEVEL 1	7	1	BODLEVEL

2.8 At A Glance: Multitasking

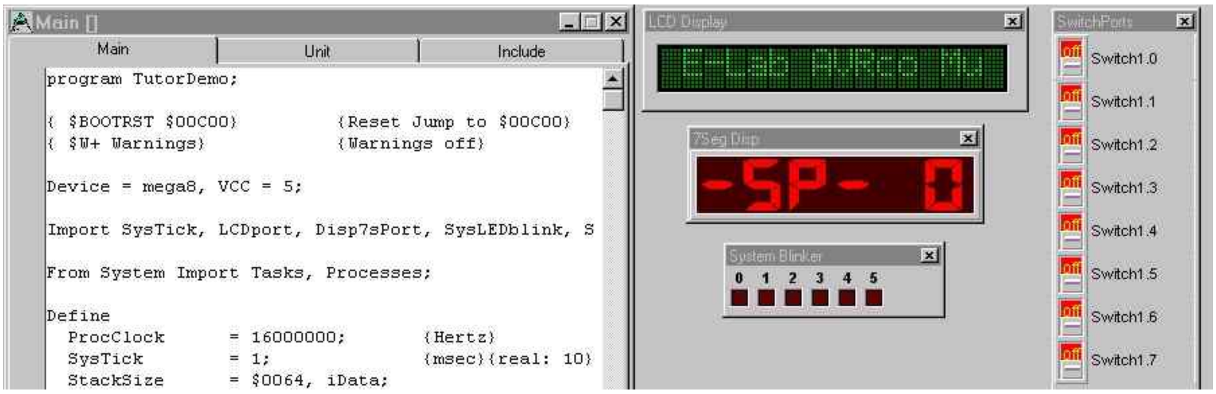
The following paragraph is an exerpt from the AVRco manual: *Introduction to Multitasking*:

> *With a socalled Embedded Application (Single-Chip application) often there is the problem, that several jobs should be done at the same time. For example the characters of a serial interface should be fetched, checked and perhaps they should be converted from hex into an integer.*
> *At the same time ports should be watched by limit-switches or a LED should flash. Additional a measurement value should be gathered by a poti and this value should be passed as a control output to an external controller. And the controller should calculate an output value in a fixed time grid.*
> *So the programmer has the problem with all these targets to do all things concurrently. The programmer is in the difficult situation to watch several processes at the same time, whereby he must pay attention, that all functions have to run* ***concurrent and independent.****.*
> *With simple time-loops etc. this problem can not be solved any more, maybe with tricks, which make the program nonelastic and bovine.*
> *So a solution is needed, which makes it possible to distribute the jobs, that they often get a chance as far as possible, but do not block other jobs. Such a system is called* ***Multi-Tasking****, whereby task is a job/assignment.*

To demonstrate the advantages of Multi-Tasking in this tutorial I use features that are well visible in the simulator. They represent functions like the above mentioned in a real application.

The demo uses a 1*16 LCD to display a long message (43 character string) as a moving message.
A 7-segment-display shows simultaneously the start-index of the actual displayed substring.
Furthermore there are 6 LEDs that are periodically switched on and off.
These are the "background jobs" that have to be done concurrently to the "main-job".
The "main job" reads a switch and toggles a state variable that determins the desired LED state (flash/off/on).

Open the *Project Management* and load the *TutorDemo* project!
Start the simulator:



```
program TutorDemo;

{ $BOOTRST $00C00}          {Reset Jump to $00C00}
{ $W+ Warnings}             {Warnings off}

Device = mega8, VCC = 5;

Import SysTick, LCDport, Disp7sPort, SysLEDblink, S

From System Import Tasks, Processes;

Define
  ProcClock      = 16000000;       {Hertz}
  SysTick        = 1;              {msec}{real: 10}
  StackSize      = $0064, iData;
```

Run it (F9) and try *Switch1.0* (remember: not too fast).
Have a look at the "*Main Program*": it only checks the switch and sets the state of the LEDs.
The moving message, the "-<u>SP</u>-Display" (<u>S</u>tring<u>P</u>ointer) and the LED on/off sequence is done without any statement in "main".

Imagine the big advantage for you:
you build and test a single part of your application. Let's say the moving message.
As far as this part is working you define it as a background task and free your main scope of this burdon.

Or the "*System Blinker*":
no matter where you change the *LEDstate* to *FlashOn* in your program –
the flashing is done by the background process. You do not have to worry about it anymore.
And you do not even need to define a task for it. The *System Blinker* is part of the scheduler and must only be imorted (the scheduler is the "supervisor" that handles the task switching).
But be aware:
this example shows only "low priority tasks". It may take some milliseconds longer to accept a switch or to toggle a LED - this does not matter all all in this example.
As far as you specify "realtime-jobs" as tasks (or processes) and must guarantee a certain time to react, Multi-Tasking becomes high sophisticated. Such systems need a complete understanding of the underlaying Multi-Tasking techniques including priorities, semaphores, pipes etc.

2.9 Useful Links

E-LAB Homepage:	http://www.e-lab.de/index_en.html
AVRco:	http://www.e-lab.de/AVRco/index_en.html
	includes links to Pascal Tutorials
	Programmers: see Hardware / Programmers
Mini Boards:	http://www.e-lab.de/diverse/components_en.html
Datasheets:	http://www.e-lab.de/AVRco/avr_sheets.html
Forum:	http://www.e-lab.de/phpBB2/

English Forum: http://www.avrfreaks.net/
Controller Manuals: see Devices

Do-it-yourself programmer and programming software:
http://www.lancos.com/prog.html
http://users.skynet.be/jiwan/Electronique/English/AVR%20Prog.htm
http://www.myplace.nu/avr/yaap/index.htm
http://s-huehn.de/elektronik/avr-prog/avr-prog.htm (german)
http://ln.com.ua/~real/avreal/index_e.html (command line program)

WARNING:

there are some very simple parallel programmers that consist only of a few resistors. These are NOT recommended for a regulary use. Bugs in your hardware could easily destroy your parallel port of the PC and they imply other restrictions.
But they can sometimes be useful to check whether you have a programming hardware or a programming software problem.

A lot of different links to all subjects can be found at

http://www.mikrocontroller.net/links.en.htm

2.10 Appendix

2.10.1 The AVRco Versions

E-LAB offers two versions of the AVRco:

- the Standard Version
  the Standard Version supports all popular AVR controllers and comes with drivers for near all of the builtin functions ("*On Chip Drivers*") and a huge amount of drivers for additional hardware ("*Soft Drivers*").
  It includes also a V24/RS232 programmer "ISP-V24" which supports SPI and JTAG programming.

- the Professional Version
  in addition to the Standard version, the Professional Version supports advanced features like support for:
  Units, Graphic LCDs, File System, IP Stacks (UDP and TCP), Heap and JTAG Debugging. It includes an USB programmer "ISP-USB" which supports SPI and JTAG programming and also JTAG debuggung.

As mentioned above, there is the free Demo Version on the E-LAB homepage. The only difference between the Standard Version and the Demo is the limitation of the Demo to 4k code size. Besides this, it comes with the full functionality of the Standard Version.

2.10.2 Source File Tutor01.pas

```
program Tutor01;

{ $BOOTRST $00C00}         {Reset Jump to $00C00}
{$NOSHADOW}
{ $W+ Warnings}            {Warnings off}

Device = mega8, VCC=5;

Import SysTick, LCDport, MatrixPort;

From System Import ;
```

```
Define
        ProcClock   = 8000000;        {Hertz}
        SysTick     = 10;             {msec}
        StackSize   = $0064, iData;
        FrameSize   = $0064, iData;
        LCDport     = PortD;
        LCDtype     = 44780;
        LCDrows     = 2;              {rows}
        LCDcolumns  = 16;             {columns per line}
        MatrixRow   = PortC, 0;       {use PortC, start with bit0}
        MatrixCol   = PinC, 4;        {use PinC, start with bit4}
        MatrixType  = 4, 2;           {4 Rows at PortC, 2 Columns at PinC}

Implementation

{$IDATA}

{--------------------------------------------------------------}
{ Type Declarations }

type
  t_KeySet          = BitSet of Keys;

{--------------------------------------------------------------}
{ Const Declarations }
const
  text : array [1..2,0..5] of string[16] =
          (('                ',
             '  E-LAB  AVRco  ',
             'Pascal  Compiler',
             '    Tutorial    ',
             '1st  Application',
             ' ??? Line 1 ??? '),
           ('                ',
             '  the 2nd Line  ',
             ' could  contain ',
             '   something    ',
             ' defined by you ',
             ' ??? Line 2 ??? '));

{--------------------------------------------------------------}
{ Var Declarations }
{$IDATA}
var
  l1, l2, l1a, l2a     : Byte;
  key                  : t_KeySet;
  Bkey [@key]          : Byte;

{--------------------------------------------------------------}
{ functions }
```

```
{---------------------------------------------------------------}
{ Main Program }
{$IDATA}

begin

  EnableInts;
  loop

  key := ReadKeyBoard;
  Bkey := Bkey AND $AA;             {1st line}


  case bkey of
    $80: l1 := 1;
       |
    $20: l1 := 2;
       |
    $08: l1 := 3;
       |
    $02: l1 := 4;
       |
    $00: l1 := 0;
       |
    else l1 := 5;                   {>1 key}
  endcase;

  key := ReadKeyBoard;
  Bkey := Bkey AND $55;             {2nd line}
  case Bkey of
    $40: l2 := 1;
       |
    $10: l2 := 2;
       |
    $04: l2 := 3;
       |
    $01: l2 := 4;
       |
    $00: l2 := 0;
       |
    else l2 := 5;                   {>1 key}
  endcase;

  if l1 <> l1a                      {new text ?}
  then
    l1a := l1;
    LCDxy(0,0);
    write (LCDout,Text[1,l1]);
  endif;


  if l2 <> l2a                      {new text ?}
  then
    l2a := l2;
    LCDxy(0,1);
    write (LCDout,Text[2,l2]);
  endif;

  endloop;
end Tutor01.
```

2.10.3 The Mega8 Konfiguration Bytes

(default is internal oscillator at 1 MHz)

Mega 8	default			>= 8 MHz	
Bit Name	binary value	P / U (Un/Programmed)		binary value	P / U (Un/Programmed)
BootLock12	1	U		1	U
BootLock11	1	U		1	U
BootLock02	1	U		1	U
BootLock01	1	U		1	U
Lock2	1	U		1	U
Lock1	1	U		1	U
RSTDISBL	1	U		1	U
WDTON	1	U		1	U
SPIEN	0	P		0	P
CKOPT	1	U		**0**	**P**
EESAVE	1	U		1	U
BOOTSZ1	0	P		0	P
BOOTSZ0	0	P		0	P
BOOTRST	1	U		1	U
BODLEVEL	1	U		1	U
BODEN	1	U		1	U
SUT1	1	U		1	U
SUT0	0	P		**1**	**U**
CKSEL3	0	P		**1**	**U**
CKSEL2	0	P		**1**	**U**
CKSEL1	0	P		**1**	**U**
CKSEL0	1	U		1	U

3 Editor PED32

3.1 Overview

3.1.1 Introduction

PED32 is a so called IDE (Integrated Development Environment).

PED32 (Programmers Editor for WIN32) serves as a platform for several Compiler, Assembler and Debugger.

PED32 contains a comfortable Project-Administration rarely to find.

PED32 contains a Multi-Window Editor (MDI) comparable with that from Borland Delphi.

PED32 with correct settings, displays all errors found by the Compiler, Assembler and Linker.

PED32 in case of error the file is loaded, the cursor is positioned to the erroneous position, the line is highlighted

PED32 uses an error list which can be scrolled and clicked by the mouse to highlight the corresponding line

PED32 samples project dependant the time used to build and finish the application.

PED32 has practically unlimited file size.

PED32 recognizes the files changed by the Compiler etc. and reloads them.

PED32 font sizes colors etc can be freely defined by the user.

PED32 has a configurable Syntax-Highlight.

PED32 supports syntax and context sensitive Help-Files.

PED32 supports Syntax-Highlight in font attributes and color

PED32 is widely configurable and therefore adaptable to many tools.

PED32 all this features together build a tool which is competitive to many others and mostly better

3.2 Projects

In opposite to many other IDEs and editors **PED32** project-related and less file-related. That means, primarily it is worked with a project, not with seperate files.

A **project** contains several source-files, e.g. the **MainFile**, the **ProjektPfad**, the used **programming-time** as well as a project-related control-file, the socalled **Control**.

For a creation of a new project by the menu item **Project/Edit Project** and the dialog **Project Admin** or the corresponding SpeedButton an appropriate control instruction must be specified in addition to project-specific definitions like project name, MainFile, project path etc.. The dialog offers a selection of the existing controls.

If all specifications are correct and complete, so, if not existing, in case of Control=PICpas a MainFile is automatically created, which can be used as a template. With the AVR here the Application Wizard is called, which helps for the creation of the program template.

Normally further definitions are not required, because the used control contains all further instructions for the IDE.

3.3 Controls

The control instruction **Control** is essential to work with a project, see above **projects**. In the main Control is a batch description.

If the control is created correctly and completely by the menu item **System/System Admin** or the dialog **System Admin**, normally it must not be changed any more.

Almost every point or line in this dialog has an according button in the upper button bar, the socalled SpeedButton. The make-line and the simulator/debugger can also be called by CTRL + F9 or F9.

Every line consists of instructions, normally EXE-files, which are processed from left to right.
Further it is possible that there are numbers in a line, seperated by a seperator-character „|“ from the text and other numbers. So other lines are included within the dialog. Pay attention to recursions!!

Example: The control PICpas is a control instruction for the E-LAB Pascal Compiler for the MicroChip PIC processor family. For all projects, which are created with PICs and the Pascal Compiler, only the Control PICpas must be specified for the project creation, and it is possible to edit, compile, debug etc. immediatly.

Here it must be said that the creation of a new control must be considered carefully. Besides there must be an intensive deal with the according dialogs. The later resulting automation needs some preparatory work.

3.4 Syntax

The syntax definition (mind you compiler-language) normally is related to a certain control, and so it can be found in a sub-dialog of System Admin. Under some circumstances it makes sense to have several Pascal „languages“, for example for several different Pascal compilers. These must be created seperatly and included in the respective control under syntax.

This creation of a respective language-syntax list is not a must. It only serves to emphasize certain syntax elements within the editor.

3.5 Menu

Overview of the menu-items of PED32

3.5.1 File Menu

New		Open a new file.
Open		Calls the file-open dialog.
Open Mainfile		Opens the MainFile of the project
Open Mapfile		Opens the MapFile of the project
Save	**Ctrl + S**	Stores the actual editor window into the corresponding file.
Save As		Stores the actual editor window with a new name
Close File		Closes the actual editor window with possible file security
History		Shows the last 10 workked projects in time series. By clicking on the chosed project is loaded.
Insert File		Reads a file at the actual caret position.
Save Block		Writes the highlighted block as a file.
Print		Prints the actual file (window) or block.
Exit	**Alt + F4**	Close PED32.

3.5.2 Edit Menu

Undo	**Ctrl + Z**	Undo the last change (Undo)
Redo	**Shift + Ctrl + Z**	Undo the last undo (Redo)
Cut	**Ctrl + X**	Cut out a highlighted block and copy it into the notebook
Copy	**Ctrl + C**	Copy highlighted block into the notebook
Paste	**Ctrl + V**	Copy notebbok to the caret position
Delete		Delete symbol right of the caret or the whole block
Select All		Mark the block from begin to the end of the file

3.5.3 Search Menu

Find	**Ctrl + F**	Search a word (at the caret)
Replace	**Ctrl + R**	Search a word (at the caret) and replace it
Find Next	**F3**	Repeat last search/replace
Replace Tabs by Softtabs		Replace hard tabs with softtabs
Replace Tabs by Spaces		Replace hard tabs with spaces
Goto Line	**F4**	Set caret to the line number x
Clear all Markers		Sets all markers back

3.5.4 Project Menu

Load Project	Opens dialog to load a project
Edit Project	Opens dialog to create a new or change an existing project
Project Information	Displays the actual state of the project
Project Options	Edit compiler switches for Conditional Compile and Project Unit Paths

3.5.5 System Menu

System Options	Opens a dialog to hide the compiler-window and define System Unit Paths
System Admin	Special definitions of the controls (control-instructions for div. Compilers)

3.5.6 IDE Menu

General Options	Opens the dialog for the font- and color definition
Tabs	Define the TABs
Popup delay	Delay of the syntax fasthelp
Edit Keyboard Macros	Opens dialog to edit Keyboard Macros

3.5.7 Window Menu

Arrange Icons	New order of editor windows, decreased to icons
Minimize all	Decrease all editor windows to icon size
Window ...	Do a certain editor window in the foreground

3.5.8 Info Menu

Help IDE	**Ctrl + F1**	Call the IDE or editor help in the context
Help Syntax	**F1**	Call the compiler or syntax help in the context
Info IDE		Call the overview of the the IDE or editor help
Info Syntax		Call the overview of the compiler or syntax help
About		Display of the actual version of PED32

3.6 Dialogs

Overview of the Dialogs of PED32

3.6.1 Project Admin

Serves to load or change an existing project, as well as to create a new project. Display of the used time of the seperate projects.

Subordinate dialogs:

Project Path	Serves to define the path of the chosed project
Main File	Serves for the selection of the Main Files .

3.6.2 Project Options

Compiler switches for the conditional compile can be defined here. These are passed to the compiler, which treats them like a **{$DEFINE *Label*}**. Also project dependant unitpaths are defined here.

3.6.3 Project Info

Display of the state of the actual project, as well as general datas like RAM and ROM consumption and allocation. Position and size of stacks and frames.

3.6.4 General Options

Serves to define the font/character size, as well as the colors for the normal text and ist background, highlighted text and background, error text and ist background and comments and strings. Also the general behaviour of the editor and IDE (Indent, Backup, Fasthelp) are defined here.

3.6.5 Macro Editor

Serves to create keyboard macros. So complete code-blocks can be created and edited. These blocks are inserted into the current text by hotkeys.

3.6.6 Character Table

Serves to insert special characters into the source text. In addition it is able to establish the hex- or decimal value of a character.

3.6.7 System Admin

Serves to create the **controls** for the seperate tools, for example compiler, assembler etc. Further the syntax of the used languages is described, which is required for the syntax-highlighting of the editor. The error evaluation of the tools must also be declarated.

Subordinate dialogs:

Syntax + FileMasks	Serves to chose the language, e.g. Pascal, the limiter for comments, as well as the file masks, e.g. *.PAS, *.ASM
Error Definitions	Declaration of the error files and their internal structure
Help File	Name and path of the compiler related Help File

3.6.8 File Open

Seves to chose and load the desired files inclusive their path

3.6.9 File Save As

Seves to store the actual file with a name and/or path

3.6.10 Print

Serves to chose a printer, ist definition and print of the actual editor file.

3.6.11 Find

Serves to define the search options and search start

3.6.12 Replace

Serves to define the search/replace options and search/replace start.

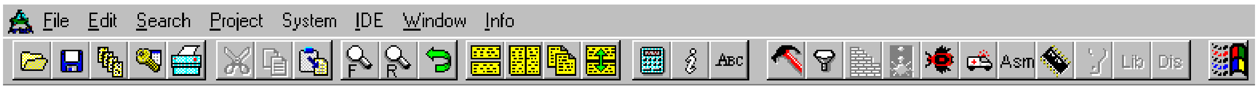
3.6.13 Goto Line

Positions the caret to the specified line.

3.6.14 TabSize

Defines the length of the tabulator.

3.7 SpeedButtons



Overview of the Speedbuttons of PED32

3.7.1 FileOpen

Calls the file-open dialog.

3.7.2 Save

Ctrl + S Stores the actual editor-window into the corresponding file

3.7.3 Project administration

Opens the project dialog

3.7.4 Application Wizard

Starts the program generator

3.7.5 Printer Dialog

Opens the printer dialog

3.7.6 Cut

Ctrl + X Cuts out a highlighted block and copies it into the notebook

3.7.7 Copy

Ctrl + C Copies the highlighted block into the notebook

3.7.8 Paste

Ctrl + V Copies the content of the notebook to the caret position

3.7.9 Find

Ctrl + F Opens the search dialog

3.7.10 Replace

Ctrl + R Opens the search/replace dialog

3.7.11 Undo

Ctrl + Z Undo the last text change

3.7.12 Tile horizontal

Devides the existing PED32 window even into the opened editors. The windows are one below the other.

3.7.13 Tile vertical

Devides the existing PED32 window even into the opened editors. The windows are side by side.

3.7.14 Cascade

The opened editors are located descending in a row

3.7.15 Split Window

The actual editor window is devided into two wondows. Both windows contain the same file, but there are different positions within the file.
So it is possible to edit at two different positions in the file at the same time. It is very helpful for some operations!

3.7.16 Calculator

opens the calculator

3.7.17 Project Info

Displays the state of the project

3.7.18 Alphabet

Opens the alphabet dialog

3.7.19 Make

Ctrl + F9

The command-line, which is in the defined control in the make-line is processed. Then eventual errors are evaluated.

3.7.20 Compile

The command-line, which is in the defined control in the compile-line is processed. Then eventual errors are evaluated.

3.7.21 Link

The command-line, which is in the defined control in the link-line is processed. Then eventual errors are evaluated.

3.7.22 Post Processor

The command-line, which is in the defined control in the PostProc-line is processed. Then eventual errors are evaluated.

3.7.23 Debugger

The command-line, which is in the defined control in the debugger-line is processed. Then eventual errors are evaluated.

3.7.24 Simulator

F9

The command-line, which is in the defined control in the simulator-line is processed. Then eventual errors are evaluated.

3.7.25 Assembler

The command-line, which is in the defined control in the assembler-line is processed. Then eventual errors are evaluated.

3.7.26 RomSim/Prommer

The command-line, which is in the defined control in the RomSim-line is processed. No error evaluation.

3.7.27 Tool

At the moment it has no function

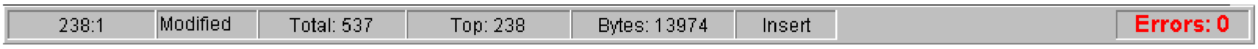
3.7.28 Librarian

The command-line, which is in the defined control in the library-line is processed. Then eventual errors are evaluated.

3.7.29 DisAssembler

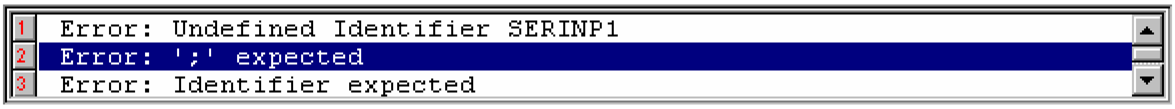
The command-line, which is in the defined control in the DisAsm-line is processed. No error evaluation.

3.8 State Bar



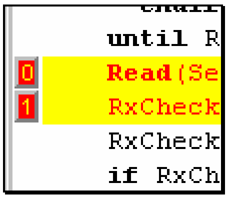
238:1	Modified	Total: 537	Top: 238	Bytes: 13974	Insert	Errors: 0
Cursor Pos line column	text changed	total linecount	upper most line in the window	file size	Insert or overwrite mode	number of last occured errors

3.9 Error window



```
1  Error: Undefined Identifier SERINP1
2  Error: ';' expected
3  Error: Identifier expected
```

If there are errors or warnings enabled in a compiler, so an error-window is opened at the bottom of the editor. It contains all errors and possible warnings with an explanation. The editor scrolls to the according position in the source by a doubleclick on one of these lines.



```
    until R
0   Read(Se
1   RxCheck
    RxCheck
    if RxCh
```

The faulty lines are marked with the corresponding color in the source, at the left there are the error-numbers. A click on this number scrolls the error-window at the bottom, so that the according error and its explanation get visible. This line additionally gets a HighLight.
If this source is essentially changed or deleted, the error highlight and the number in the source will be deleted.

3.10 HotKeys and ShortCuts = Keyboard commands

3.10.1 IDE and Syntax Help

Ctrl + F1	Call help of the IDE. Help is depending on the actual dialog or call Focus
F1	Syntax-Help. Help is depending on the word below the caret.

3.10.2 File and window operations

Ctrl + S	Stores the actual editor window
Ctrl + TAB	Switch over to the next editor window

3.10.3 Move caret

←	Caret 1 character to left
→	Caret 1 character to right
↑	Caret 1 line up
↓	Caret 1 line down
Ctrl + ←	Caret 1 word to left
Ctrl + →	Caret 1 word to right
HOME	Caret to begin of the line
END	Caret to end of the line
PageUp	1 screenpage in direction to begin of file
PageDown	1 screenpage in direction to end of file
Ctrl + PageUp	Caret to the actual screen page begin
Ctrl + PageDown	Caret to the actual screen page end
Ctrl + HOME	Caret to begin of file
Ctrl + END	Caret to end of file
F4	Set caret to line number x
Ctrl + n	Jump to the marker “n“ (0..9)
Shift + Ctrl + n	Set marker “n“ (0..9)

3.10.4 Edit

Insert	Insert/overwrite on/off
Delete	Delete character right of the caret or the whole block
BackSpace	Delete character left of thecaret or the whole block
Ctrl + Enter	Insert empty line at caret position

Ctrl + N	Insert empty line at caret position
Ctrl + Y	Delete line completely at caret position
Ctrl + T	Delete word completely right of the caret position
Ctrl + P	Insert control character at caret position
Ctrl + Z	Undo last change
Alt + BackSpace	Undo last change
Shift + Ctrl + Z	Undo last undo

3.10.5 Search/replace

Ctrl + F	Search word (at caret position)
Ctrl + R	Search word (at caret position) and replace it
F3	Repeat search or replace

3.10.6 Caret block commands

Shift + ←	caret 1 character to left with blockextension/abridgement
Shift + →	caret 1 character to right with blockextension/abridgement
Shift + ↑	caret 1 line up with blockextension/abridgement
Shift + ↓	caret 1 line down with blockextension/abridgement
Shift + Ctrl + ←	caret 1 word to left with blockextension/abridgement
Shift + Ctrl + →	caret 1 word to right with blockextension/abridgement
Shift + HOME	caret to begin of line with blockextension/abridgement
Shift + END	caret to end of line with blockextension/abridgement
Shift + PageUp	1 screenpage in direction to begin of file with blockextension/abridgement
Shift + PageDown	1 screenpage in direction to begin of file with blockextension/abridgement
Shift + Ctrl + PageUp	caret to the actual screenpage Anfang with blockextension/abridgement
Shift + Ctrl + PageDown	caret to the actual screenpage Ende with blockextension/abridgement
Shift + Ctrl + HOME	caret to begin of file with blockextension/abridgement
Shift + Ctrl + END	caret to begin of file with blockextension/abridgement

3.10.7 Edit block

Shift + Delete	Cut out block, copy of the block into Windows-notebook
Ctrl + X	Cut out block, copy of the block into Windows-notebook
Ctrl + Insert	Copy block into Windows-notebook. Block unchanged.
Ctrl + C	Copy block into Windows-notebook. Block unchanged.
Shift + Insert	Copy block out of Windows-notebook to caret position.
Ctrl + V	Copy block out of Windows-notebook to caret position.
Shift + Ctrl + I	Shift block to right
Shift + Ctrl +U	Shift block to left.

3.10.8 Diverse

Ctrl + F9	Start Make Tool . Start Compiler, Assembler (Linker)
F9	Simulator/Debugger

3.10.9 Keyboard Macros

With the dialog **Edit Keyboard Macros** the user is able to create textblocks, e.g. program sequences, which can be inserted to a according ShortCut or HotKey at the cursor position in the text.

3.11 Projects

3.11.1 Working with projects

In opposite to many other IDEs and editors **PED32** project-related and less file-related. That means, primarily it is worked with a project, not with seperate files. The IDE knows a project, if it is created once until it is deleted. The programmer must not maintain directories or according tools. Load the project and work.

A link from e.g. Pascal-files with PED32 is not necessary and does not make sense. Only a link from PED32 with *.ppro files makes sense.

A **project** contains several source-files, e.g. the **MainFile**, the **ProjektPfad**, the used **programming-time** as well as a project-related control-file, the socalled **Control**.

For a creation of a new project by the menu item **Project/Edit Project** and the dialog **Project Admin** or the corresponding SpeedButton an appropriate control instruction must be specified in addition to project-specific definitionss like project name, MainFile, project path etc.. The dialog offers a selection of the existing controls.

If all specifications are correct and complete, so, if not existing, a MainFile is automatically created, which can be used as a template. The template is a normal text-file. For creation of the respective **Control** the name and the path of ththis template must be registered. If an application wizzard program generator) is specified in the control, so the template is started instead of the program generator.

Normally further definitions are not required, because the used control contains all further instructions for the IDE.

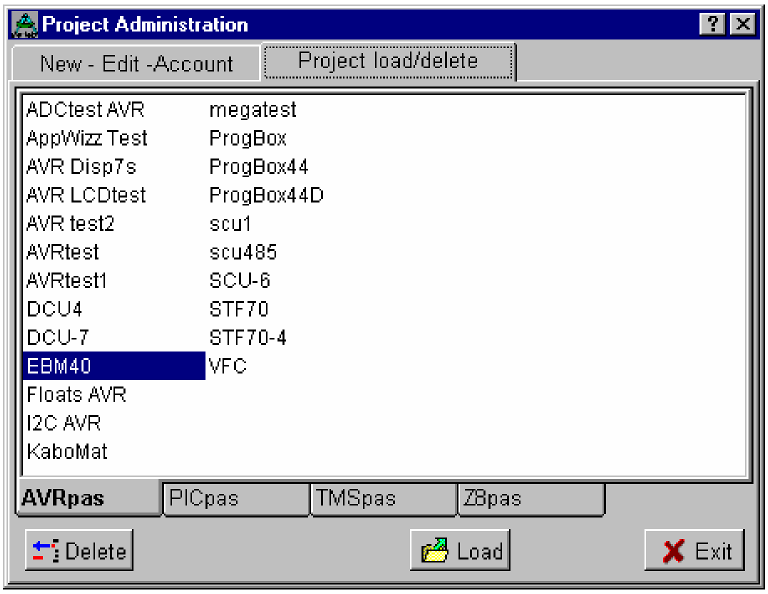
Like already mentioned, the IDE PED32 stores the files of all projects. A call of an existing project is limited on two clicks in future. It is possible to process seperate files, but the actual intention is the file processing within a project.

After a certain adaption time from a normal editor to projects the user learns fast to cherish the advantages.

So the low additional expenditure for the creation of a project parameter is well-accepted.

3.11.1.1 Load Project

To load an existing project the dialog **Project Admin** must be opened. This can be made by the menu-item **Project .. Load Project** or by the SpeedButton . So the following dialog is opened. The dialog page **Project load/delete** is selected by a click on the respective tab.



ADCtest AVR
AppWizz Test
AVR Disp7s
AVR LCDtest
AVR test2
AVRtest
AVRtest1
DCU4
DCU-7
EBM40
Floats AVR
I2C AVR
KaboMat
megatest
ProgBox
ProgBox44
ProgBox44D
scu1
scu485
SCU-6
STF70
STF70-4
VFC

With a doubleclick on the chosed entry or a click on load button the selected project is loaded.

A further possibility is in the file-menu with the menu-item **History.** Here a list of the last processed projects in order of time is diplayed. Also load a chosed project with a click .

Obsolete projects are deleted by the **Delete** button.

The projects assigned to the seperate controls (compiler) are displayed on seperate pages.

After the selection of the chosed project all project-relevant files are loaded out of the file **PED32.ini**. These files contain a.o. the home directory of the project and the used control.

PED32 now switches the **Current Directory** in windows to this directory. All further file operations within the IDE which are made without path specification, now are in this directory.

Next the IDE loads the project control file **xxx.ppro**. This file now contains all further project files as well as e.g. the last opened files.

Now the according control is loaded and next the last opened files. At last is checked, if an error-file is existing. If this is the case, it is opened and evaluated. So the loading process is finished and it can be worked.

Starting or loading by DragAndDrop does not make sense and is not provided.

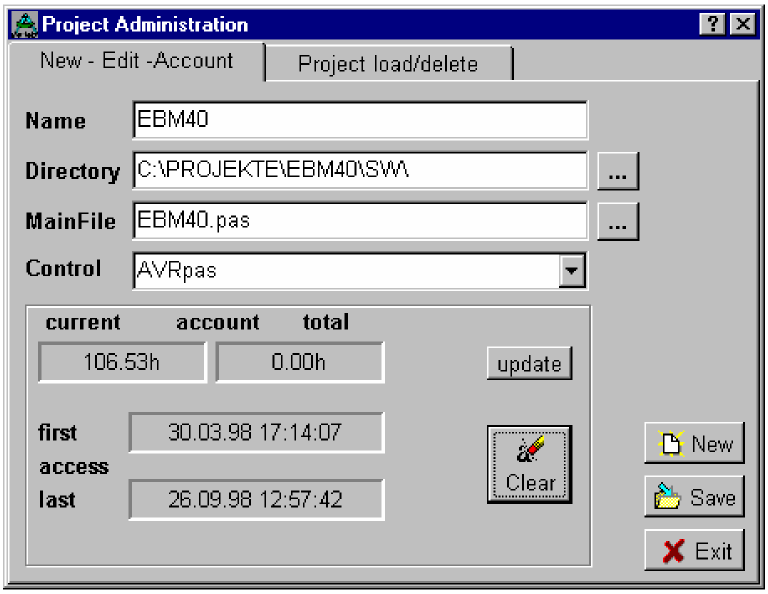
A link from *.ppro files to PED32 is possible. If this link is made once, it is possible to start the IDE PED32 at any time within the explorer or similar programs by a doubleclick on a file with the format xxx.ppro. This now is loading the clicked project.

A doublestart of PED32 is not possible, because both programs must maintain the same INI-files in competition. Sure there will be strange results.

The editor in PED32 watches all loaded files. If a loaded file is changed from outer side, for example by an editor, the IDE gives the information that the file has been changed, and the inquiry, if it should be loaded again. The re-load is automtically, if for example the compiler generates a changed assembler source.

3.11.1.2 Edit/New Project

To change an existing project or to create a new one, the dialog **Project Admin** must be opened. This can be made by the menu-item **Project .. Edit Project** or by the SpeedButton . Then the following dialog is opened. The dialog page **New-Edit-Account** is selected by a click on the respective tab.



Name: EBM40
Directory: C:\PROJEKTE\EBM40\SW\
MainFile: EBM40.pas
Control: AVRpas

current	account	total
106.53h		0.00h

first access: 30.03.98 17:14:07
last: 26.09.98 12:57:42

If a new project should be created, the **New** button must be clicked. The further procedure is identic for New and Edit.

In the edit window **Name** the chosed project name is registered. This name is also used as project filename. But it can contain all characters, which are allowed in WIN95 etc, e.g. space.

By an edit of the window **Directory** the working directory is specified.
A doubleclick on this window or the next button opens the dialog **Project Path**. So it is possible to chose an existing directory as working directory.
By an edit of the window **MainFile** the **main-file** is specified.

Another possibility to open the dialog **MainFile** is a doubleclick on this window or a click on the next button. So an already existing file in the working directory is declared to a MainFile.

If the registered MainFile is not existing, so, if specified in the chosed control, a template-file with the name *MainFile* is copied into working directory, if there is no program generator provided.

The window **Control** displays the actual chosed control. By a click on the right arrow-button of this window the actual known controls are offered for selection. The desired control is selected by a click. A new control, if necessary, first must be created by the menu **System|System Admin** and the dialog **SysAdmin**. If a control, for example PICco32, is created once, it can always be used for all projects with the Pascal-compiler in future.

Seperated by a frame, the time-relevant project files are displayed left at the bottom. These are access files, devided in project creation (**first access**) and last call (**last access**).

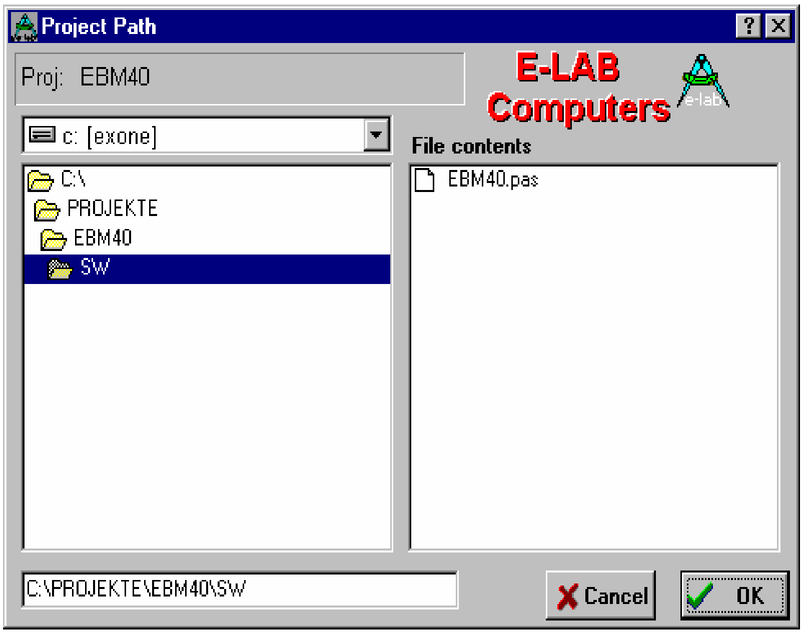
Above the used process times are displayed. **Total account** indicates the last accounted time effort. **Current account** indicates the current time after the last account. The sum of both times is the actual used time for the project.

If there is an intermediate account, you must click the **Update** button. Now current account is added with the total account and current account is set to zero. Total account now contains the absolute time.

The times can be reset by **Clear**. So the access files are set to the actual date and time of day. S the project is quasi started new.

All changes must be confirmed and saved with **Save**. If this is not made, so if the dialog should be closed, it is inquired if a save of the change is desired. If the inquiry is negated, the changes get invalid.

3.11.1.3 Project Path

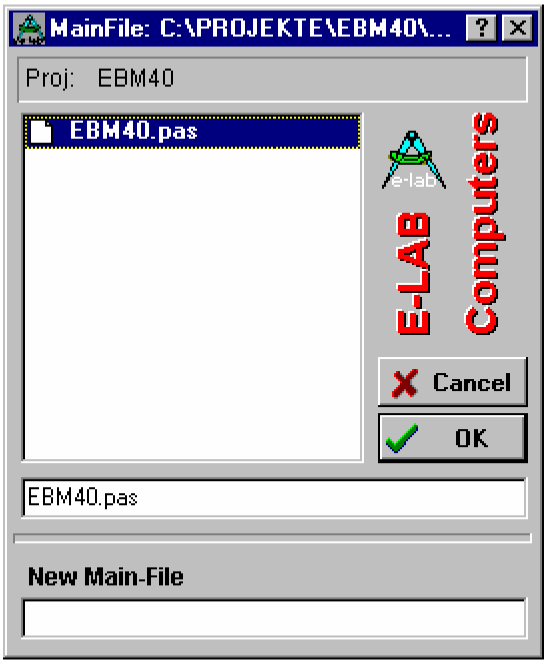


Sub-dialog of the dialog **Edit/New Project** .

For a new creation or an edit of a project, a home- or working-directory must be assigned to the project. By a doubleclick in the dialog **Edit/New Project** on the edit field **Directory** or dthe right button the dialog **Project Path** is opened.

Here it is possible to declare an existing directory to a working directory of the chosed project. The right half contains the relevant files of this directory for information. For selection pay attention that desired path appears completely at the bottom in the field. It is able to edit this field, too. If the changed path is not existing, it is inquired, if the directory should be created new.

3.11.1.4 MainFile



Sub-dialog of the dialog **Edit/New Project** .

For a new creation or edit of a project a mainfile must be assigned to the project. For a new creation of a project the desired filename typed in the edit field **Main File** in the dialog **Edit/New Project**.

If the file is already existing, the dialog **Main File** can be opened by a doubleclick on the field or a click on the button.

Here an existing file can be declared to the MainFile of the chosed project. If no corresponding file is existing, so the name of the new mainfile must be typed in the bottom field. To be compatible with other tools, the name should correspond to the DOS conventions.

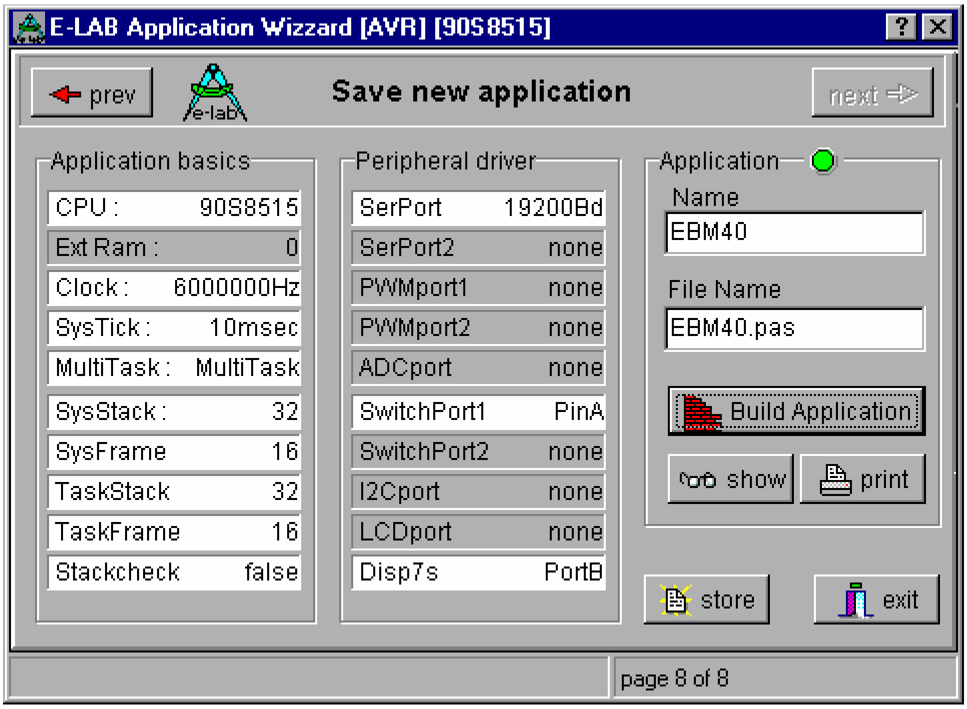
If a project is created once with the above mentioned dialogs, it can be called at any time.

In the working directory of the project there is created and maintained automatically a project control file. This file contains all project-related files. The filename of this file is "*ProjektName*.ppro". The structure is corresponding to a windows-ini-file and so it is a normal textfile. This textfile should only be changed in case of reperatures.

At the end of the total procedure (new creation or edit of a project) the IDE checks, if the typed project mainfile is existing. If this is not the case, so it is checked in the assigned control, of an application wizzard or a template has been specified and there is a corresponding procedure.

3.11.1.5 Application Wizard

If an Application Wizard is defined in the according control, it is called, if there is no mainfile existing in a new project. So it is possible to create comfortably a new application.



E-LAB Application Wizzard [AVR] [90S8515]

Save new application

Application basics

CPU :	90S8515
Ext Ram :	0
Clock :	6000000Hz
SysTick :	10msec
MultiTask :	MultiTask
SysStack :	32
SysFrame	16
TaskStack	32
TaskFrame	16
Stackcheck	false

Peripheral driver

SerPort	19200Bd
SerPort2	none
PWMport1	none
PWMport2	none
ADCport	none
SwitchPort1	PinA
SwitchPort2	none
I2Cport	none
LCDport	none
Disp7s	PortB

Application ●

Name: EBM40

File Name: EBM40.pas

page 8 of 8

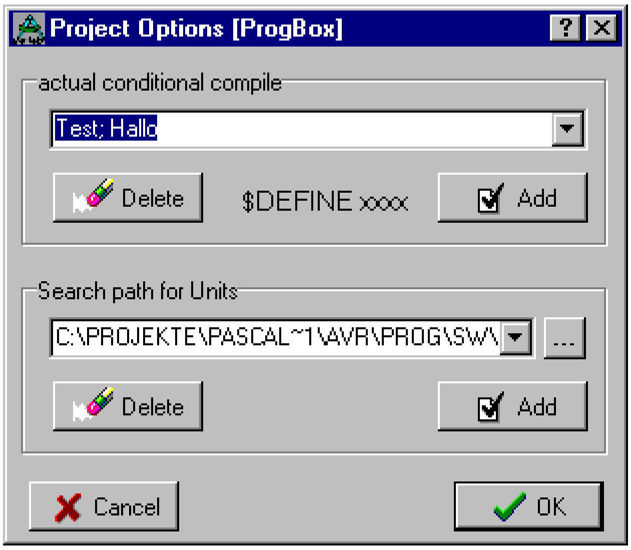
Example for an application expert

AppWizz for E-LAB ***AVRco***

3.11.1.6 Template

If there is no wizzard existing, but a template-file, so it is copied into the working directory with the name MainFile. The template-file can be any textfile with any content.

3.11.1.7 Project Options



Project Options [ProgBox]

actual conditional compile: Test; Hallo

$DEFINE xxxx

Search path for Units: C:\PROJEKTE\PASCAL~1\AVR\PROG\SW\

E-LAB Compiler of the serie ***Pascal-scm*** know the socalled Conditional Compile (see also Compiler databook). Such **$DEFINE** instructions can be created, edited and made valid with this dialog. The dialog contains all Defines, which were created for this project.

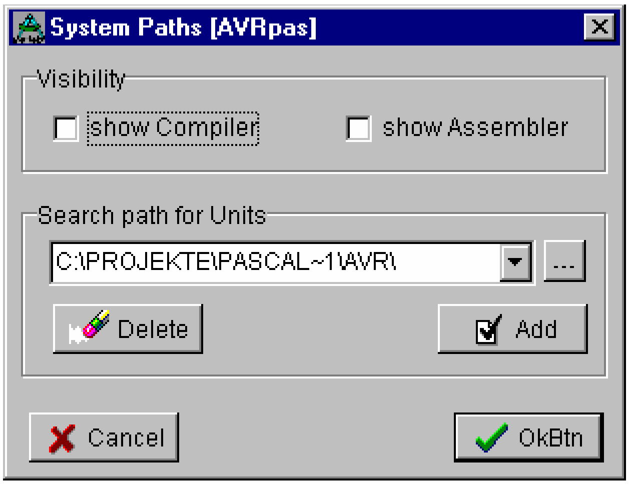
A new define is typed in the edit-field. Several values, which are valid at the same time must be devided with a semicolon. If the line is complete, it must be added by the add-button.
A list of all defines you can get by a click on the arrow button. By a further click on an arrow of the opened window an existing line is chosed. With the ok-button this definition is allocated to all further compile runs.

PED32 inserts this line into the xxx.ppro file, where it is processed by the compiler. The compiler interprets the seperate instructions of this line as if is in the source code. Out of the line **Test; Sample** you get →

```
{$DEFINE Test}
{$DEFINE Sample}
```

Project related Unit Search Paths can be defined in the lower edit field. All defined paths in the list are passed to the Compiler via the „xxx.ppro“ file. The Compiler must be able to interpret this lines.

3.11.1.8 System Options



System Paths [AVRpas]

Visibility

□ show Compiler   □ show Assembler

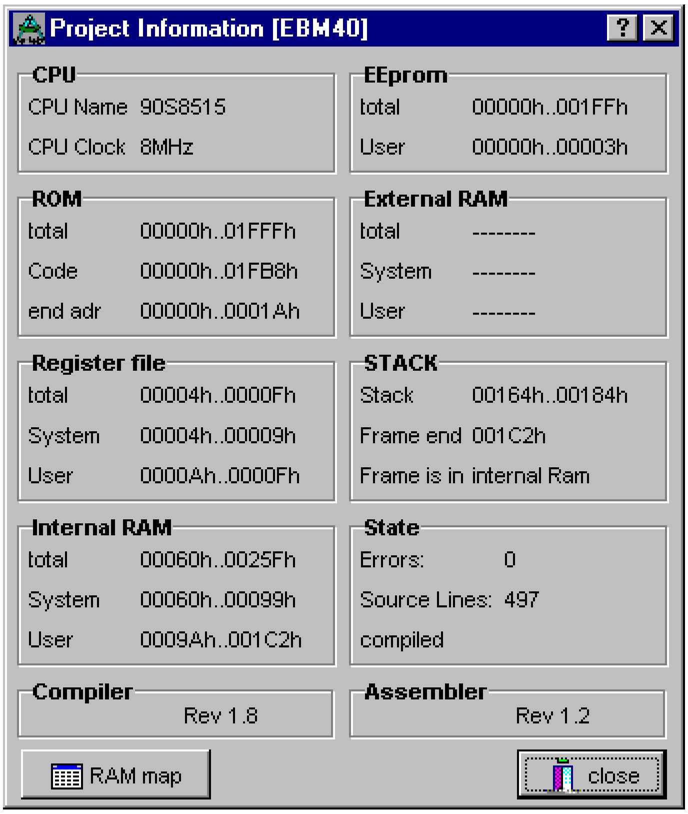
Search path for Units

C:\PROJEKTE\PASCAL~1\AVR\

Visibility
In case of a compiler or assembler run the check boxes define wether the tools are visible or not. If visible a window is opened and the processed lines are counted. This wastes time and the tools slow down.

Search path for Units
The search path for units are defined here and are passed to the compiler. All paths are relevant in the entire system. That means this type of paths are not project dependant but always in use.

3.11.1.9 Project Information



Project Information [EBM40]

CPU
CPU Name 90S8515
CPU Clock 8MHz

EEprom
total 00000h..001FFFh
User 00000h..00003h

ROM
total 00000h..01FFFh
Code 00000h..01FB8h
end adr 00000h..0001Ah

External RAM
total --------
System --------
User --------

Register file
total 00004h..0000Fh
System 00004h..00009h
User 0000Ah..0000Fh

STACK
Stack 00164h..00184h
Frame end 001C2h
Frame is in internal Ram

Internal RAM
total 00060h..0025Fh
System 00060h..00099h
User 0009Ah..001C2h

State
Errors: 0
Source Lines: 497
compiled

Compiler
Rev 1.8

Assembler
Rev 1.2

All relevant parameters of a project can be displayed and analysed by this dialog. So it is an exception that the long winded analysis of the assembler listing is necessary.

With the button RAM map you get additional information about the exact memory usage.

3.12 Controls

3.12.1 What is a control?

The control instruction **Control** is essential to work with a project, see above **projects**. In the main Control is a batch description.

If the control is created correctly and completely by the menu item **System/System Admin** or the dialog **System Admin**, normally it must not be changed any more.

Almost every point or line in this dialog has an according button in the upper button bar, the socalled SpeedButton. The make-line and the simulator/debugger can also be called by CTRL + F9 or F9.

Every line consists of instructions, normally EXE-files, which are processed from left to right.

Further it is possible that there are numbers in a line, seperated by a **seperator-character** „|" from the text and other numbers. So other lines are included within the dialog. Pay attention to recursions!!

Example: The control **PICpas** is a control instruction for the E-LAB Pascal Compiler for the MicroChip PIC processor family. For all projects, which are created with PICs and the Pascal Compiler, only the Control PICpas must be specified for the project creation, and it is possible to edit, compile, debug etc. immediatly.

This instruction (Control) is in the home directory of PED32 and has the name PED32.ini, all other already defined controls can also be found in this file. This file is structured like a standard windows ini-file and so it is a textfile. This file should only be changed in case of reperatures. PED32 maintains this file automatically.

Here it must be said that the creation of a new control must be considered carefully. Besides there must be an intensive deal with the according dialogs. The later resulting automation needs some preparatory work.

The seperate items of a control normally are socalled batch-instructions, which are executed, if the respective SpeedButton is clicked. If, for example the SpeedButton *Link* is clicked, the line linker of the control is processed. But it is not omplicitly necessary to call a linker, there can also be any operations, depending on the entry of this control.

Except the fixed link between the seperate SpeedButtons and the control entries there is no further link, that means every control entry can be defined freely.

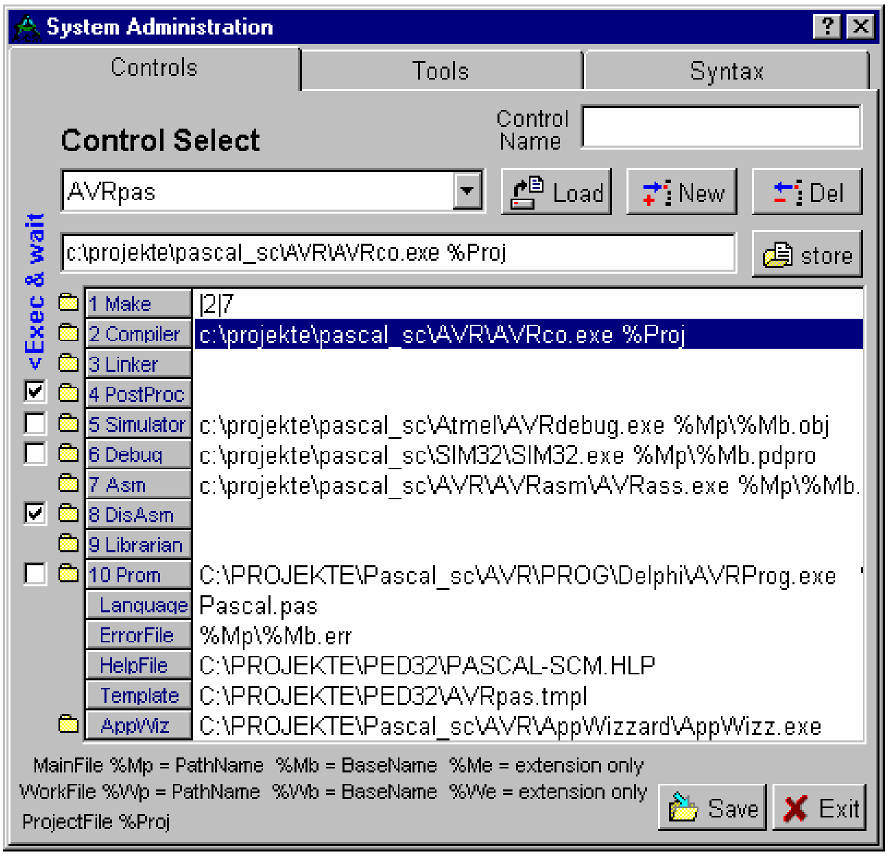
But some of the instructions (Make, Compiler, Linker, Assembler and Librarian) do an error check after their execution. Therefor it is searched for the errorfile within the working directory, its name and structure is also in the respective control.

Because a control knows only one errorfile and one structure, the called tools in this control all must have the same error file name and protocols, otherwise errors can not be evaluated.

The E-LAB Tools like for example PICco32 and PICasm32 fulfill this requirement.

3.12.2 Control Edit

To change an existing control or to create a new one the dialog **System Admin** must be opened. This is made by the menu item **System/System Admin**. After that the following dialog is opened. The dialog page **Controls** is chosed by a click on the respective tab.



Control Select

Control Name:

AVRpas

c:\projekte\pascal_sc\AVR\AVRco.exe %Proj

	🗀	1 Make	|2|7
	🗀	2 Compiler	c:\projekte\pascal_sc\AVR\AVRco.exe %Proj
	🗀	3 Linker	
☑	🗀	4 PostProc	
□	🗀	5 Simulator	c:\projekte\pascal_sc\Atmel\AVRdebug.exe %Mp\%Mb.obj
□	🗀	6 Debug	c:\projekte\pascal_sc\SIM32\SIM32.exe %Mp\%Mb.pdpro
	🗀	7 Asm	c:\projekte\pascal_sc\AVR\AVRasm\AVRass.exe %Mp\%Mb.
☑	🗀	8 DisAsm	
	🗀	9 Librarian	
□	🗀	10 Prom	C:\PROJEKTE\Pascal_sc\AVR\PROG\Delphi\AVRProg.exe '
		Language	Pascal.pas
		ErrorFile	%Mp\%Mb.err
		HelpFile	C:\PROJEKTE\PED32\PASCAL-SCM.HLP
		Template	C:\PROJEKTE\PED32\AVRpas.tmpl
	🗀	AppWiz	C:\PROJEKTE\Pascal_sc\AVR\AppWizzard\AppWizz.exe

MainFile %Mp = PathName %Mb = BaseName %Me = extension only
WorkFile %Wp = PathName %Wb = BaseName %We = extension only
ProjectFile %Proj

If an existing control should be **deleted**, so first it must be loaded by the load button and by a click on the select field. Now the loaded control can be deleted by the delete button.

The creation of a **new** control is identic with edit an existing control, expect that first a name must be specified.

This happens by a click on the new-button. Now an inputfield opens above the **new**-button. Here the name of the new control must be typed. If the name is complete, it is confirmed by a click on the **load**-button.

The new control now must be chosed out of the list **Control Select** and must be loaded by the **load**-button. The futher procedure is equal to the edit of a control and is described now.

Control Edit.
The control, which must be edited, is chosed out of the list **Control Select** and is loaded by the **load**-button. All already processed items of this control are obvious in the items list. Every singular item (Batch line) can be chosed by a click on the respective line.

The chosed term then appears in the **edit-window** above for further procedure. If here are changes, they must be confirmed and strored with the **Store** button. The changes are visible in the list immediatly.

If the changes made in the editfield should be invalid, so the sore button may not be clicked, but the next item in the list must be . So the changes are invalid.

The items 1..10 (Make..RomSim) are normal batch-lines, whixh can be changed with the edit field. They have a respective SpeedButton. The items Syntax, ErrorFile, HelpFile, Template and AppWizz open an additional dialog. These dialogs are described later.

Structure of a batch line.
A batch line (items 1..10) is able to use other batch lines as reference. The entry **2|Hallo.exe|7** means, that first the item 2 (Compile), then the program *Hallo* and then the item 7 (Assemble) is processed.

Further the **MainFile**, the **ProjectFile**, the **working directory**, as well as the actual **Editor window** can be referenced. Therefor the placeholders **%Mp, %Mb, %Me, %Proj, %Wp, %Wb** and **%We** are determined.

%Mp indicates the working register
%Mb indicates the filename of the MainFile without extension
%Me indicates the extension of the MainFiles (e.g. pas)
%Wp indicates the register of the actual editor window
%Proj indicates the path and filename of the project control files
%Wb indicates the filename of the actual editor window without extension
%We indicates the extension of the actual editor window (e.g. pas)
%Proj indicates the project (File) name, not the MainFile

An example for file reference:
Assuming that the working directory of the project is "C:\Projekte\Tests"and the mainfile is "MyProject.Pas", the following entry is created:

c:\projekte\PICco32\PICco32.exe %Mp\%Mb.%Me

This batch line calls the program "PICco32.exe" and passes it in the commando-line "%Mp\%Mb.%Me" as the following string:

"C:\Projekte\Tests\MyProject.Pas".

For div. lines at the left there is a **CheckBox**, which specifies, if , after the respective program call, the IDE has to wait, until the called program is finished or if it can take over the control. (**Exec and Wait**) For some simulators/emulators a complete reset after every compile procedure is long-winded, so the checkbox is switched off, and e.g. the emulator always stays loaded after the first start.

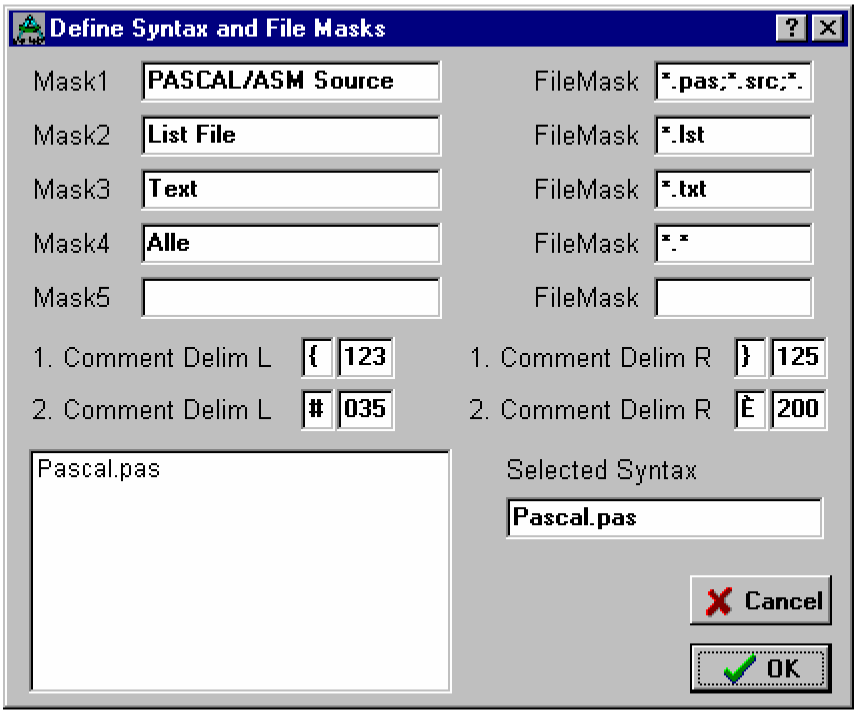
If all items are typed crrectly and completely, the dialog has to be saved with the **Save** button. Here all changes can be made invalid again by using the **Exit** button at the end, not the save button.

Attention:

Do not program a recursion!! A recursion would be, if in item1 the item2 would be called and in item2 item1. Or in item1 item1 itself! The batch processor would execute alternate item1 and item2 without end or until a system crash.

3.12.3 Syntax select

If the item „syntax“ is clicked in the the batch list for **editing** or **creating** a new **control**, the following sub-dialog is displayed. It serves e.g. for selection of the desired high-level language (not compiler), of the file-masks as well as of the comment limits within the source. The several item are linked with the „language“, but they belong to the calling control, e.g. PICpas.



Mask1	PASCAL/ASM Source	FileMask	*.pas;*.src;*.
Mask2	List File	FileMask	*.lst
Mask3	Text	FileMask	*.txt
Mask4	Alle	FileMask	*.*
Mask5		FileMask	

1. Comment Delim L	{	123	1. Comment Delim R	}	125
2. Comment Delim L	#	035	2. Comment Delim R	È	200

Pascal.pas

Selected Syntax: Pascal.pas

Upto 5 **file masks** can be defined, which can be used by the IDE32 for the several file dialogs.

Every entry consists of a comment field (left) and the mask (right).

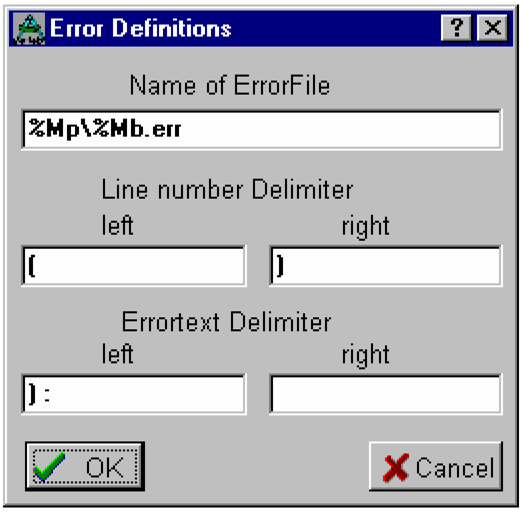
The **Comment Delimiters** (limits) on the left and right are used by the editor for the comment HighLight.

At the left bottom a selection of the high-level languages (not compiler), which are placed at disposal, is displayed. By clicking an item the „language“ is copied in the field **Selected Lanugage**.

3.12.4 Error File define

An essantial point of the software development with a compiler or assembler is the error-evaluation. Without this feature you are put back to the stone age of software. But the essential aspect is that the evaluation after a compile etc. loads the faulty file, positions the caret to the faulty position, and, if possible, it displays an error text or description.

Presupposition is, that tools (compiler etc.) create an error file, which can be evaluated. The IDE must know the structure of this file. Because of the the file and its structure must be specified in the dialog below (sub-dialog of control edit).



Name of ErrorFile: %Mp\%Mb.err

Line number Delimiter: left [right]

Errortext Delimiter: left]: right

The name and the path of the expected file must be typed in the edit field **Name of ErrorFile**. Here the placeholder, which are described above in control edit, are also permitted.

The structure of the file is devided in a line into the declaration of the line-number, limited by two delimiters, and the error-description, also limited by two delimiter.

Because a control only knows one error file and so one structure, the called tools in this control must all use the same error file name and protocols, otherwise errors can not be evaluated.

The E-LAB Tools like for example PICco32 and PICasm32 fulfill this requirement.

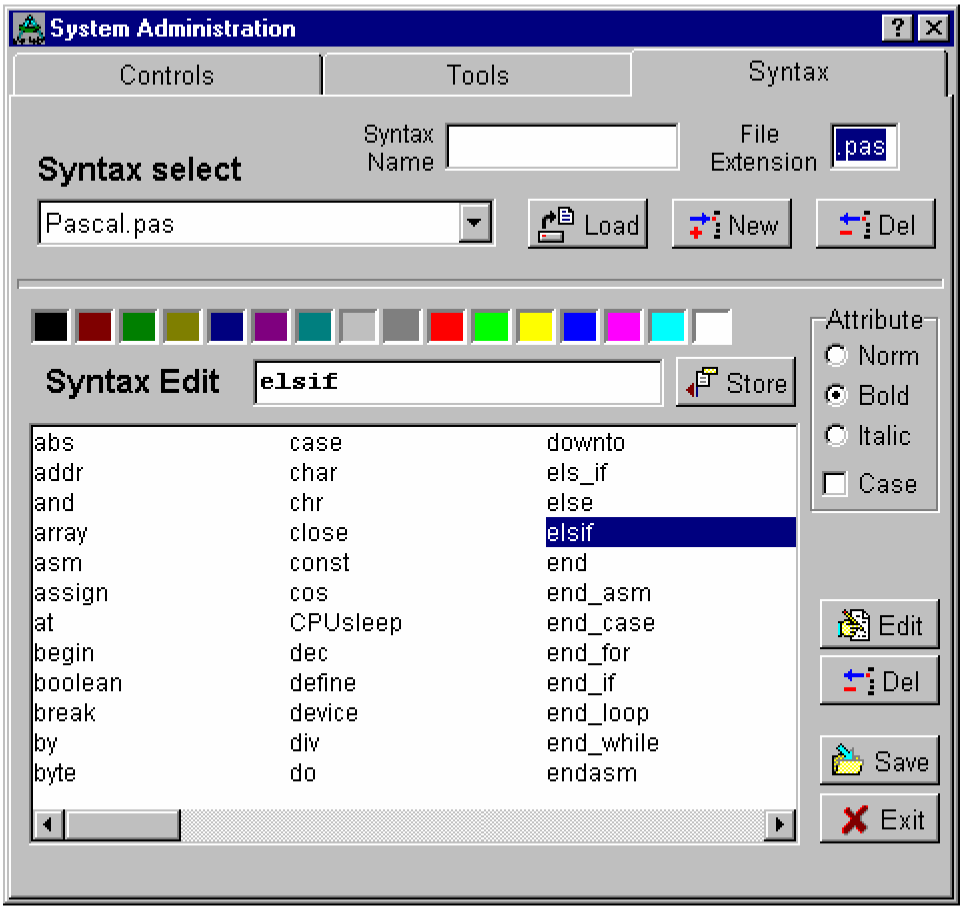
3.13 Syntax

Whose language

The syntax definition (mind you compiler-language) normally is related to a certain control, and so it can be found in a sub-dialog of **System Admin|Control Edit**. Under some circumstances it makes sense to have several Pascal „languages“, for example for several different Pascal compilers. These must be created seperatly and included in the respective control under syntax.

This creation of a respective language-syntax list is not a must. It only serves to emphasize certain syntax elements within the editor.

3.13.1 Syntax edit



Syntax select

Syntax Name

File Extension .pas

Pascal.pas

Syntax Edit elsif

Attribute
○ Norm
● Bold
○ Italic
□ Case

abs
addr
and
array
asm
assign
at
begin
boolean
break
by
byte
case
char
chr
close
const
cos
CPUsleep
dec
define
device
div
do
downto
els_if
else
elsif
end
end_asm
end_case
end_for
end_if
end_loop
end_while
endasm

To change an existing language-syntax or create a new one, the dialog **System Admin** must be opened. This is made by the menu-item **System/System Admin**. After that the following dialog is opened. The dialog page **Syntax** is selected by a click on the respective tab.

If an existing file should be **deleted**, first it must be loaded with the load-button and a click in the selection fiel. The loaded language can be deleted with the delete-button.

The creation of a **new** language syntax is identic with the edit of an existing one, except that first a name must be specified.

This happens with a click on the **new**-button. Now an inputlield is opening above the new-button. Here the name of the new language must be typed. If the name is complete, this must be confirmed with a click on the **load**-button.

The new language syntax now must be chosed out of the list **Syntax Select** and loaded with the **load**-button. The further procedure corresponds to the edit of a language and is following described.

Edit language syntax.
The language, which must be edited, is chosed out of the list **Syntax Select** and loaded with the **load**-button. All already existing syntax words are visible in the list. Every seperate word can be chosed by a doubleclick on the respective entry or by a click on the **edit**-button.

The chosed term then appears in the **edit-window** above for further procedure. Here the term can be changed. A **change** in the text does not create a change in the list, but a **new term!** A change of the text can be reached with **load, delete and new input** of the term. With a click on the **store**-button it is checked, if the new or changed term is already (identic) in the list. If this is the case, the old term stays existing and the text attributes are taken over. If the term is not in the list, the complete text with its attributes is taken over.

So it is secured that new or changed terms create a new term in the list, even if they distinguish in only one letter.

Attribute changes, made by the RadioButtons **Norm, Bold, Italic** as well as by the CheckBox **Case** do not create a new term as well as the change of the font color, always providing there have been no changes in the text itself.

The font attribute norm (normal), Bold and Italic exclude each other. Only one attribute is possible for one term.

The CheckBox **Case** specifies, if the editor should pay attention to capitalization/small letters of this term. If case is active, so the editor uses the defined font attribute and color only if the word found in the source text is abolutely identic (capitalization/small letters) with the entry.

To every term a textcolor can be assigned. This happens by clicking on the respective color level.

All attribute changes are visible in the editfield immediatly. The list field also contain attributes, but it is not possible to make them visible.

If there have been changes, so you must click on the **store**-button to store them. The text changes are visible in the list immediatly.

If the changes made in the edit field should not be stored, do not click the store-button, but the next item of the list must be selected. So the changes are invalid.

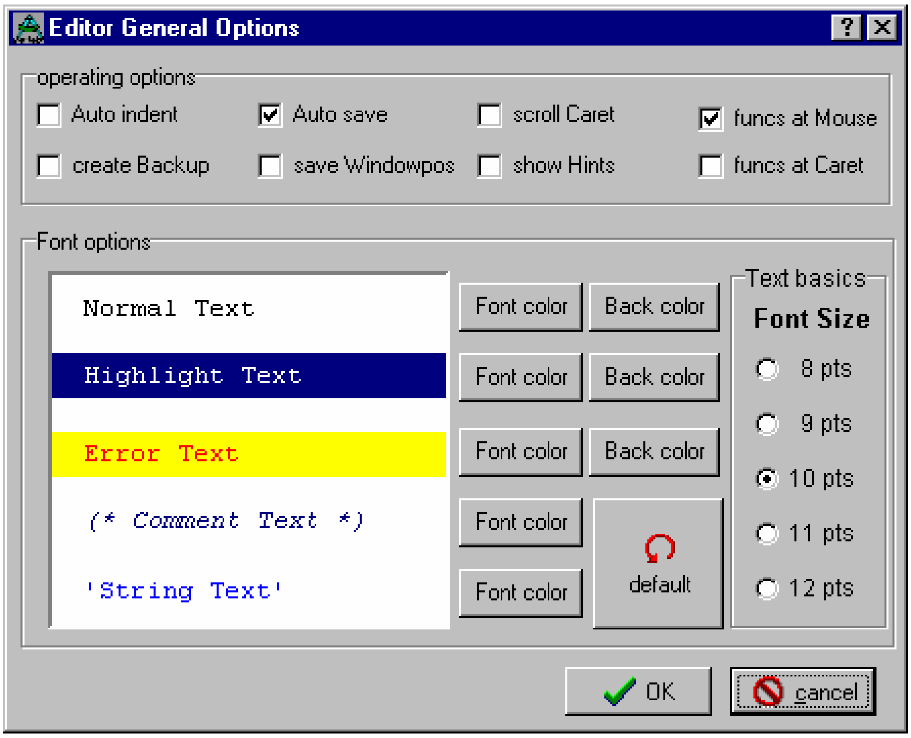
To store all changes (file *NewSyntax*.**lng)** the process must be saved with **save** at the end or exit.

3.14 Editor Setup

3.14.1 Fonts, Colors, Backup and Fasthelp

With this dialog, calling with the menu-item **IDE/General Options**, the font color, the font size and the common behaviour of the editor and IDE can be defined.

Auto indent Automatic indentation is select.
Create Backup before any file is stored, a backup of the existing file is made, that means the origin file is renamed to xxx.BAK
Auto save before call and running a tool (Compiler etc) all open files will be saved
save Windowpos stores the positions and size of all open editors when a project is closed. If this project is re-opened, the positions and sizes will be restored.
scroll Caret with scroll operations in an editor window, the caret is also scrolled/dragged. So it is always in the visible area of the editor window.
show Hints sets all small hintwindows of buttons, editfields etc. visible.
show funcs at Mouse FastHelp windows (Syntax) are displayed at the mouse position
show funcs at Caret FastHelp windows (Syntax) are displayed at the caret position



Editor General Options

operating options

☐ Auto indent ☑ Auto save ☐ scroll Caret ☑ funcs at Mouse
☐ create Backup ☐ save Windowpos ☐ show Hints ☐ funcs at Caret

Font options

Normal Text
Highlight Text
Error Text
(* Comment Text *)
'String Text'

Text basics
Font Size
○ 8 pts
○ 9 pts
● 10 pts
○ 11 pts
○ 12 pts

For all 5 possible text types a related text color can be selected.

For 3 of the text types only the respective background color is selectable.

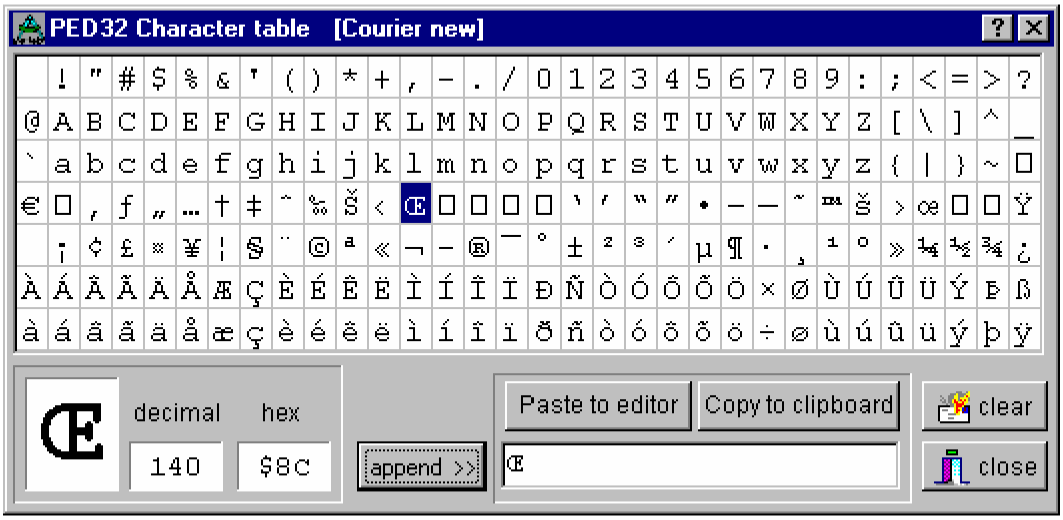
The textsize for all is defined by fontsize.

A click to the button **default** restores all font and color settings to the default value.

The **ok** button stores the selected font and color attributes into PED’s ini file.

3.14.2 Character table

This table serves to look for the ordinal- or hex values of characters. Further it is able to create a text line, which can be copied into the notebook or into the editor.



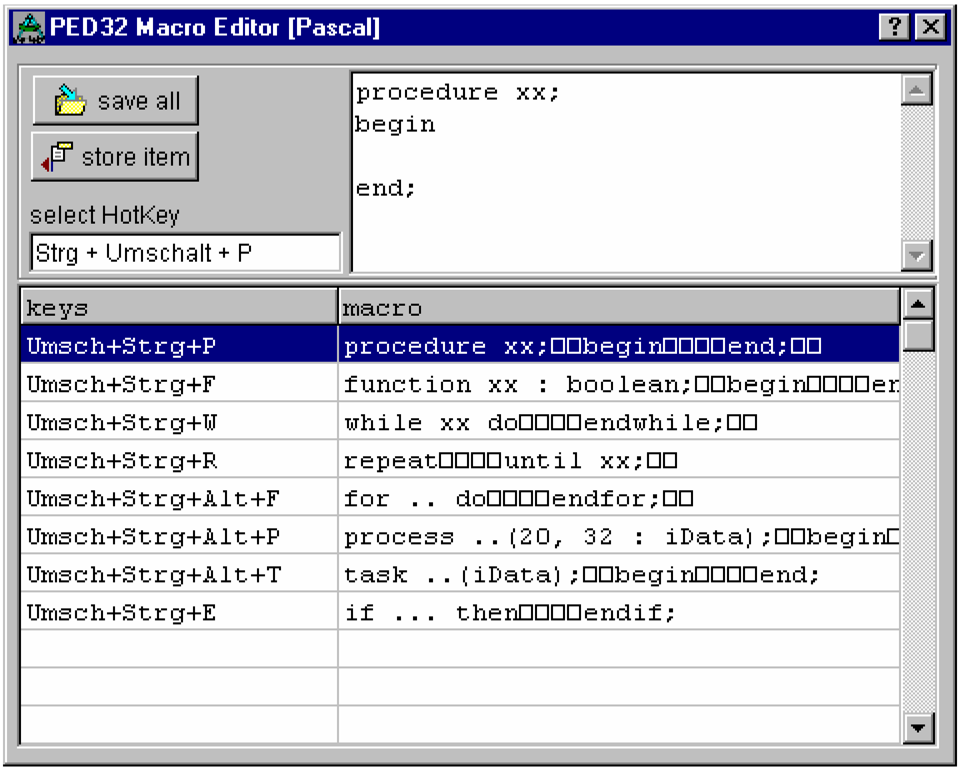
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	□
€	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	□	□	□	‘	’	“	”	•	–	—	˜	™	š	›	œ	□	□	Ÿ
	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Œ

decimal	hex
140	$8C

Œ

3.14.3 Keyboard Macros



Strg + Umschalt + P

```
procedure xx;
begin

end;
```

keys	macro
Umsch+Strg+P	procedure xx;□□begin□□□□end;□□
Umsch+Strg+F	function xx : boolean;□□begin□□□□er
Umsch+Strg+W	while xx do□□□□endwhile;□□
Umsch+Strg+R	repeat□□□□until xx;□□
Umsch+Strg+Alt+F	for .. do□□□□endfor;□□
Umsch+Strg+Alt+P	process ..(20, 32 : iData);□□begin□
Umsch+Strg+Alt+T	task ..(iData);□□begin□□□□end;
Umsch+Strg+E	if ... then□□□□endif;

Keyboard Macros are texts, which can be inserted with socalled Hot-Keys into the actual text.

For this the macros must be created with this dialog. The call is made by **IDE/Edit Keyboard Macros**.

To create a new macro a free field is clicked blue with the mouse button. The macro text now can be typed into the upper input field. With a click on the field **select HotKey** The caret is positioned on this field. With the desired key-combination the hotkey is assigned, e.g. „Ctrl+Shift+P“ for Procedure.

With **Store Item** thr edit field and the HotKey is stored in the table. The complete table is saved with **save all**.

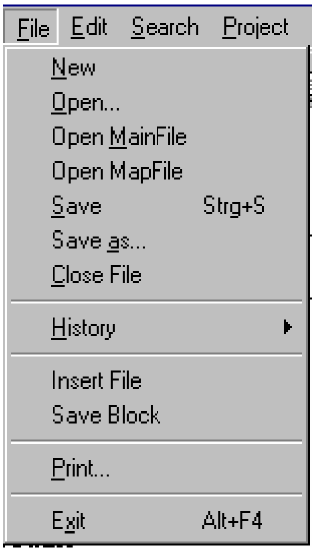
The edit of an existing entry is the same procedure.

Pay attention that the ALT-key does not function in case the main menu has a underlined letter.

For edit now the according text can be inserted at the position of the caret with the respective key function (HotKey).

3.15 Menus

3.15.1 File Menu



New Open a new file.
Open Calls the the file-open dialog.
Open Mainfile Opens the MainFile of the project
Open Mapfile Opens the MapFile of the project
Save Ctrl + S stores the actual editor window in the corr. file
Save As Stores the actual editor window with a new name
Close File Closes the actual editor window with possible file storage

History Displays the last 10 processed projects in order of time. By clicking the chosed project is loaded.
Insert File Read a file at the actual caret position.
Save Block Write the highlighted block as a file.

Print Print the actual file (window) or block.
Exit Alt + F4 Finish PED32.

3.15.1.1 History

The sub-menu **History** displays 10 processed projects in order of time. The last opened project is always at first position. So an overview of the actual projects is possible. With a click on the respective entry it is opened or loaded.

The further procedure is identic with above described **Load Project**.

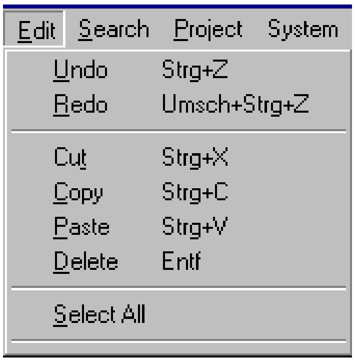
3.15.1.2 Insert File

With the menu-item Insert File a dialog for file-selection is opened. If a file is selected, then it is copied to the caret position in the actual editor window.

3.15.1.3 Save Block

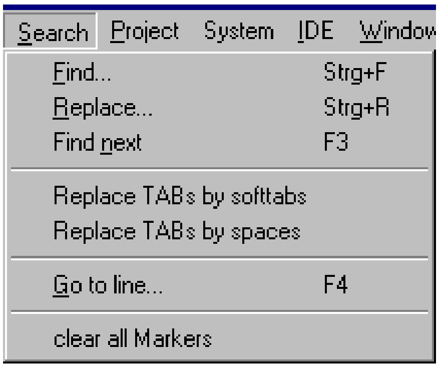
With the menu-item Save Block a dialog for file-name input is opened. If a file name was typed, so the highlighted block in the actual editor window is written into this file. An possible existing content of the file gets lost.

3.15.2 Edit Menu



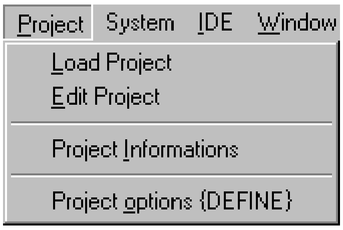
Undo	Undo the last change (Undo)
Redo	Indo the last undo (Redo)
Cut	Cut out thw maked block and copy it into the notebookk
Copy	Cpoy highlighted block into the notebook
Paste	Copy notebook to the caret position
Delete	Delete symbol right of the caret or the whole block
Select All	Mark block from begin upto the end of the file

3.15.3 Search Menu



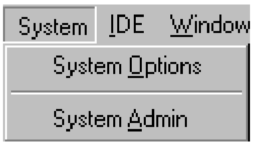
Find	**Ctrl + F**	Search word (at caret)
Replace	**Ctrl + R**	Search word (at caret) and replace it
Find Next	**F3**	Repeat last search/replace
Replace Tabs by SoftTabs		Fill hard tabs with spaces
Replace Tabs by spaces		Replace hard tabs by spaces
Goto Line	**F4**	Set caret to line number n
clear all Markers		Delete all marker-buttons

3.15.4 Project Menu



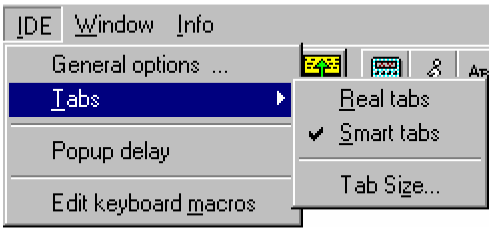
Load Project	Opens the project dialog to load a project
Edit Project	Opens the project dialog to edit a project
Project Information	Opens the info dialog with the files of the project
Project options	Opens the project dialog to input conditional compile switches.

3.15.5 System Menu



System Options	Opens the System Options dialog. Show Compile/Assemble and Project Unit Paths can be selected or edited
System Admin	Opens the system/control dialog

3.15.6 IDE Menu



General Options	Opens the editor option dialog
Tabs	Define of the TABs
PopUp delay	determines the delay of the syntax help
Edit Keyboard Macros	Opens the keyboard macro dialog

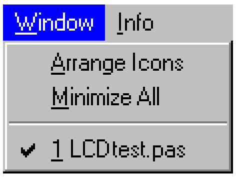
3.15.6.1 Tabs

For tabs it is distinguished between **real Tabs** and smart tabs. For real tabs generally a tab character is inserted with the tab-key in the text. The item Auto Indentation here has no function. These tabs are executed with the in the dialog TabSize specified length by the editor (pseudo spaces).

The **Smart Tabs** are leading to a different behaviour of the editor. For using the enter key at the end of line anew line is inserted. The input caret jumps to the first visible letter of the line above, if **Auto Indentation** is switched on. If this is not the case, the caret stays at the begin of the line. The TAB key has a double function for Auto Indent. If it is pressed, the caret jumps to the next word begin of the line above. If there is no word existing, the number of spaces, specified in TabSize, is filled.

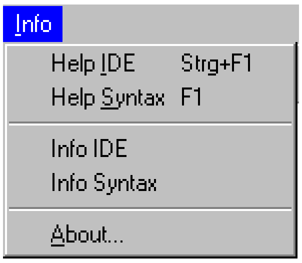
Without **Auto Indent** always the number of "TAB Size" spaces are filled.

3.15.7 Window Menu



Arrange Icons	New order of editor windows decreased to icons
Minimize all	Decrease all editor windows to icon size
Window ...	Do a certain editor windoe in the foreground

3.15.8 Info Menu



Help IDE	Call the IDE or editor help in the context
Help Syntax	Call the compiler or syntax help in the context
Info IDE	Call the overview of the the IDE or editor help
Info Syntax	Call the overview of the compiler or syntax help
About	Display of the actual version of PED32

The IDE PED32 contains two help systems. One refers to the IDE or the editor, the other refers to the respective compiler or assembler.

Help can be switched on by the above menu or by the **F1** key. Help for the compiler can be reached by F1 context-related, the IDE-help is reached by Ctrl + F1. In dialogs the IDE-help can be reached by F1 or the help-button **?**.

3.15.8.1 Help IDE

For every dialog and menu-items there is a help. In case of open dialogs you can get help by the **F1** key or **Ctrl+F1** or by the help-button **?**. Help is corresponding to the databook in the essential items.

3.15.8.2 Help Syntax

If an online-help should be called for a term in the source program (e.g. `repeat`) , so the caret must be placed to the word ***repeat*** in the editor window, then the key **F1** must be pressed. Help is opened at the position of the **repeat** explanation.

3.15.8.3 Info IDE

You can get an overview of the help items of the IDE by the menu "Info IDE" above. Then do the procedure as usual, e.g. search, copy etc.

3.15.8.4 Info Syntax

You can get an overview of the help items of the IDE by the menu "Info syntax" above. Then do the procedure as usual, e.g. search, copy etc.

3.15.8.5 About... and compiler registration

Here the actual version of IDE/PED32 is displayed. Two register fields serve to enable the compiler or the compiler of E-LAB Computers. The procedure is enclosed as info to the respective compiler.

4 Simulator / Debugger

by Gunter Baab

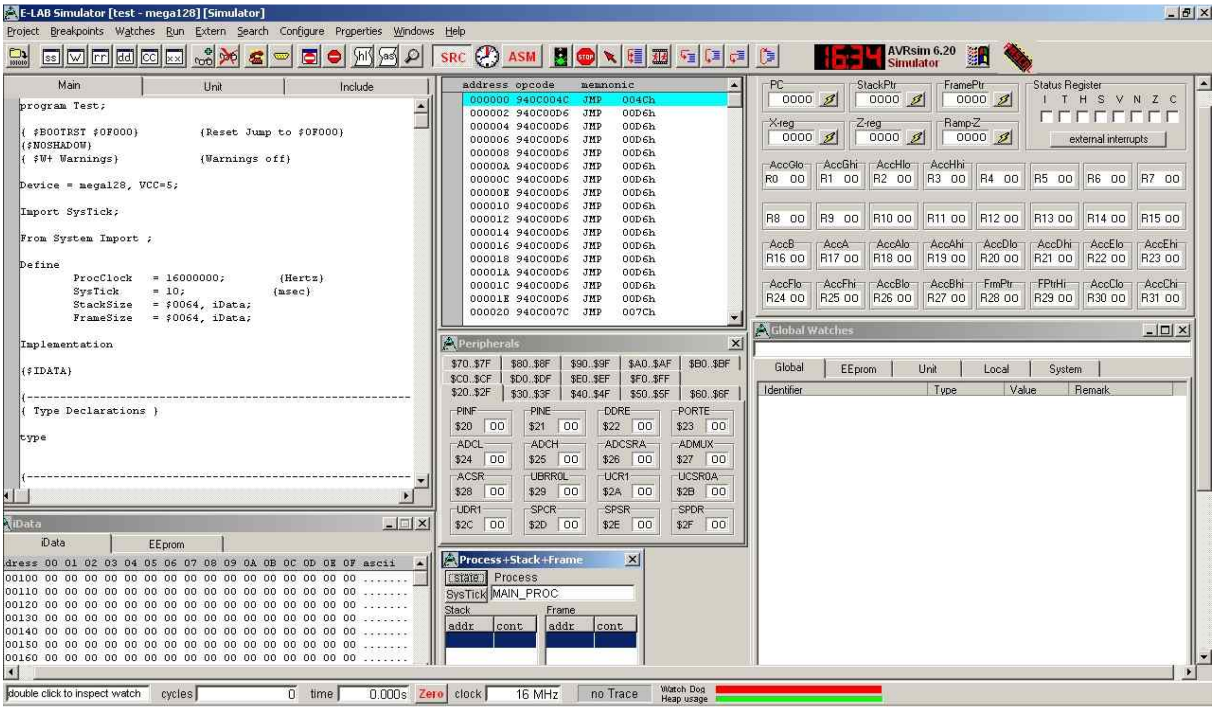
4.1 Introduction

The Simulator an indispensable tool to find logical errors in your programs.
<u>Each program should first be checked with the Simulator.</u>

Start the Simulator from inside the IDE with F9 or use one of the SpeedButtons

4.2 Overview – the Desktop

As usual the desktop consists of several areas:
header, menubar, toolbar, working area and statusbar.
The appearance of the working area varies depending on the selected windows (see the following examples).



```
program Test;

{ $BOOTRST $0F000}          {Reset Jump to $0F000}
{$NOSHADOW}
{ $W+ Warnings}             {Warnings off}

Device = mega128, VCC=5;

Import SysTick;

From System Import ;

Define
        ProcClock   = 16000000;        {Hertz}
        SysTick     = 10;             {msec}
        StackSize   = $0064, iData;
        FrameSize   = $0064, iData;

Implementation

{$IDATA}

{--------------------------------------------------------------
{ Type Declarations }

type


{--------------------------------------------------------------
```

```
address opcode    memnonic
000000 940C004C   JMP    004Ch
000002 940C00D6   JMP    00D6h
000004 940C00D6   JMP    00D6h
000006 940C00D6   JMP    00D6h
000008 940C00D6   JMP    00D6h
00000A 940C00D6   JMP    00D6h
00000C 940C00D6   JMP    00D6h
00000E 940C00D6   JMP    00D6h
000010 940C00D6   JMP    00D6h
000012 940C00D6   JMP    00D6h
000014 940C00D6   JMP    00D6h
000016 940C00D6   JMP    00D6h
000018 940C00D6   JMP    00D6h
00001A 940C00D6   JMP    00D6h
00001C 940C00D6   JMP    00D6h
00001E 940C00D6   JMP    00D6h
000020 940C007C   JMP    007Ch
```

PC 0000 StackPtr 0000 FramePtr 0000
X-reg 0000 Z-reg 0000 Ramp-Z 0000

Status Register: I □ T □ H □ S □ V □ N □ Z □ C □

AccGlo R0 00	AccGhi R1 00	AccHlo R2 00	AccHhi R3 00	R4 00	R5 00	R6 00	R7 00
R8 00	R9 00	R10 00	R11 00	R12 00	R13 00	R14 00	R15 00
AccB R16 00	AccA R17 00	AccAlo R18 00	AccAhi R19 00	AccDlo R20 00	AccDhi R21 00	AccElo R22 00	AccEhi R23 00
AccFlo R24 00	AccFhi R25 00	AccBlo R26 00	AccBhi R27 00	FrmPtr R28 00	FPtrHi R29 00	AccClo R30 00	AccChi R31 00

Peripherals: $70..$7F | $80..$8F | $90..$9F | $A0..$AF | $B0..$BF | $C0..$CF | $D0..$DF | $E0..$EF | $F0..$FF | $20..$2F | $30..$3F | $40..$4F | $50..$5F | $60..$6F

PINF $20 00	PINE $21 00	DDRE $22 00	PORTE $23 00
ADCL $24 00	ADCH $25 00	ADCSRA $26 00	ADMUX $27 00
ACSR $28 00	UBRR0L $29 00	UCR1 $2A 00	UCSR0A $2B 00
UDR1 $2C 00	SPCR $2D 00	SPSR $2E 00	SPDR $2F 00

Identifier	Type	Value	Remark

```
dress 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ascii
00100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
```

state Process: MAIN_PROC
SysTick

Stack

addr	cont

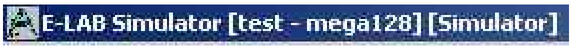
Frame

addr	cont

double click to inspect watch | cycles 0 | time 0.000s | clock 16 MHz | no Trace

4.2.1 the Header

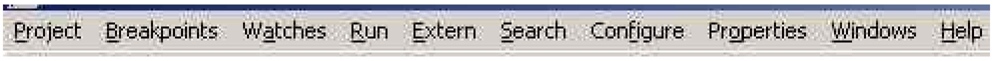
The Header shows the project name and the used controller type:



E-LAB Simulator [test - mega128] [Simulator]

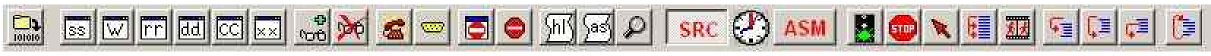
4.2.2 the Menubar

The Menubar contains the usual PopUp menus.
See in the chapter "the Handling of the Simulator" for a detailed description of these menus.



4.2.3 the Toolbar

The toolbar contains "SpeedButtons" for many functions. In the Menu "Windows" this bar can be enabled / disabled.
The SpeedButtons are –besides the Menubar- a further possibility to call the different functions.

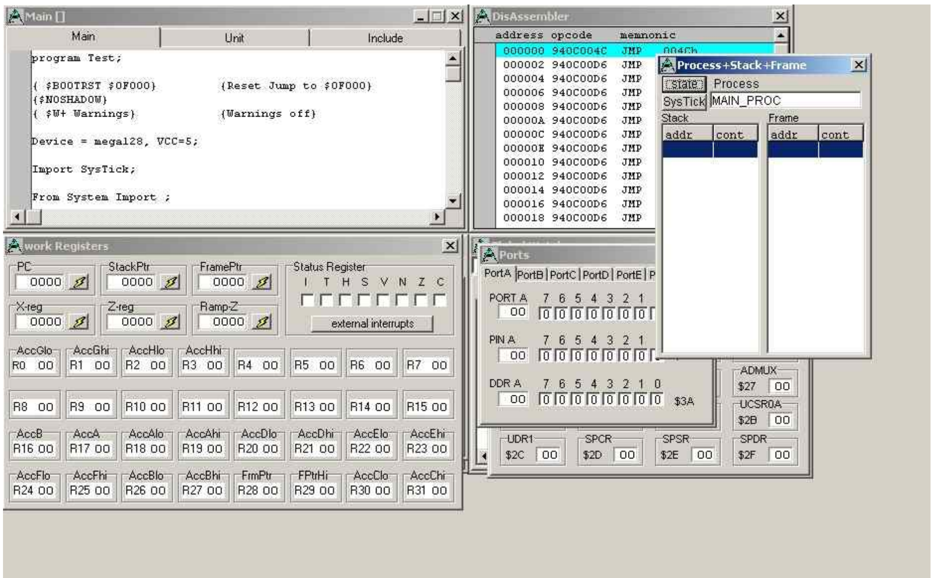


As far as the mouse cursor touches a speed button you can see the meaning of this button in the Statusbar (see chapter "the Statusbar").

4.2.4 the Working Area

As mentioned, the appearance of the working area varies depending on the selected windows.
The state (size/position) of the windows is stored with the project and rebuilt at a new start of the simulator.
The simulator distinguishes mandatory and optional windows.
The mandatory windows are always on the working area and can be minimized.

In the following example all mandatory windows are maximized.



```
program Test;

{ $BOOTRST $0F000}          {Reset Jump to $0F000}
{$NOSHADOW}
{ $W+ Warnings}             {Warnings off}

Device = mega128, VCC=5;

Import SysTick;

From System Import ;
```

```
address opcode    memnonic
000000 940C004C  JMP   004Ch
000002 940C00D6  JMP
000004 940C00D6  JMP
000006 940C00D6  JMP
000008 940C00D6  JMP
00000A 940C00D6  JMP
00000C 940C00D6  JMP
00000E 940C00D6  JMP
000010 940C00D6  JMP
000012 940C00D6  JMP
000014 940C00D6  JMP
000016 940C00D6  JMP
000018 940C00D6  JMP
```

Process: MAIN_PROC

addr	cont

addr	cont

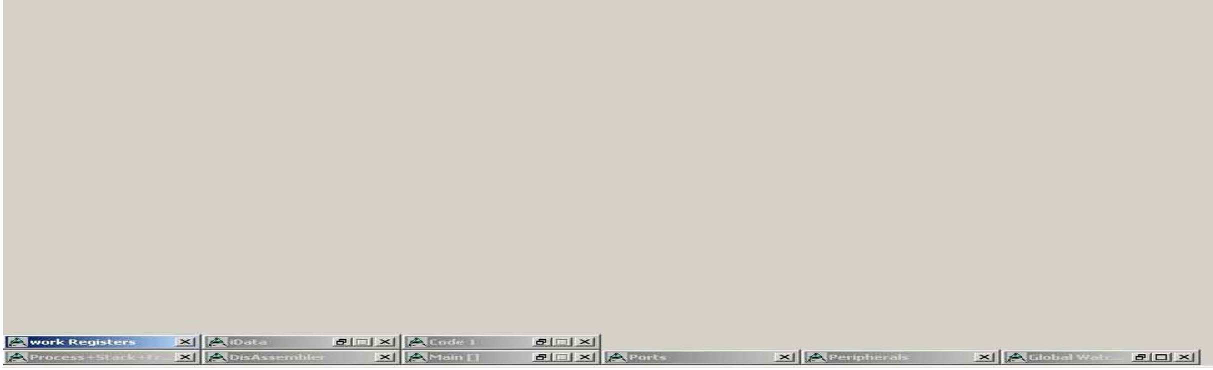
PC 0000 StackPtr 0000 FramePtr 0000
X-reg 0000 Z-reg 0000 Ramp-Z 0000
Status Register I T H S V N Z C □ □ □ □ □ □ □ □

AccGlo	AccGhi	AccHlo	AccHhi				
R0 00	R1 00	R2 00	R3 00	R4 00	R5 00	R6 00	R7 00
R8 00	R9 00	R10 00	R11 00	R12 00	R13 00	R14 00	R15 00
AccB	AccA	AccAlo	AccAhi	AccDlo	AccDhi	AccElo	AccEhi
R16 00	R17 00	R18 00	R19 00	R20 00	R21 00	R22 00	R23 00
AccFlo	AccFhi	AccBlo	AccBhi	FrmPtr	FPtrHi	AccClo	AccChi
R24 00	R25 00	R26 00	R27 00	R28 00	R29 00	R30 00	R31 00

PORT A 00 — 7 6 5 4 3 2 1: 0 0 0 0 0 0 0
PIN A 00 — 7 6 5 4 3 2 1: 0 0 0 0 0 0 0
DDR A 00 — 7 6 5 4 3 2 1 0: 0 0 0 0 0 0 0 0 $3A

ADMUX $27 00
UCSR0A $2B 00
UDR1 $2C 00 SPCR $2D 00 SPSR $2E 00 SPDR $2F 00

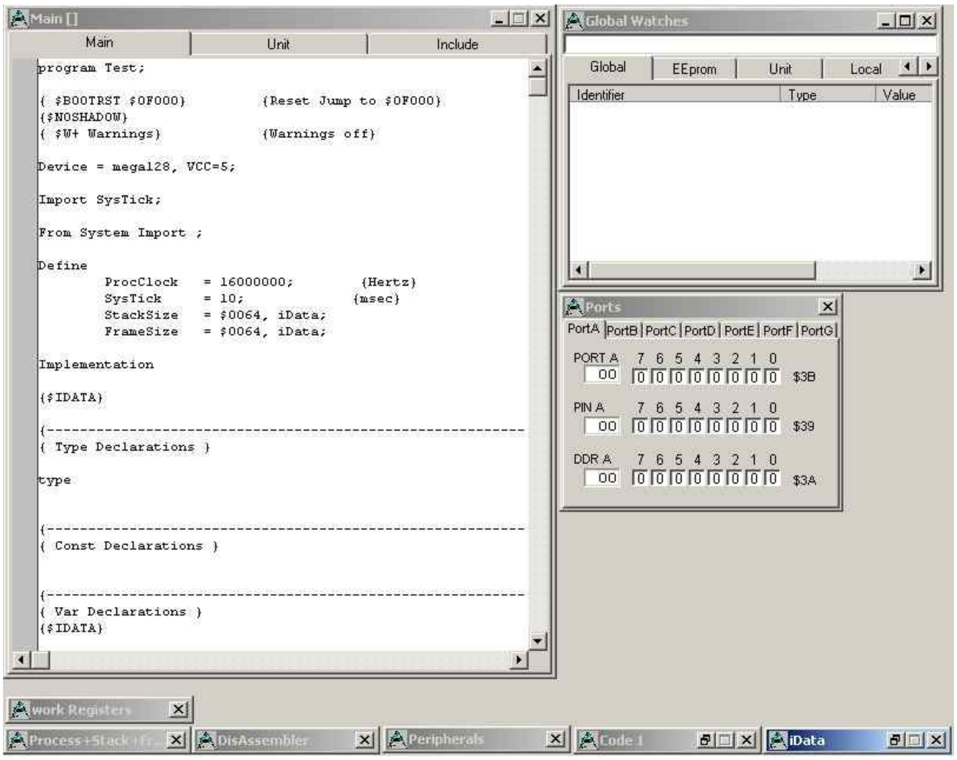
With all mandatory windows minimized and no optional windows open you see the following representation:



To minimize the mandatory windows use the corresponding _ □ ☒ button

To maximize these windows use a double click on the header (the name).

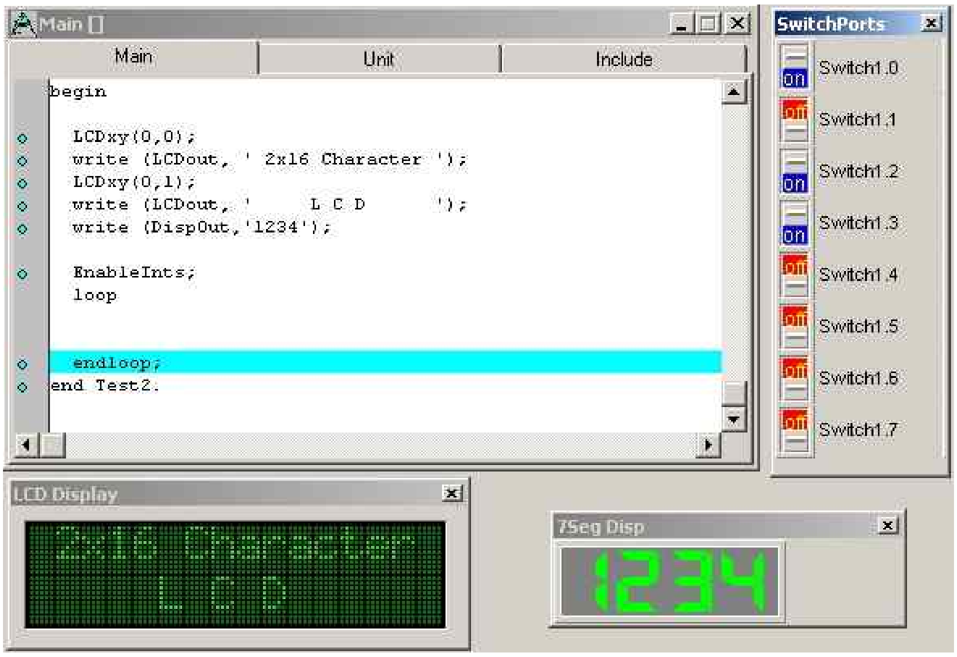
A typical representation could be as follows:



```
program Test;

{ $BOOTRST $0F000}         {Reset Jump to $0F000}
{$NOSHADOW}
{ $W+ Warnings}            {Warnings off}

Device = mega128, VCC=5;

Import SysTick;

From System Import ;

Define
        ProcClock   = 16000000;        {Hertz}
        SysTick     = 10;             {msec}
        StackSize   = $0064, iData;
        FrameSize   = $0064, iData;

Implementation

{$IDATA}

{--------------------------------------------------------------
{ Type Declarations }

type


{--------------------------------------------------------------
{ Const Declarations }


{--------------------------------------------------------------
{ Var Declarations }
{$IDATA}
```

The optional available windows depend on the used hardware and the driver imports (unused options are greyed out).

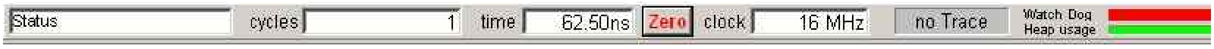
In the following example the SwitchPort-, the LCD- and the 7-Segment driver was imported and the corresponding windows (and the source windows) are maximized.



```
begin

  LCDxy(0,0);
  write (LCDout, ' 2x16 Character ');
  LCDxy(0,1);
  write (LCDout, '     L C D      ');
  write (DispOut,'1234');

  EnableInts;
  loop


  endloop;
end Test2.
```

4.2.5 the Statusbar

The statusbar is on the lower screen edge



"Status" shows hints about the available action at the actual mouse cursor position, e.g. the meaning of the SpeedButton at the mouse cursor.

"cycles" shows the number of the executed processor cycles.
In the example above 1 cycle was executed.

"time" shows the used time (in a real system). Above 62,5 nsec., that corresponds to a clock frequency of 16Mhz.

"zero" resets the "cycles" and "time" to 0 for a new measurement.

"clock" is the frequency defined in the project.

The "Trace" area indicates an active trace (Assembler or Pascal). See the chapter "Menu Run".

At the progress bars for "Watchdog" and "Heap" you see the remaining time to trigger the watchdog or the used / free memory of the heap memory area. These functions are in the above example not used (see chapter “Heap”).

4.3 the Handling of the Simulator

The following chapters describe the menus of the menubar an their options.

4.3.1 Menu Projekt

4.3.1.1 Open / Save / Save as / Print / Printer Setup / Close

These options work like the "File" menu in the Windows / Office standard.

4.3.1.2 Reload / Reload EEprom

To debug the (simulated) memory areas IDATA and EEPROM can be changed.
Sometimes it is helpful to reload the original contents.
You could terminate and restart the simulator. But it is easier to use these functions.
"Reload EEProm" initializes only the EEProm area, "Reload" loads the whole project.
For JTAG Mode: see chapter "JTAG / OWD Debugging – UpLoad/DownLoad".

4.3.2 Menu Breakpoints

You can set breakpoints in the "Source"- or "Disassembler"- window (see chapter "Menu Windows-Source / Disassembler").

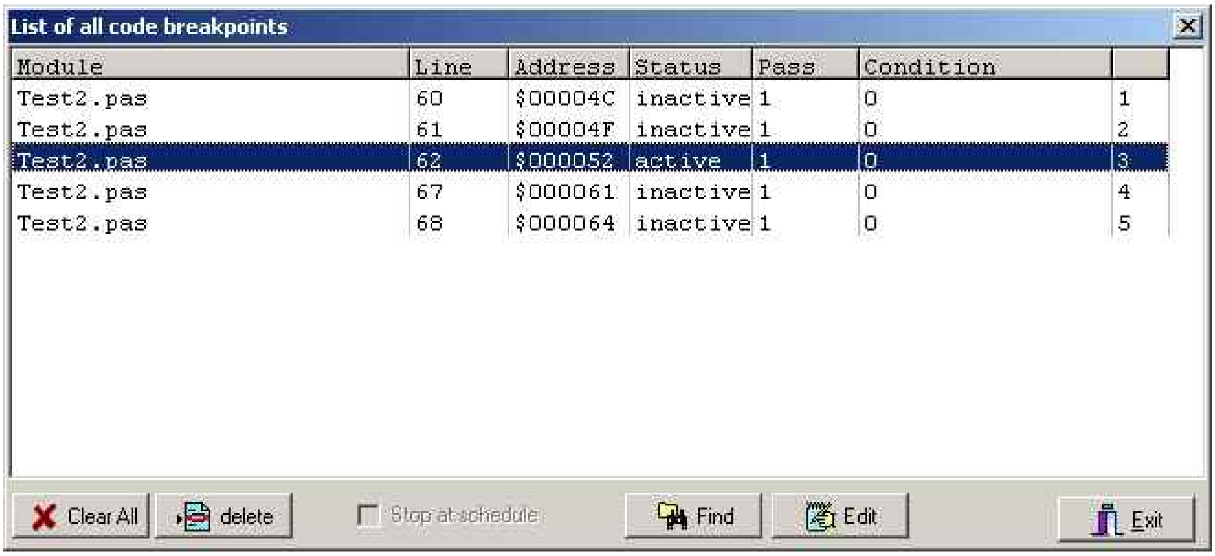
Hints:
the simulator "remembers" the breakpoints using absolute code addresses. These breakpoints are retained at a restart of the simulator. If the source changed the breakpoints may be at other locations.

There are also SpeedButtons for "Toggle Breakpoint" (or function key F5) and "Show List".

In JTAG Mode there is a limit of max. 3 breakpoints: see chapter "JTAG / OWD Debugging – Hardware Breakpoints".

4.3.2.1 Show list

Shows an overview of all active / inactive code-breakpoints:

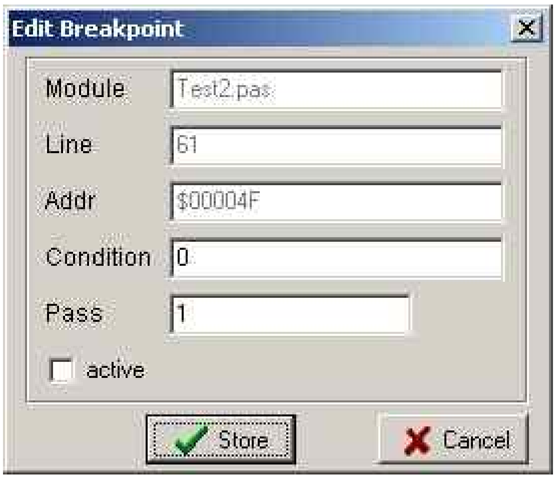


List of all code breakpoints

Module	Line	Address	Status	Pass	Condition	
Test2.pas	60	$00004C	inactive	1	0	1
Test2.pas	61	$00004F	inactive	1	0	2
Test2.pas	62	$000052	active	1	0	3
Test2.pas	67	$000061	inactive	1	0	4
Test2.pas	68	$000064	inactive	1	0	5

"Line" is the source line, "Address" the address in the flash memory.

By the "Edit" button or a double click on the breakpoint you can de- / activate it.



Module: Test2.pas
Line: 61
Addr: $00004F
Condition: 0
Pass: 1
□ active

Deactivated breakpoints are shown in the source window but they do not stop the simulator anymore. "Pass" is a counter. It is decremented every time the program passes that breakpoint. At zero the simulator stops the program. This is not a permanent setting. It has to be set at every "break". The other options are for future enhancements.

4.3.2.2 Reset all Breakpoints

Deletes all breakpoints in the source. Do not confuse this with the state "inactive" where the breakpoints are still in the source but are disabled.
JTAG Hardware Breakpoints werden hier nicht gelöscht ?!?!? ***Sollten aber, muss ich prüfen…***
Da ist noch ein Bug drin, wird noch ausgebaut. (+rh*)

4.3.2.3 Stop after ..

Here is the stop condition the number of performed processor cycles. By selecting this option a popup window to enter the number of cycles comes up.
This option in not available in JTAG mode.

4.3.2.4 Stop on Schedule, Stop on TASK kill

These are on / off options and helpful for multitasking programming.
"Stop on Schedule" stops the simulation as far as the "scheduler" performs a task switch.
"Stop on TASK kill" stops the simulation as far as the "scheduler" has to "kill" a task. On a correctly designed multitasking system this should not happen.
This option in not available in JTAG mode.

4.3.2.5 Memory Write Breakpoints

This option stops the simulator as far as a memory address in the RAM / EEProm (a variable) is changed.

hint:
the simulator "remembers" these breakpoints by the memory address and they are retained at a new start. If the source file changed the breakpoints may be at other locations.

Attention:
these breakpoints are not shown in the "Show List" !!! "Show List" displays only the code breakpoints.

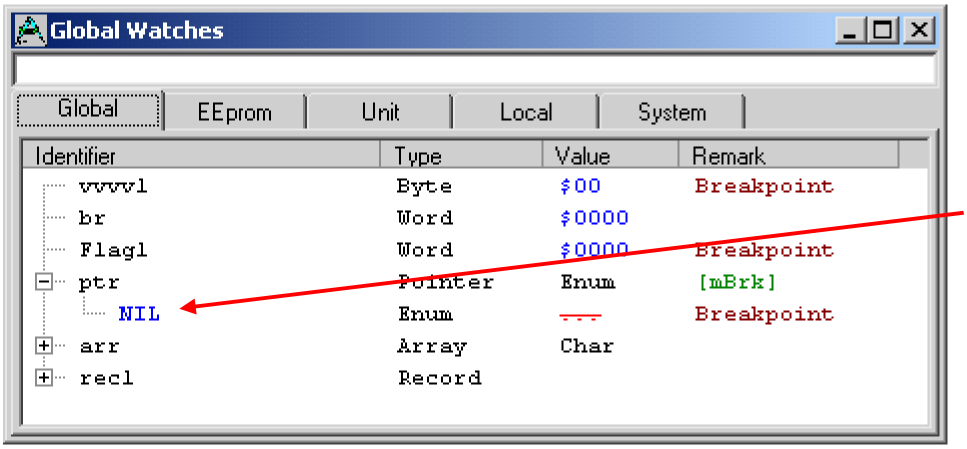
Use a double click to edit a breakpoint. Actually only the address is important.
"Pass" / "Condition" is for future enhancements.

The menu "Watches-Add Watch" (see below) offers a simple possibility to set a breakpoint at the address of a variable.

When dealing around with pointers but also in common it's possible that some variables are overwritten or destroyed without knowing the reason for this. To better find out the circumstances the Simulator is enhanced now so that it is possible to place breakpoints onto variables.

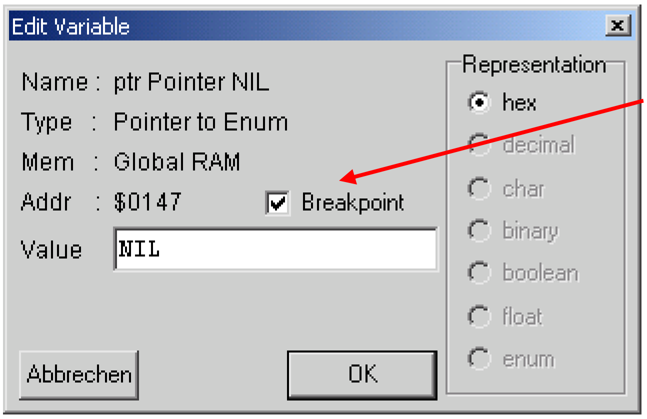
Each time if such a variable is overwritten or changed by the application the program stops and an info message is raised. So it is very easy to determine which part of the program or statement writes or changes the variable.

A memory breakpoint can only placed to such a variable which is already placed into the watch window.



Identifier	Type	Value	Remark
vvvvl	Byte	$00	Breakpoint
br	Word	$0000	
Flagl	Word	$0000	Breakpoint
⊟ ptr	Pointer	Enum	[mBrk]
NIL	Enum	---	Breakpoint
⊞ arr	Array	Char	
⊞ recl	Record		

A double click onto the blue editfield of a watch variable opens the watch edit dialog.

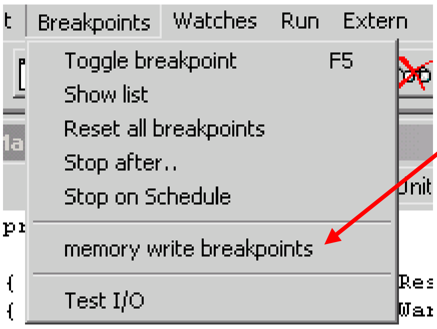


Name : ptr Pointer NIL
Type : Pointer to Enum
Mem : Global RAM
Addr : $0147 ☑ Breakpoint
Value NIL

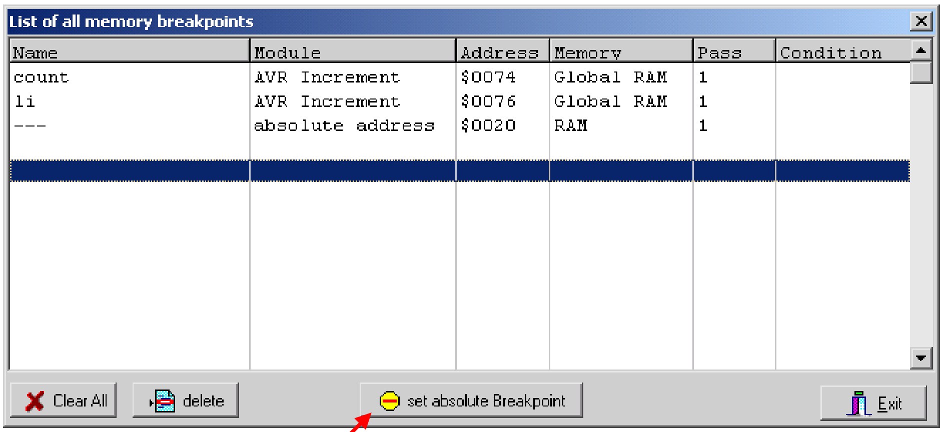
Representation
● hex
○ decimal
○ char
○ binary
○ boolean
○ float
○ enum

Now the memory breakpoint can be switched on or off in the watch edit dialog.

Breakpoints to records, arrays and pipes are not possible. But it's possible to place a overlay variable onto such a variable so these types can be included indirectly.

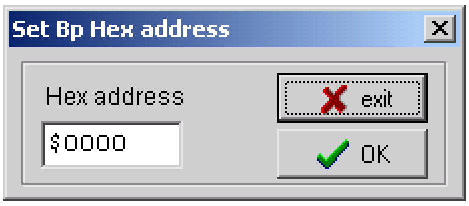


In the Breakpoints menu there is the item *memory write breakpoints* which shows an overview of all memory breakpoints.


List of all memory breakpoints

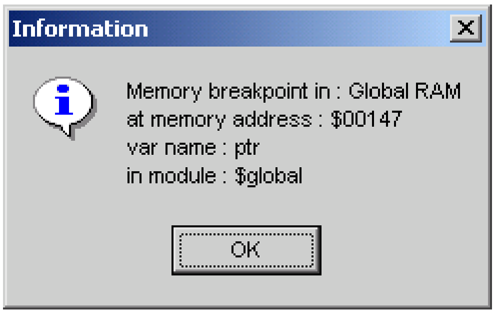
Name	Module	Address	Memory	Pass	Condition
count	AVR Increment	$0074	Global RAM	1	
li	AVR Increment	$0076	Global RAM	1	
---	absolute address	$0020	RAM	1	

It's also possible to set a Memory Breakpoint to an absolute address.


Set Bp Hex address

Hex address $0000

Then this absolute address must be edited here.


Information

Memory breakpoint in : Global RAM
at memory address : $00147
var name : ptr
in module : $global

If in a debug run a write access to such a variable is detected the execution is halted immediately and the info on the left pops up.

4.3.2.6 Test I/O

Not yet implemented. For future enhancements.

4.3.3 Menu Watches

To select the variables to watch (show their content in the simulator) and select their presentation.
In JTAG mode you must note the chapter "JTAG / OWD Debugging – UpLoad/DownLoad" !

4.3.3.1 Add Watch

The simulator lists here all global variables. The variables are listed at the "Globals" tap. At the "System" tab you find all system variables, at the "Unit" tab the (global) variables that are defined in units. Local variables of procedures and functions are *not* available. These are created at runtime when the corresponding procedure or function is running.

The button "Memory Break" easily permits a setting of a "memory write breakpoint".
"Add to WatchList" and "Remove Watch" adds or removes the selected variable to / from the "WatchList".

Hint:
the "WatchList" is a mandatory window and described in the chapter "Windows-Watches".
With the "Edit" button you can change the actual value of a variable, select the presentation
(e.g. decimal / binary / hex) and also define a "memory write breakpoint".

Another possibility to add watches is a double click on a variable in the list. A double click on a variable that is already in the "WatchList" opens the "Edit" menu.

Hint:
You may also click in the "Source" window on a variable to add/delete it to/from the "WatchList" or
to edit it (see chapter "Windows-Source").

4.3.3.2 Delete all Watches

For a fast delete of the "WatchList".

4.3.3.3 Popup Raw Display

This is a "check box selection" for complex variables (structures / arrays). If active, a double click on such a variable in the "WatchList" shows a hex memory dump of it.
In JTAG mode this also updates the variable.

4.3.3.4 default Watch representation

Shows a matrix of all variable types (byte, word, integer, ...) to select the possible default representations (hex, decimal, binary, ...). An individual selection is done in the "Edit" Menu (see chapters "Watches-Add Watch" and " Windows-Watches").

4.3.4 Menu Run

To start the simulation (in the desired manner). In addition there are various settings for the different ways of the simulation and selections to enable traces. In JTAG mode traces are *not* possible.
For the majority of these important functions there are also SpeedButtons.
In single step mode the middle SpeedButtons

select Pascal (SRC – Source Step) or Assembler (ASM – Assembler Step) stepping.

In JTAG mode you must note the chapter "JTAG / OWD Debugging" !

4.3.4.1 Reset processor Ctrl+F2

Sets the program counter to 0 to restart the simulation from the beginning. Corresponds to a hardware reset on the real system. The processor registers are loaded with the specified initial values.

4.3.4.2 Go F9

Starts the simulation with maximum speed.

4.3.4.3 Goto cursor pos F4

Starts the simulation with maximum speed until the cursor position in the source window or a breakpoint is reached.
warum ist die Option "greyed out" und nur der entsprechende SpeedButton geht?
→ klärt rh noch. ***Wird noch freigegeben.*** Kommentar bitte stehen lassen……….(+rh*)

4.3.4.4 Stop simulation F2

Stops the simulation.

4.3.4.5 Step into F7

Performs a single step. Functions, procedures and loops are also executed in single steps.

4.3.4.6 Step over F8

Performs a single step. Functions, procedures and loops are *not* executed in single step mode
(i.e. are executed in a single step).

4.3.4.7 Step out F6

Executes the current function / procedure / loop with maximum speed and stops (as far as there is no previous break condition).

4.3.4.8 Multiple Steps Shift+F9

Executes a predefined number of steps with maximum speed.
See below: "Multiple step value".

4.3.4.9 Animate Ctrl+F9

Starts the simulation with reduced speed.
See below: "Animation speed".

4.3.4.10 Multiple step value

Set the number of steps for "Multiple Steps" (see above).

4.3.4.11 Animation speed

Set the speed for "Animate" (see above).

4.3.4.12 Enable Trace ASM / Enable Trace HLL

To track a trace either on Assembler or on Pascal level.
Hints:
to view the trace see chapter "Windows-view Trace".
Assembler traces show in addition the Pascal source statements. During a trace session you can change from ASM to HLL or "disable" (and resume) the trace tracking.
In JTAG mode these functions are not available.

4.3.4.13 Clear Trace buffer

Clears the trace buffer that is otherwise it is constantly updated.
In JTAG mode this functions is not available.

4.3.4.14 Call Stack Ctrl+F3

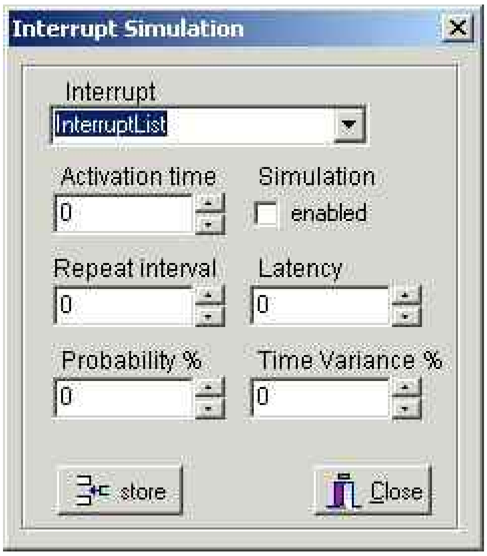
Shows the actual nesting of the return stack (the return procedure- / function names).
In JTAG mode this functions is not available.

4.3.5 Menu Extern

You find here all processor external devices that can be simulated (in opposite to e.g. the ports). Except "Interrupts" all items are also in the menu "Windows". See chapter "Menu Windows" for the descriptions.

4.3.5.1 Interrupts

To simulate the external interrupts (at the processor pins INT0, INT1, ...).


Interrupt Simulation

Interrupt: InterruptList

Activation time: 0

Simulation: □ enabled

Repeat interval: 0

Latency: 0

Probability %: 0

Time Variance %: 0

This function is actually *not* available.

4.3.6 Menu Search

Is used together with the windows "Source", "iData" (the RAM memory), "EEProm" or "Code" (the Flash memory) and maximizes these windows if necessary. To search for specific locations (addresses), hex patterns or text.

4.3.6.1 Show Code at..

Show the flash content at a specified address. Maximizes the "Code" window if necessary.

4.3.6.2 Show Data at..

Show the RAM or EEProm content at a specified address. Maximizes the "iData / EEProm" window if necessary.

4.3.6.3 Search Code hex pattern..

Searches a hex pattern in the code area. Maximizes the "Code" window if necessary.

4.3.6.4 Search Data hex pattern..

Searches a hex pattern in the data area. Maximizes the "iData / EEProm" window if necessary.

4.3.6.5 Search in Source F3

Performs a full-text search in the source file. The usual options (forward, backward, whole word etc.) are available. Maximizes the "Source" window if necessary.
warten auf Antwort von rh – derzeit keine Maximierung von Source ***Kommt noch***
und bleibt deshalb als Hinweis stehen. Das Source Window wird zukünftig der gleiche Editor wie beim PED sein, nur read-only. Aber Syntax-highlight, Kommentar Beachtung etc. (+rh*)

4.3.7 Menu Configure

To make some settings concerning the desktop and the simulation.

4.3.7.1 Show Hints

Enables / disabled the popup help at the mouse cursor. You can also see these hints in the statusbar on the lower left corner of the screen.

4.3.7.2 Save as default

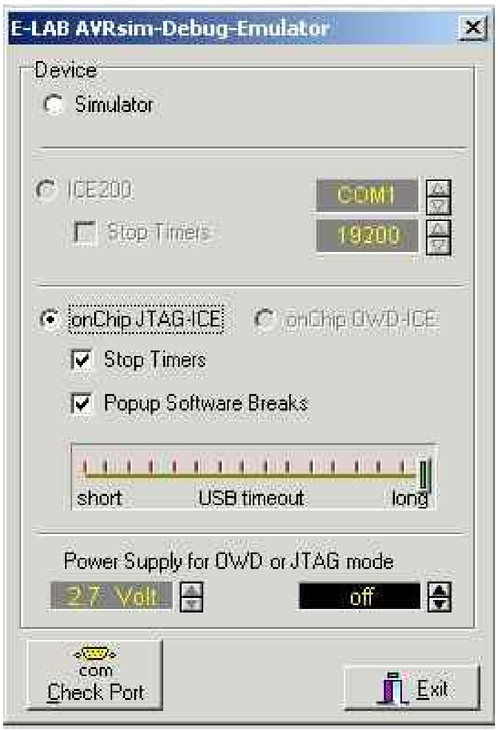
Concerns the positions and alignments (maximized / minimized) of the windows.

4.3.7.3 Config with default

Restores the positions and alignments (maximized / minimized) of the windows.

4.3.7.4 COMport [ICE..Monitor]

To configure the debug hardware if any connected.



E-LAB AVRsim-Debug-Emulator

Device

○ Simulator

○ ICE200 COM1

□ Stop Timers 19200

● onChip JTAG-ICE ○ onChip OWD-ICE

☑ Stop Timers

☑ Popup Software Breaks

short USB timeout long

Power Supply for OWD or JTAG mode

2.7 Volt off

"Simulator": no hardware connected that supports debugging

"ICE200": a Atmel ICE200 Emulator is connected. "Stop Timers" stops the timer of the controller as far the simulation is stopped – e.g. at a breakpoint.
The ICE200 communicates via a serial interface with the PC. You also can configure this interface.

"onChip JTAG-ICE", "onChip OWD-ICE": the internal debugging machine of the controller is used.
JTAG (already implemented) is the standardized JTAG debug Interface (JTAG = "Joint Test Action Group").
"OWD" is reserved for the future "One Wire Debug" Interface. "Stop Timers" stops the timer of the controller as far the simulation is stopped – e.g. at a breakpoint.
"Popup Software Breaks" shows an info window at a software breakpoint.
"Check Port" is with hardware debugging mandatory. It also initializes the debugger.

The JTAG programmers communicate via a USB interface with the PC. You can use the "USB Timeout" to optimize the speed (be careful!).
"Power Supply for OWD or JTAG mode" selects the programmer as current source for the target.
The button "Check Port" searches all available interfaces for a programmer.
As the programmer is also initialized by that button, it is a mandatory selection!

4.3.8 Menu Properties

As the simulator can not reach the speed of a real system, some functions are working with inconvenient effects. The following "check box switches" help to improve this behaviour.

4.3.8.1 Short mDelay

"mDelays" are (in the real system) delays in the range of milli-seconds.
In the simulator they can take quite a lot of time. So this switch shortens these delays.

4.3.8.2 Fast RTC

Is used to increase the speed of the realtime clock for a more realistic simulation of the RTC driver.

4.3.8.3 Short Beep

Adjusts the duration of the sound driver output to the lower speed of the simulator to archieve a more realistic simulation.

4.3.9 Menu Windows

Is used to open optional windows and to get informations about the multi tasking system.
If the corresponding driver is not loaded (and the function is not available) the option is greyed out.

4.3.9.1 Toobar

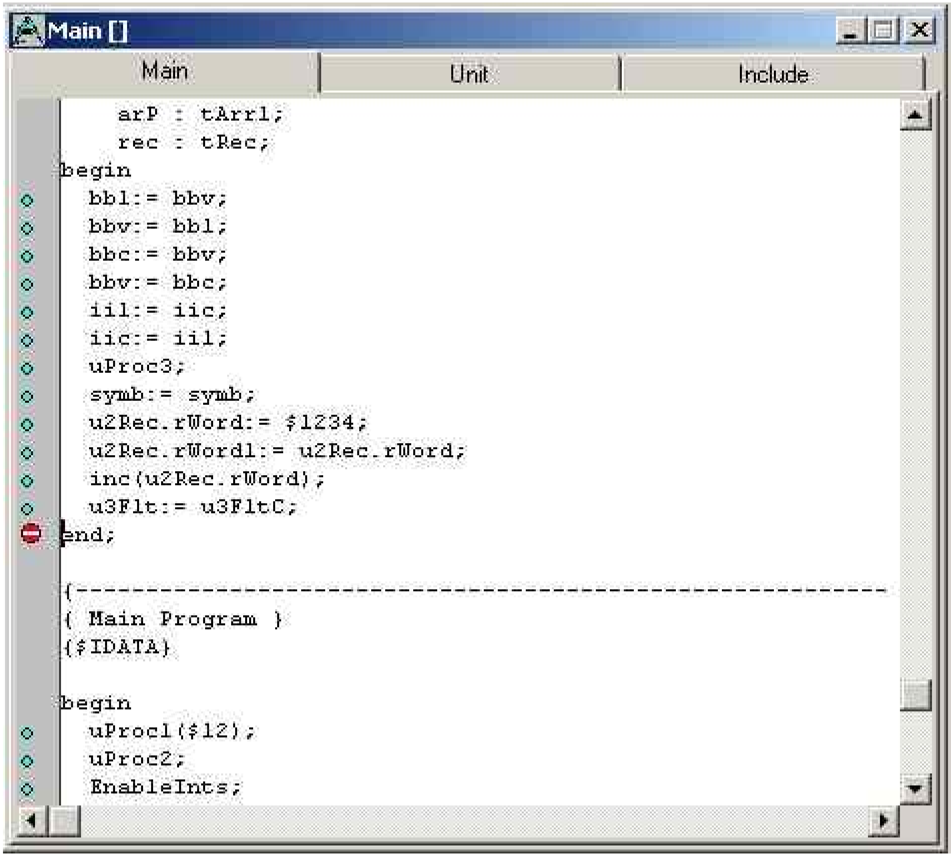
Enables / disables the toolbar with the SpeedButtons.

4.3.9.2 Arrange icons

Arranges the headers of the minimized mandatory windows at the lower screen edge.
Optional windows are completely closed (not minimized to headers or icons).

4.3.9.3 Source

Is a mandatory window.



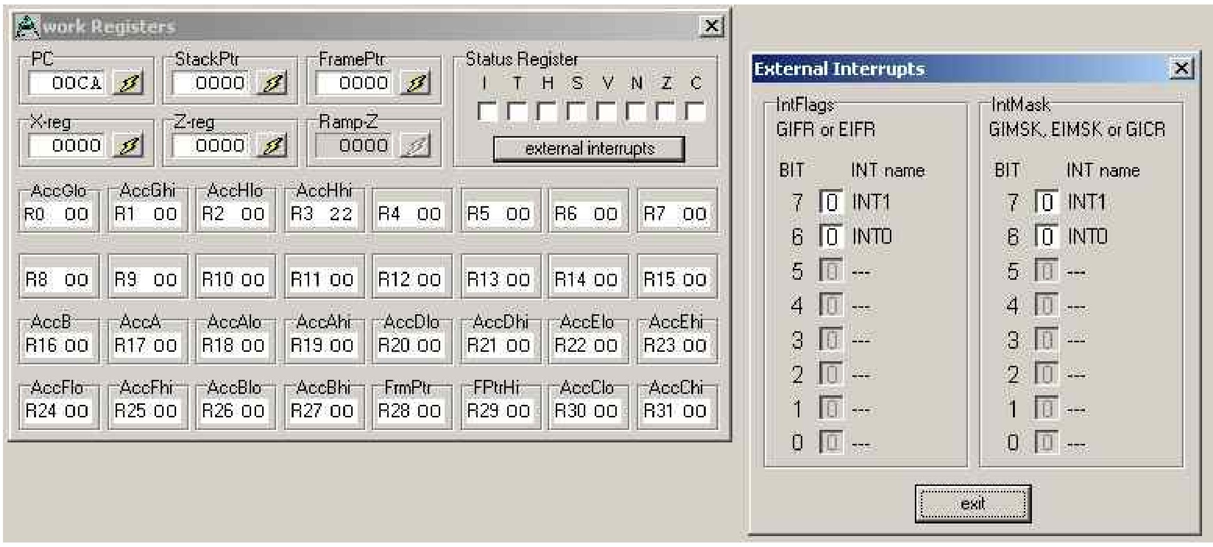
```
    arP : tArr1;
    rec : tRec;
begin
  bb1:= bbv;
  bbv:= bb1;
  bbc:= bbv;
  bbv:= bbc;
  iil:= iic;
  iic:= iil;
  uProc3;
  symb:= symb;
  u2Rec.rWord:= $1234;
  u2Rec.rWord1:= u2Rec.rWord;
  inc(u2Rec.rWord);
  u3Flt:= u3FltC;
end;

{--------------------------------------------------------
{ Main Program }
{$IDATA}

begin
  uProc1($12);
  uProc2;
  EnableInts;
```

"Main" contains your Pascal source. The windows "Unit" and "Include" are at first empty.
By a double click in one of these windows you select a unit or include file to display.
As a prerequisite you must have the unit as source file (not only as a precompiled .PCU).
If your program branches to a unit or an include this file is automatically loaded in the corresponding window.

Furthermore a right click offers the option "Find Text" to search.
If the text cursor points to an executable statement the options "Set PC to line pos" (so the next single step will execute this statement) and "Goto line pos" (run the program to the actual cursor position) are available.
A click on the small blue circles (in the left most column) can be used to set / reset a breakpoint.
A double click on a variable adds this variable to the "Watch Window".

4.3.9.4 Work Registers

Is a mandatory window.



PC 00CA
StackPtr 0000
FramePtr 0000
X-reg 0000
Z-reg 0000
Ramp-Z 0000

Status Register

I	T	H	S	V	N	Z	C
☐	☐	☐	☐	☐	☐	☐	☐

AccGlo	AccGhi	AccHlo	AccHhi				
R0 00	R1 00	R2 00	R3 22	R4 00	R5 00	R6 00	R7 00
R8 00	R9 00	R10 00	R11 00	R12 00	R13 00	R14 00	R15 00
AccB	AccA	AccAlo	AccAhi	AccDlo	AccDhi	AccElo	AccEhi
R16 00	R17 00	R18 00	R19 00	R20 00	R21 00	R22 00	R23 00
AccFlo	AccFhi	AccBlo	AccBhi	FrmPtr	FPtrHi	AccClo	AccChi
R24 00	R25 00	R26 00	R27 00	R28 00	R29 00	R30 00	R31 00

IntFlags
GIFR or EIFR

BIT		INT name
7	0	INT1
6	0	INT0
5	0	---
4	0	---
3	0	---
2	0	---
1	0	---
0	0	---

IntMask
GIMSK, EIMSK or GICR

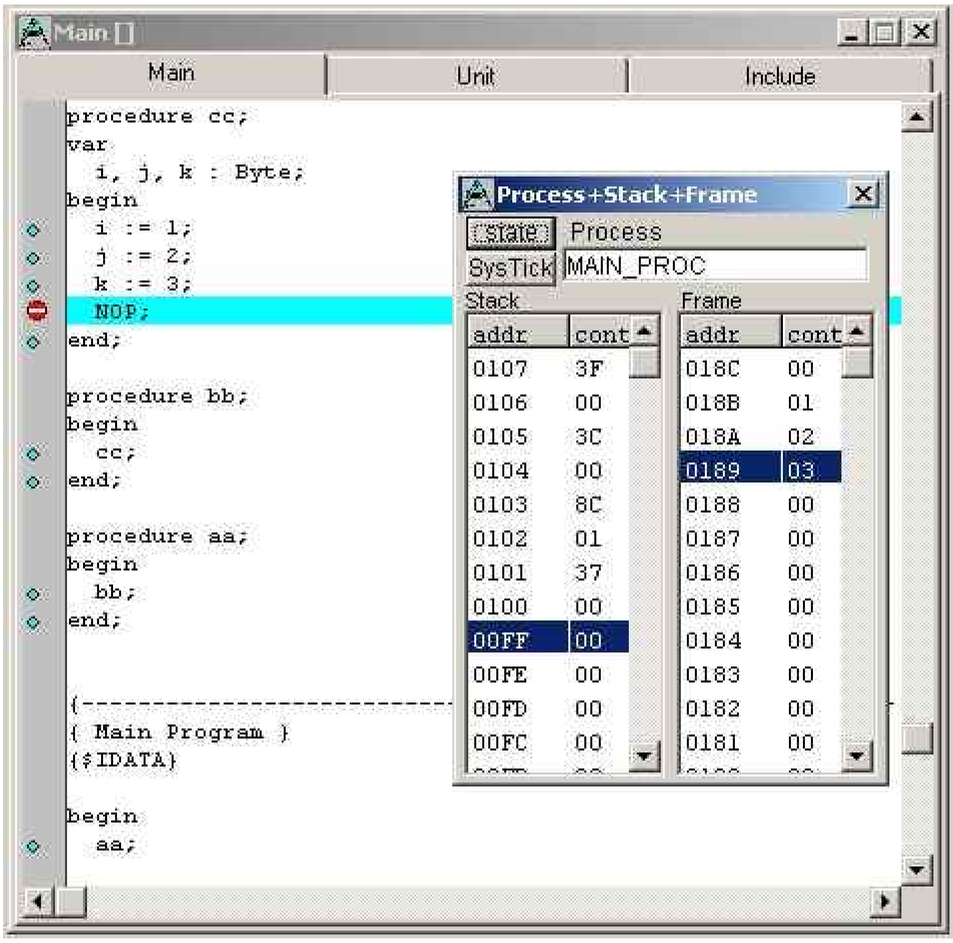
BIT		INT name
7	0	INT1
6	0	INT0
5	0	---
4	0	---
3	0	---
2	0	---
1	0	---
0	0	---

Useful for debugging on Assembler level. It shows the internal processor registers, the symbolic names that uses the AVRco and the external interrupts.
Note that you can here also change the contents of the registers.

In JTAG mode see the chapter "JTAG / OWD Debugging – UpLoad / DownLoad".

4.3.9.5 Processes

Is a mandatory window.



```
procedure cc;
var
  i, j, k : Byte;
begin
  i := 1;
  j := 2;
  k := 3;
  NOP;
end;

procedure bb;
begin
  cc;
end;

procedure aa;
begin
  bb;
end;


{---------------------------
{ Main Program }
{$IDATA}

begin
  aa;
```

Process: MAIN_PROC

Stack

addr	cont
0107	3F
0106	00
0105	3C
0104	00
0103	8C
0102	01
0101	37
0100	00
00FF	00
00FE	00
00FD	00
00FC	00

Frame

addr	cont
018C	00
018B	01
018A	02
0189	03
0188	00
0187	00
0186	00
0185	00
0184	00
0183	00
0182	00
0181	00

In the upper example there are 3 nested procedures. The lowest one has 3 local byte variables. You see the stack and frame content when the breakpoint was reached. The positions of the stack / framepointer are blue highlighted.
The buttons "state" and "SysTick" are of interest for multitasking and explained in the chapters "Windows – view Process states" and "Windows – SysTick / Scheduler Timings".

4.3.9.6 Disassembler

Is a mandatory window.

Disassembler shows the Assembler code. The actual position of the program counter is highlighted in bright blue, the position of the text cursor in dark blue. By a click on a line in the source (a line that generates code) the disassembler window shows the corresponding code area. In the same way a click in the Assembler code shows the correspondent part in the source window.
Use a right click to set the program counter to the address of the statement at the mouse cursor.

4.3.9.7 Code memory

Is a mandatory window.
Shows a hexdump of the Assembler code. With right click you can select different presentations and use a search function.

4.3.9.8 Data memory

Is a mandatory window.
Shows a hexdump of the RAM and the EEProm memory. You also can change these areas.
A right click makes a search function available. In JTAG /OWD mode there are additional functions to Upload / Download these memory areas..

4.3.9.9 Watches

Is a mandatory window.
Shows the actual value of variables. In JTAG mode see the chapter "JTAG / OWD Debugging" !
By a double click you can change the value of a variable and select the presentation mode (hex, decimal, …).
For a better overview the variables are grouped under the tabs "Global", "EEprom", "Unit", "Local" and "System".
Under the local tab you can -by a right click- select to display (to upload) the local variables in JTAG mode at a stop inside a procedure or function.
"check Frame and Stack" uploads in JTAG mode when the window is updated also the stack- and frame pointers and checks these for an overflow condition.

A right click in the other windows opens a menu to add or remove watches and to set memory breakpoints. The option "Refresh all Watches" is very important in JTAG / OWD mode.

4.3.9.10 Ports

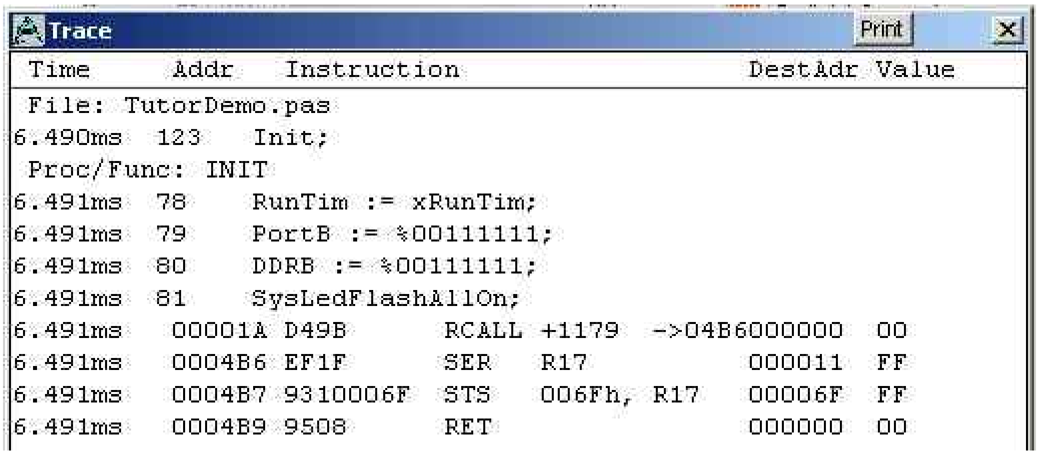
Is a mandatory window.
Shows the available port registers (PortX, PinX, DDRx) of the used controller with their symbolic names. You also can modify the contents for debugging purposes.

4.3.9.11 Peripherals

Is a mandatory window.
Shows all available registers of your controller sorted by their addresses. Except the read-only registers you can also change their contents. This window is useful for debugging of Assembler routines.

4.3.9.12 view Trace

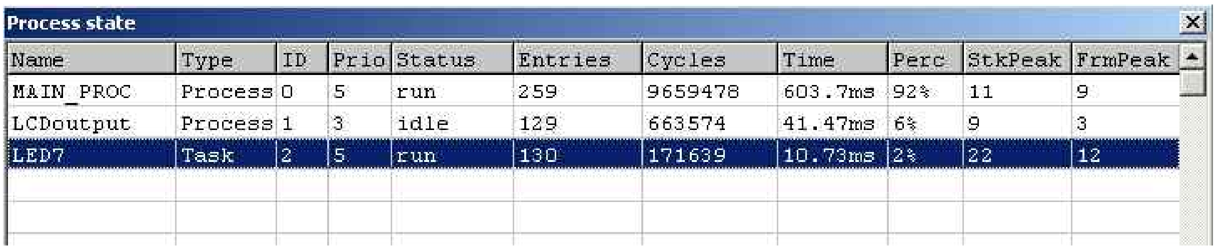
Shows the traces (SRC / HLL) that can be enabled in the menu "Run".
In the following example the lines 78, 79, 80 were executed with HLL trace enabled. The line 81 was executed with ASM trace enabled.
In JTAG mode traces are not possible.



```
 Time      Addr   Instruction                      DestAdr Value
 File: TutorDemo.pas
6.490ms  123    Init;
 Proc/Func: INIT
6.491ms  78     RunTim := xRunTim;
6.491ms  79     PortB := %00111111;
6.491ms  80     DDRB := %00111111;
6.491ms  81     SysLedFlashAllOn;
6.491ms   00001A D49B      RCALL +1179  ->04B6000000  00
6.491ms   0004B6 EF1F      SER   R17           000011  FF
6.491ms   0004B7 9310006F  STS   006Fh, R17    00006F  FF
6.491ms   0004B9 9508      RET                 000000  00
```

4.3.9.13 view Process states

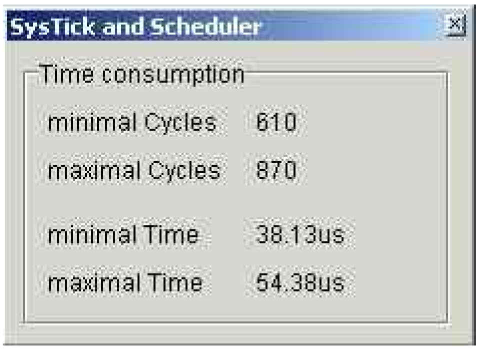
This window is used to get informations about the MultiTasking system.
You can check the priorities of the tasks / processes and the max. used stack and frame memory ("StkPeak" / "FrmPeak"). You should simulate the application for a longer time and try to simulate all possible events (I/Os, external interrupts etc.).
This function is not available in JTAG mode.



Name	Type	ID	Prio	Status	Entries	Cycles	Time	Perc	StkPeak	FrmPeak
MAIN_PROC	Process	0	5	run	259	9659478	603.7ms	92%	11	9
LCDoutput	Process	1	3	idle	129	663574	41.47ms	6%	9	3
LED7	Task	2	5	run	130	171639	10.73ms	2%	22	12

4.3.9.14 SysTick / Scheduler timings

Is used to check the MultiTasking system by displaying the used resources of the SysTick and the Scheduler.
This function is not available in JTAG mode.



SysTick and Scheduler

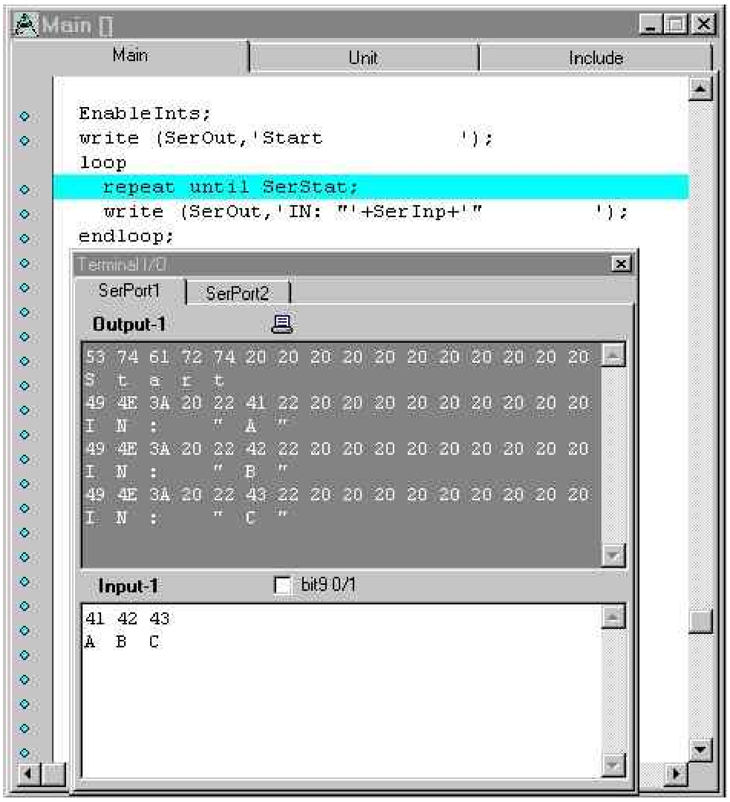
Time consumption

minimal Cycles	610
maximal Cycles	870
minimal Time	38.13us
maximal Time	54.38us

4.3.9.15 Terminal I/O

Import: SerPort / SerPort2

Is an optional window to debug the communication via the serial interface(s).
The actual controllers come with USARTS that support 9 databits (e.g. for multi processor communication). You select the value of the 9th bit in the input area. To enter hex values use the Ctrl + 0...9, A...F keys.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).



```
EnableInts;
write (SerOut,'Start            ');
loop
  repeat until SerStat;
  write (SerOut,'IN: "'+SerInp+'"         ');
endloop;
```

```
53 74 61 72 74 20 20 20 20 20 20 20 20 20 20 20
S  t  a  r  t
49 4E 3A 20 22 41 22 20 20 20 20 20 20 20 20 20
I  N  :     "  A  "
49 4E 3A 20 22 42 22 20 20 20 20 20 20 20 20 20
I  N  :     "  B  "
49 4E 3A 20 22 43 22 20 20 20 20 20 20 20 20 20
I  N  :     "  C  "
```

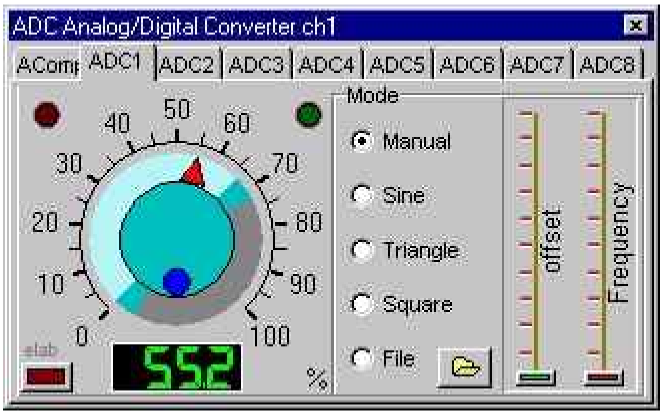
```
41 42 43
A  B  C
```

In the upper (grey) window are the outputs of "SerOut". In the lower window the user inputs that are read via "SerInp" and that come in the real system from the serial interface.

4.3.9.16 ADC

Import: ADCPort

Is an optional window to adjust the analog values at the ADC inputs and the voltage levels at the AIN0 and AIN1 pins of the analog comparator.
To read the converted input of channel "i" the program uses "GetADC(i)". With only one channel defined the program has to use "GetADC" (without an "i").
The automatic functions (Sine, Triangle, ...) are not yet implemented and for future enhancements.



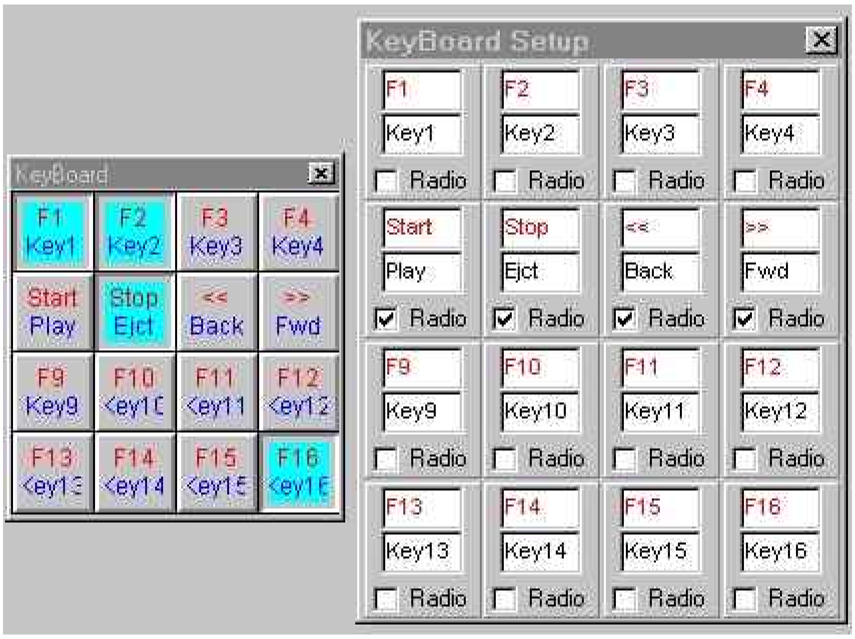
Mode: ● Manual, ○ Sine, ○ Triangle, ○ Square, ○ File

Controllers with 10 bit ADC have 2 result bytes. By default the result is right adjusted (the high byte contains only the 2 most significant bits). Only this mode is supported by the simulator.
The optional left adjusted representation (the high byte contains the 8 most significant bits) is not supported.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.17 KeyBoard 4x4

Import: MatrixPort

Is an optional window to simulate a 16 key matrix keyboard.



F1 Key1 ☐ Radio	F2 Key2 ☐ Radio	F3 Key3 ☐ Radio	F4 Key4 ☐ Radio
Start Play ☑ Radio	Stop Ejct ☑ Radio	<< Back ☑ Radio	>> Fwd ☑ Radio
F9 Key9 ☐ Radio	F10 Key10 ☐ Radio	F11 Key11 ☐ Radio	F12 Key12 ☐ Radio
F13 Key13 ☐ Radio	F14 Key14 ☐ Radio	F15 Key15 ☐ Radio	F16 Key16 ☐ Radio

A right click on any key opens the window "KeyBoard Setup" to enter individual key names and to define *one* set of radio buttons.
There are a lot of functions to read the keys e.g. "ReadKeyBoard".
The keyboard driver has also a memory function to avoid the necessity of a permanent keyboard polling.
Furthermore the system exports a semaphore "KeyBoard" to provide an optimal support of Multi Tasking (see procedure "WaitSema").
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.18 KeyBoard 8x8

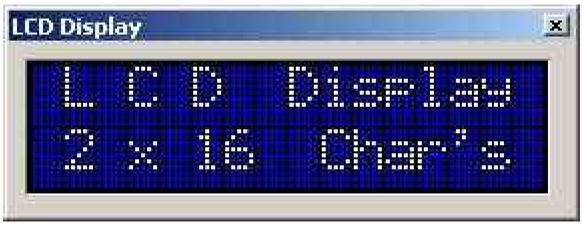
Import: KeyPort8

Is an optional window like the "KeyBoard 4x4". This keyboard has always 8 rows and up to 8 columns.
Radio Buttons are *not* available.
There are a lot of functions to read the keys e.g. "ReadKeyBoard8".
The keyboard driver has also a memory function to avoid the necessity of a permanent keyboard polling.
Furthermore the system exports a semaphore "KeyBoard8" to provide an optimal support of Multi Tasking (see procedure "WaitSema").
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.19 LCD display

Import: LCDport

Is an optional window that simulates one or two LCDs. Supported are LCDs with 1,2 or 4 lines and up to 40 characters per line that are connected to 7 or 8 port pins. Supported display controller types are HD4470, HD66712, KS0070 or KS0073.
By a right click you can select the colours of the background and the pixels (pixel on and off colour).
An output is done by the procedure "LCDout". User defined characters are supported.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).



4.3.9.20 LCD_M display

Import: LCDmultiPort

Is an optional window that simulates up to 8 LCDs. Supported are LCDs with 1,2 or 4 lines and up to 40 characters per line that are connected via a I²C I/O expander. Supported display controller types are HD4470, HD66712, KS0070 or KS0073.
By a right click you can select the colours of the background and the pixels (pixel on and off colour).
An output is done by the procedure "LCDout_M". User defined characters are supported.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.21 7seg display

Import: Disp7sPort

Is an optional window that simulates 1 to 8 7-segment displays. By a right click you can select the colours of the background and the segments.
The possible modes of connection are multiplex mode or via latched shift registers.
An output is done by "DispOut" (as far as possible there are also alphanumerical characters displayed).
Furthermore there are a lot of functions like blinking etc. available and you can create own character sets.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.22 I2C 7seg display

Import: I2C_Disp7

Is an optional window that simulates up to 16 7-segment displays that are connected via I²C expander.
You can build up to 4 groups of displays and each group is limited to max. 8 digits.
By a right click you can select the colours of the background and the segments.
An output is done by "I2C_Disp7Out" (as far as possible are also alphanumerical characters displayed).
Furthermore there are a lot of functions like blinking etc. available and you can create own character sets.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.23 14seg display

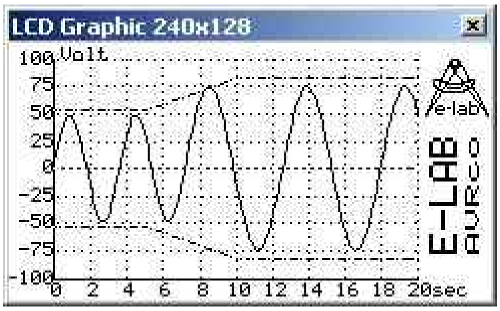
Import: Disp14sPort

Is an optional window that simulates an 2..8 digit 14-segment display that operates in multiplex mode.
By a right click you can select the colours of the background and the segments.
An (alphanumeric) output is done by the procedure "Disp14out". Furthermore there are a lot of functions like blinking etc. available and you can create own character sets.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.24 LCD Graphic

Import: LCDgraphic

Is an optional window to simulate graphical displays. By a right click you can select the colours of the background and the pixels.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

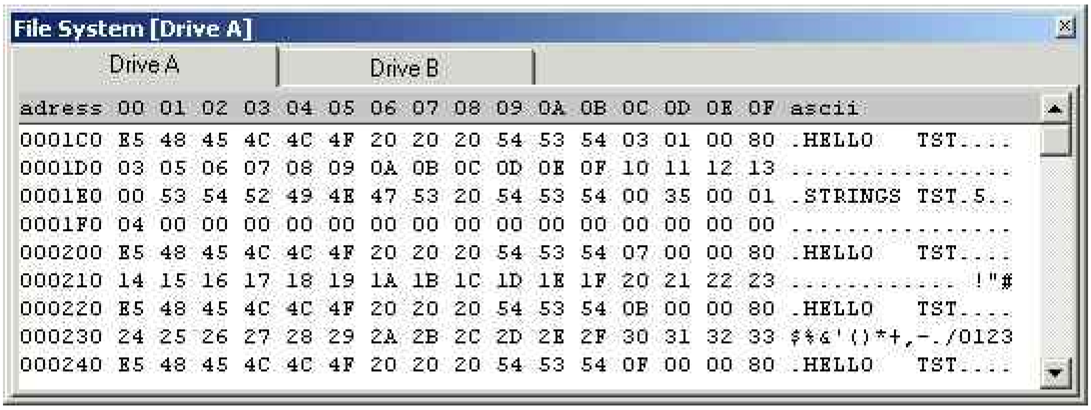


The AVRco comes with a lot of high level functions (textual and graphical). You only have to define the User Device "GraphIOS" containing basic functions (e.g. "write Byte" or "set Pixel").
For many of the most common display controllers (e.g. T6963, SED1531, HD61202...) the "GraphIOS" is already contained in examples.

4.3.9.25 File System

Import: FileSystem

Is an optional window.
The drives are simulatted in the memory of the PC and this window corresponds to a disk editor for the defined drives (max. 4: Drive A to Drive D).
You also can change the contents (in hex or ASCII area). By a right click you have an option to format the drive.



```
adress 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ascii
0001C0 E5 48 45 4C 4C 4F 20 20 20 54 53 54 03 01 00 80 .HELLO   TST....
0001D0 03 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 ................
0001E0 00 53 54 52 49 4E 47 53 20 54 53 54 00 35 00 01 .STRINGS TST.5..
0001F0 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000200 E5 48 45 4C 4C 4F 20 20 20 54 53 54 07 00 00 80 .HELLO   TST....
000210 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 ............ !"#
000220 E5 48 45 4C 4C 4F 20 20 20 54 53 54 0B 00 00 80 .HELLO   TST....
000230 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 $%&'()*+,-./0123
000240 E5 48 45 4C 4C 4F 20 20 20 54 53 54 0F 00 00 80 .HELLO   TST....
```

"FileSystem" is very simple and about 6 kByte Flash and 500 Byte Ram are sufficient.
Suitable hardware could be external SRAM, EEPROM, Flash, a Foppy, a Harddisk etc.
The AVRco comes with a lot of high level functions (e.g. create / delete files, read / write, set file attributes, etc.). You only have to define the User Device "FileIOS" containing the basic functions.
The "FileIOS" for the Atmel Flash AT45DB161 is contained in an example.
As the "FileSystem" has a proprietary structure the devices are *not* compatible to DOS / Windows.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.26 FAT16

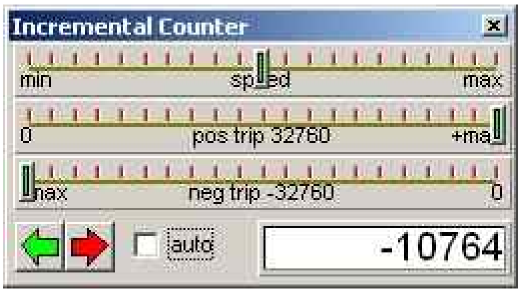
Import: FAT16

Is an optional window.
The drive is simulatted in the memory of the PC and this window corresponds to a disk editor for the logical drive. You also can change the content (in hex or ASCII area). By a right click you have an option to format the drive.
In contrast to the "FileSystem" this drivers is compatible to the PC file system. A min. file system needs about 12k Flash and 1k Ram. Supported are MMC, CF Flash Cards and standard IDE drives.
For other hardware there is a general driver included.
The AVRco comes with a lot of high level functions (e.g. create / delete files, read / write, etc.).
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.27 Incr Counter

Import: IncrPort

Is in optional window to simulate an Incremental Encoder connected to the input of the analog comparator.

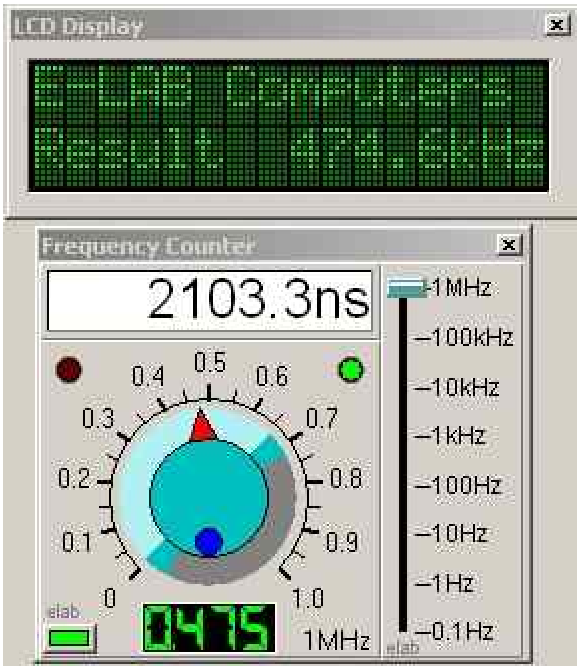


The green/red arrows are used to control the actual value ("turn the encoder"). The driver uses the interrupt so changes are only possible in "Run" mode.
In "auto" mode the encoder runs automatically trough the whole scale. The max. scale depends on the selected resolution (16 or 32 bits). You can use the "pos trip" / "neg trip" to limit the scale. “Speed” is used to set the “rotation speed”.
The driver is internally working with a 16 or 32 bit counter. "Get / Clear / SetIncrementalVal" is used to read / delete /set this counter.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.28 Frequ Counter

Import: FreqCount

Is an optional window that simulates the Frequency Counter driver. Only the first channel can be simulated. Actually there is no simulation of the optional 2nd channel "FreqCount2" available.



With the slide switch on the right you select a frequency range 0...0,1Hz, 0...1Hz, etc. up to 0...1MHz.
The poti is used to select the frequency. The small button on the lower left corner starts / stops the frequency counter.
The frequency counter can also be used for pulse width measurement. The corresponding functions to read the value are "GetFreqCounter" or "GetTimeCounter".
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.29 I2C PortExpand

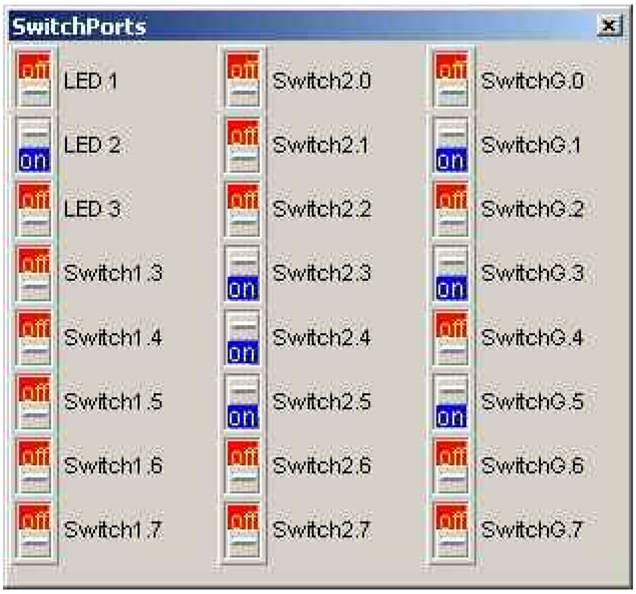
Import: I2Cexpand
Is an optional window to simulate up to 8 ports that are connected via I²C expander. The used PCA9554A expanders have the same internal registers as the processor ports (PORT / PIN / DDR)
and these registers are displayed the same way as the internal ports in the "Ports" window. The symbolic names are "PORT 0...7" / "PIN 0...7" and "DDR 0 ...7". You can also change the value of the port bits. Further controls are not possible.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.30 System Blinker

Import: SysLEDblink
Is an optional window to simulate the SystemLed driver. There are no controls available.
You access the driver via several functions (e.g. "SysLEDon", "SysLEDflashOn" etc.).
With "SysLedFlashMsg" you can signal a blink pattern (e.g. for Error Codes).
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.31 SwitchPort

Import: SwitchPort1 / SwitchPort2 / SwitchPort_G
Is an optional window to simulate the SwitchPort driver. Only the imported SwitchPorts are displayed (the following screen shot shows an example with all SwitchPorts imported).



off	LED 1	off	Switch2.0	off	SwitchG.0
on	LED 2	off	Switch2.1	on	SwitchG.1
off	LED 3	off	Switch2.2	off	SwitchG.2
off	Switch1.3	on	Switch2.3	on	SwitchG.3
off	Switch1.4	on	Switch2.4	off	SwitchG.4
off	Switch1.5	on	Switch2.5	on	SwitchG.5
off	Switch1.6	off	Switch2.6	off	SwitchG.6
off	Switch1.7	off	Switch2.7	off	SwitchG.7

A right click on any key is used to enter individual key names and to define "auto release" keys.
You can read the (debounced) keys with the functions "INP_STABLEx" and "INP_RAISEx" or the
the whole (bebounced) port with "PORT_STABLEx".
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.32 RC5receiver

Import: RC5Rxport

Is an optional window to simulate an infrared receiver of remote controls.
You define in a small window the message to send (to be received).
In welchem Mode? Standard oder Extended? Wenn das abhängig von RC5mode ist, warum sind dann bei "rc_6bit" CMDs > $3F einstellbar? ***Muss ich prüfen*** Kommentar stehen lassen.
Es gibt den 6bit und den 7bit Mode. Bei 6bit ist der maxWert = $3f und bei 7bit = $7f
Bitte bei Gelegenheit nochmal checken! Auch bei 6bit Mode konnte ich Werte > $3f einstellen! **(+rh*)**

You enter the value in hex. A checked "toggle" sets this bit = 1. The button "send" starts / stops the transmission. The function "RecvRC5" reads the received data.
A simulation of a transmitter is actually not implemented.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.33 Servos

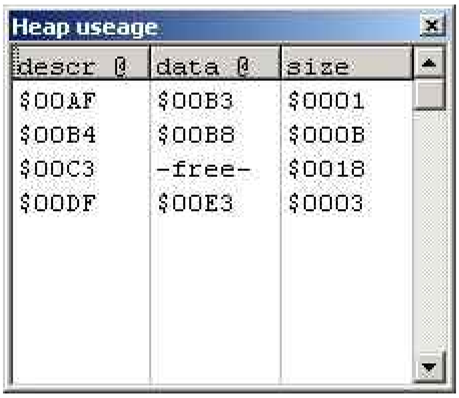
Import: ServoPort

Is an optional window to display up to 8 digital servos. It shows the servo positions of the channels 0 to 7 in percentage of (pos./neg.) maximum range. There are no adjustments to made.
The program controls the servos via the procedure "SetServoChan".
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.34 Heap

Import: Heap

Is an optional windows to show the actual usage of the heap memory.



Heap useage

descr @	data @	size
$00AF	$00B3	$0001
$00B4	$00B8	$000B
$00C3	-free-	$0018
$00DF	$00E3	$0003

You allocate heap memory with the function "GetMem". This function returns a pointer (heap is only accessible be pointers).
The function "FreeMem" is used to deallocate the memory.

Each memory block on the heap uses 4 management bytes. These addresses are shown at "descr@".
"data @" shows the start address of the data area, "size" the number of used data bytes.

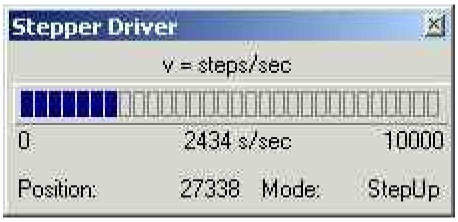
The upper example shows:

- the heap area starts at address $00AF
- the first used block (a single data byte) uses the address range $00AF to $00B3 (including) = 5 byte
- the second block (11 data bytes) uses 15 bytes
- the heap is fragmented (caused by a "FreeMem"). There is a free area between $00C3 and $00DE (including) = 28 bytes = a block of 24 data bytes

In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.9.35 Stepper

Import: StepPort
Is an optional window.



Shows the speed of a stepper motor in steps/sec. as bar and decimal value.
The scaling of the bar graph is between 0 and max. speed entered in the source by the definition of "StepMaxFreq" .

The actual position (in steps) and the actual mode are displayed:

Mode "StepUp":	step ramp up (increasing speed)
Mode "StepDown":	step ramp down (decreasing speed)
Mode "StepStop":	motor is stopped
Mode "StepRun":	stepping with max. speed

There are a lot of functions to control the power drivers:

"StepperOneCW":	one step clock wise
"StepperOneCCW":	one step counter clock wise
"StepRampCW":	step up ramp clock wise

etc.

The StepPort is used to control H-bridge drivers. To control ICs with intelligent builtin functions there is in addition a UserPort available.
In JTAG mode the simulation / debugging is done with the real hardware so this function is not available (makes no sense).

4.3.10 Menu Help

Only the "Info" function to display the actual version of the simulator is implemented.

4.4 SysTick and Scheduler timings

The SysTick is the muscle of the AVRco system. It does most of the background jobs ie. debouncing of ports, readout of the ADC, software RTC, handling some Timeouts, Beeper, Blinker, refresh of LED-Displays, software Timer, Keyboard scan etc. With MultiTasking the Scheduler is also a part of the SysTick. In common, with extreme and heavy loaded systems the SysTick must do an extreme heavy job.

Because the SysTick is always a hardware timer interrupt (Timer0 or Timer2) the global interrupt is completely disabled until the SysTick has finished it's job. In many cases this disabling is not critical because the other interrupts will be somewhat delayed and will be accepted and processed after the SysTick job is done. But with interrupt bursts, for example from a rapidly changing external Int0 etc. it's possible that an interrupt is not processed because the next one from the same source is already pending. The system will not become instable but there will be some missing informations.

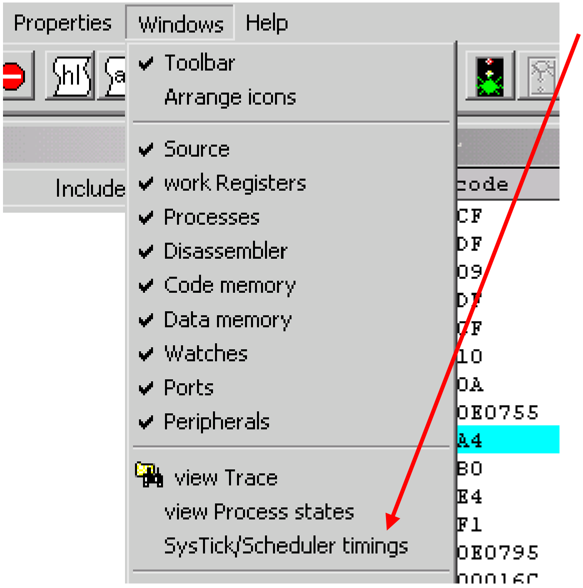
This can be very important with a serial interface (Rx with buffer). If the interrupt is disabled more than two durations of a character transfer the first received character in the UART is overwritten by the second one. New AVRs have a double buffered receive register so this problem is very relaxed now. But also here it is possible to have character losses when working with very high baudrates, heavy duty SysTick job and slow processor clock. So it always makes sense to run the CPU with it's maximum speed. With many driver imports this is a must. The kind of application is not much important, but the processings in the SysTick are the main reasons.

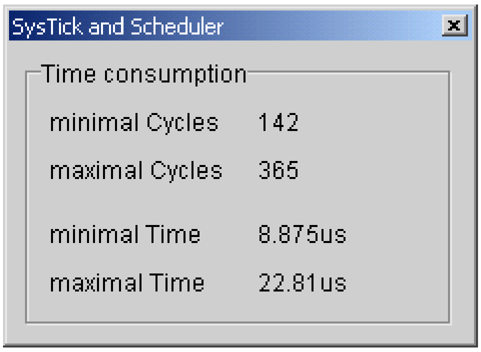
It's impossible to have any formula which shows the interrupt disable time in the SysTick. There are too many different factors, depending of the count and type of the imports. But knowing this time is essential.

To get a coarse number (clock cycles or usec) the AVRco Simulator supports this measuring. While a debug session is running it counts the CPU cycles which have elapsed while the SysTick job is processed. The smallest and the largest values are stored. But this can only be done in the simulation mode, with ICEs and ROM Monitor this is not possible. The debug session should execute all states of the application, all functions and procedures should be executed in order to get real results.

After a sufficient and long simulation the gathered values can be displayed. To do this the menu "Windows"in the Simulator must be opened. Then click the menu item **SysTick/Scheduler timings**.

This opens the window shown below. Here the sampled minimal and maximal interrupt disable times in the SysTicks are shown.




SysTick and Scheduler

Time consumption

minimal Cycles 142

maximal Cycles 365

minimal Time 8.875us

maximal Time 22.81us

The picture shows a maximum disable time of about 23usec. With multiprocessing and slow processor clock and much imports this value can easily exceed 100usec. The consequences then are: also with obviously simple applications it makes sense to let the CPU run at it's maximum speed.

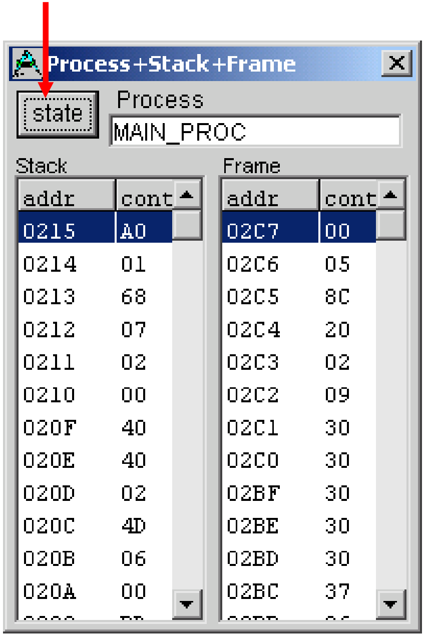
4.5 Determine Frame and Stack usage with the Simulator

An absolute critical (deadly) item with a system design is the definition of the Stack and Frame sizes. Experienced programmers can do an estimation which may work. But with small resources (RAM) a tuning becomes necessary. If the Stack or Frame is too small this can lead to bad surprises, sometimes after weeks or months if several circumstances meet.

A Stack or Frame overrun/violation mostly shows an irregular bevaviour of the program, calculation have wrong results. With text displays destroyed strings are shown. This is because string operations and convertizing is extremely stack and frame intensive.

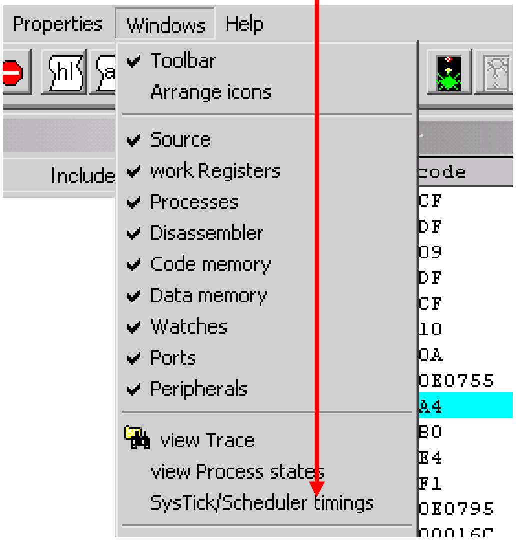
Because there are no common values and also there is no formula for calculation, the necessary stack and frame size must be determined by a Simulator session. But here the missing environment and reality is much more a problem. Some procedures, functions, processes and tasks are only executed if certain external events occur. This is not known by the Simulator. So it's the job of the application programmer to invoke all these events in a proper manner. For example manipulating ports etc. in the Simulator.

The AVRco Simulator not only checks all stack and frame operations during a simulation run but also records the maximum (peak) values separately for all existing stacks and frames. After a substantial simulation run these informations can be displayed in two ways. Either clicking the **state** button of the Stack/Frame window or by the **view Process states** item of the Windows menu.



Process: MAIN_PROC

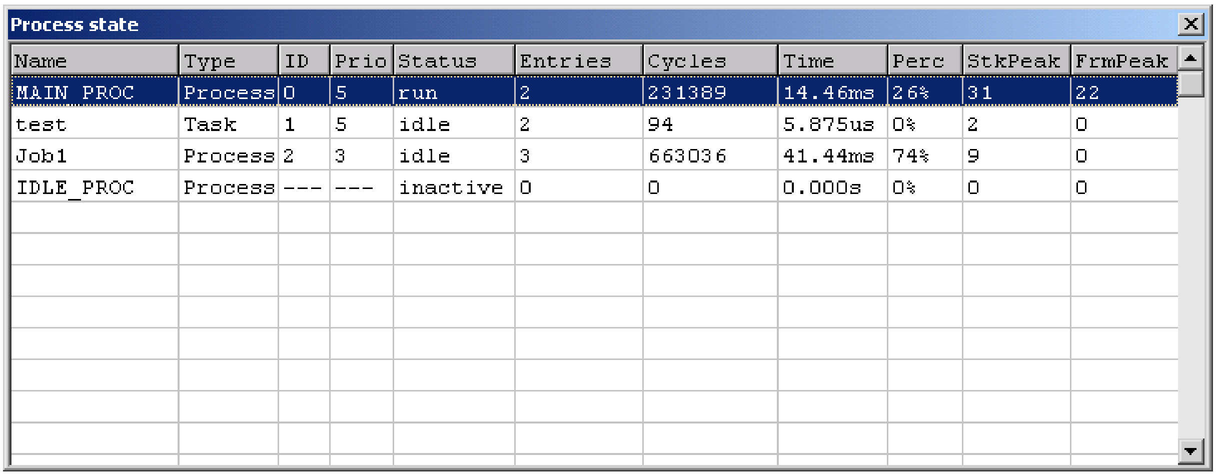
Stack		Frame	
addr	cont	addr	cont
0215	A0	02C7	00
0214	01	02C6	05
0213	68	02C5	8C
0212	07	02C4	20
0211	02	02C3	02
0210	00	02C2	09
020F	40	02C1	30
020E	40	02C0	30
020D	02	02BF	30
020C	4D	02BE	30
020B	06	02BD	30
020A	00	02BC	37



Then the dialog below opens which shows a good overview of the stacks and frames used in this app.

With a long and indeepth simulation run a peak value can be used as a good maximum value.

But to be absolutely sure and for security reasons this value should be increased upto 50%



Name	Type	ID	Prio	Status	Entries	Cycles	Time	Perc	StkPeak	FrmPeak
MAIN_PROC	Process	0	5	run	2	231389	14.46ms	26%	31	22
test	Task	1	5	idle	2	94	5.875us	0%	2	0
Job1	Process	2	3	idle	3	663036	41.44ms	74%	9	0
IDLE_PROC	Process	---	---	inactive	0	0	0.000s	0%	0	0

4.6 Frame and Stack check at runtime

Big processors of the 16 and 32bit range have an internal hardware check to catch stack and frame overflows at runtime and then notify the app by a trap or interrupt. But unfortunately the AVR doesn't have this beneficent feature.

It's possible to implement this in software but first tries showed that the code size is increased by upto 50% and the application slows down in the same range. The reason for this is that **each** PUSH and **each** POP needs a stack pointer check, otherwise this check will be useless. The same is also true with the FramePointer R28/R29.

But with a few small tricks there is a practicable way. It's much less secure as a continously hardware or software check but it's useful.

The checks can be implemented in 2 steps. The first simple one is a system function which exactly knows where each stack or frame begins and where it ends. With a call of this function it checks the entire stack/frame area starting with the lowest address (top-of). An unused byte must show a "00". If this location was used by the stack/frame so it mostly shows a value <> "00". But this is not absolutely true because there could be a push of a "00". So the function checks all memory locations of this stack/frame until a value <> "00" is read. The count of the zeros found determines the obviously unused area. The function returns this value as it's result.

Check of the stack and frame of the main program with a multitasking **and** non-multitasking environment:

```
        Function GetStackFree : word;
        Function GetFrameFree : word;
```

Check of the stack and frame of processes and tasks in a multitasking environment:

```
        Function GetStackFree(prcs : Process) : word;
        Function GetFrameFree(tsk : Task) : word;
```

4.6.1 Extended stack and frame checks

An enhanced version places some patterns (word) at the bottom end of the stacks and frames. If this word has changed so it's clear that an overflow happened. In this case the function always returns with "-1". Otherwise the count of the unused bytes is returned. A small driver must be imported.

```
From System Import LongInt, StackChecks, ..;
```

Check of the stack and frame of the main program with a multitasking **and** non-multitasking environment:

```
        Function CheckStackValid : integer;
        Function CheckFrameValid : integer;
```

Check of the stack and frame of processes and tasks in a multitasking environment:

```
        Function CheckStackValid (prcs : Process) : integer;
        Function CheckFrameValid (tsk : Task) : integer;
```

It must be clear that that it's impossible to detect the reason or location which caused the stack or frame violation. But at least the presence of an overflow can be determined.

An common problem with all such kind of tests is always "what to do" if such a problem happens? A screen message like a PC is impossible because it's not present and will not work because in most cases the system is instable caused by the stack or frame overflow.

But with an ICE this still makes sense. The memory can be read out and assembler single stepping still works. But at least the knowledge that a stack or frame violation happened is very useful.

4.7 JTAG / OWD Debugging

implemented with great support from Victor Chekrygin

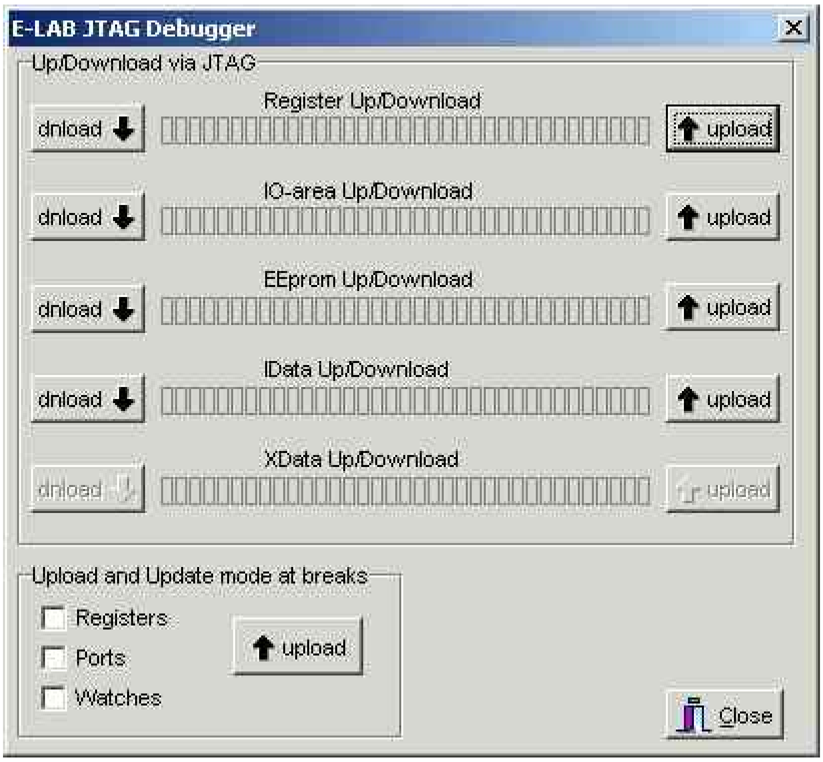
Take care with the programmer software: you must also select the JTAG mode at
"Options – Programmer options"!

4.7.1 UpLoad / DownLoad

An UpLoad/DownLoad of the data takes up to one minute. This is mainly caused by the primitive debug machine of the controllers.

This is the reason why, at a breakpoint, not all data are *automatically* refreshed.
If you need the actual data you must click the green arrow in the toolbar.
Hint: this button is only available if JTAG debugging is active.

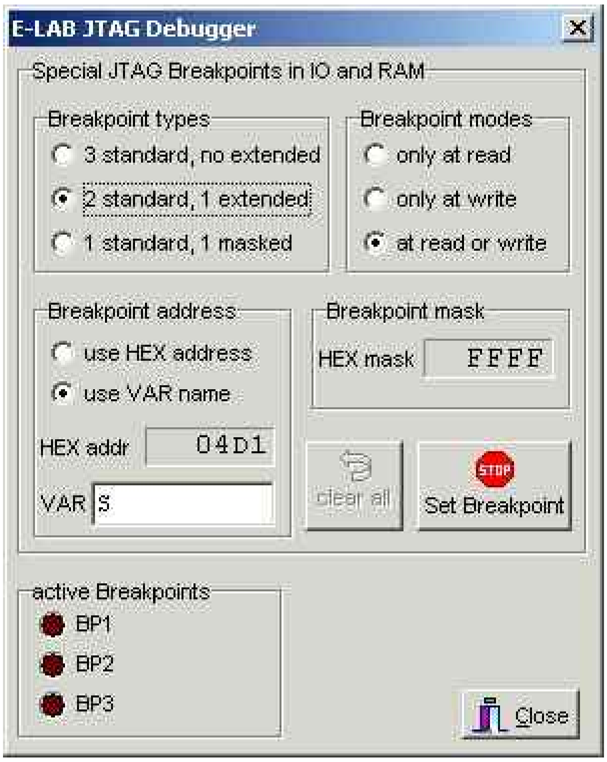
This click opens the window below:



You can select the different areas to "Upload" (from controller to simulator) or to "Download" (from simulator to controller).
By the "check-boxes" at "Upload and Update mode at breaks" you can select an automatic upload of the registers/ports/watches. Note that the *complete* RAM ($IDATA) is *not* updated by this.
By a double click on the register names in the window "work Registers" you can selectively update individual registers and change their values.
A double click in the window "watches" updates all watches. A double click on a complex variable (array / structure) updates the whole variable.

4.7.2 Hardware Breakpoints

With activated JTAG debugging there is another button in the toolbar available: ⊖
It opens a window to define the max. 3 hardware breakpoints in JTAG mode (the maximum number is restricted by the implementation of the debugging system inside the controllers).



E-LAB JTAG Debugger

Special JTAG Breakpoints in IO and RAM

Breakpoint types
- ○ 3 standard, no extended
- ● 2 standard, 1 extended
- ○ 1 standard, 1 masked

Breakpoint modes
- ○ only at read
- ○ only at write
- ● at read or write

Breakpoint address
- ○ use HEX address
- ● use VAR name

HEX addr 04D1

VAR S

Breakpoint mask

HEX mask FFFF

active Breakpoints
- ● BP1
- ● BP2
- ● BP3

"Standard" breakpoints are unrestricted settable in program code.
"Extended" breakpoints are stopping the simulator at a memory access. "Breakpoint modes" select the different accesses.
"masked" breakpoints are also concerning the data memory. You can select a whole area of addresses. A binary 0 in the mask marks a "don't care" bit.
e.g. in the upper example (if you select "1 standard, 1 masked"):
addr = 04D1, mask = FFFF one break condition at 04D1
addr = 04D1, mask = FFFE (bit 0 = don't care): 2 break conditions at 04D0 and 04D1.

4.7.3 Tips and Notes for JTAG Debugging

take care to keep the firmware on your ISP on the actual state !
Most enhancements need a firmware upgrade.
(Firmware Download/Update into the ISP-USB ICE only with external 5Volt supplied to the Target plug!)

The update wait time after a Break or a Step depends of the count and complexity of the watches.
Many watches, much waiting.
The automatic display of local vars in Procedures/Functions can be disabled to save time (right mouse click into the watch window).

The JTAG Debugger optional supports Stack and Frame overflow checks with function calls.
But this is only possible after a single step, a program stop or a breakpoint.
Because also this causes a big data traffic between the PC and AVR this checks are disabled by default.
Enable it in the "Local watch" window with a right mouse click.

Support the debugger in it's heavy duty job and write only one Pascal statement into one source line.
It will respond with a faster stepping rate.

All files of the system must be synchron in order to do a safe JTAG debugging.
Any change in the source code ohne without a re-compile and Flash download can resulkt in a very strange behaviour of the debugger.

In the setup dialog of the JTAG-ICE there is a control element for the USB-Port. It supports fine tuning some PC dependant Timeouts.
The control should be set to long with the first tries.
After having stable connections it can be moved into the short direction.
But the speed improvements are not very high, so please do not overrun the USB connection.

4.8 Compiler Switches and the Consequences

4.8.1 {$D+} {$D-}

Enables / disables debug informations. If disabled the following statements can not be processed in single step mode.

4.8.2 {$E+} {$E-}

If disabled the following statements are not executed by the simulator. E.g. can be helpful at long mDelays

5 Lookup and Interpolate

5.1 Nonlinear functions of sensors

Many sensors and other functions show a nonlinear curve. This means that the relation between an input value and the corresponding output value is not linear. Examples: PT100, PTC, NTC, light-detectors, and also diodes. This is only a very small count of nonlinear sensors. Normally the measure result has a fixed relation to the external events. The temperature for example. But in many cases this relationship is not linear, but logarithmic, cubic etc. A PT100 shows a resistance of 100 Ohms at 0degC, at 50degC 124Ohms and at 100degC 143Ohms. The relation between temperature and the resistance value is nonlinear.

There are two ways to calculate the temperature from the resistance:

1. With a proper formula, which mostly is a complex thing, one can calculate the the temperature which corresponds with the resistance.
2. Build a so called LookUp-Table. Insert in steps resistance values and the related temperature values. Access the table with the resistance value as an index. The result is the temperature. If the input value reaches from 100 to 200 there must be 100 value pairs in the table. More difficulty is a large span of the input values (0 to 1023). Then the table must have 1024 entries.

5.2 Linearising

The herein used implementation is table based, with paired values (search/result). The LookUp algorithm searches with a known value in the table until either this value is found or this argument fits between two values. The searching is done with a binary search function for best speed results.

If a proper value(s) is found it will be linear interpolated. These method allows short tables, dependent of the required accuracy. If the count of the value pairs is relative high, the linear interpolation results in an acceptable accuracy.

5.2.1 System Functions:

```
function InterPolX(const LookUp     : pointer; x : integer; var y : integer) : boolean;
function InterPolX(const LookUp     : pointer; x : longint; var y : longint) : boolean;
function InterPolX(const LookUp     : pointer; x : float; var y : float) : boolean;

function InterPolY(const LookUp     : pointer; y : integer; var x : integer) : boolean;
function InterPolY(const LookUp     : pointer; y : longint; var x : longint) : boolean;
function InterPolY(const LookUp     : pointer; y : float; var x : float) : boolean;
```

The pointer must point into a table in the ROM. The second argument is the search value. It’s type defines the operation (Integer, LongInt or Float) of the functions. The result is placed into the location of the third argument, if the function was succesful.

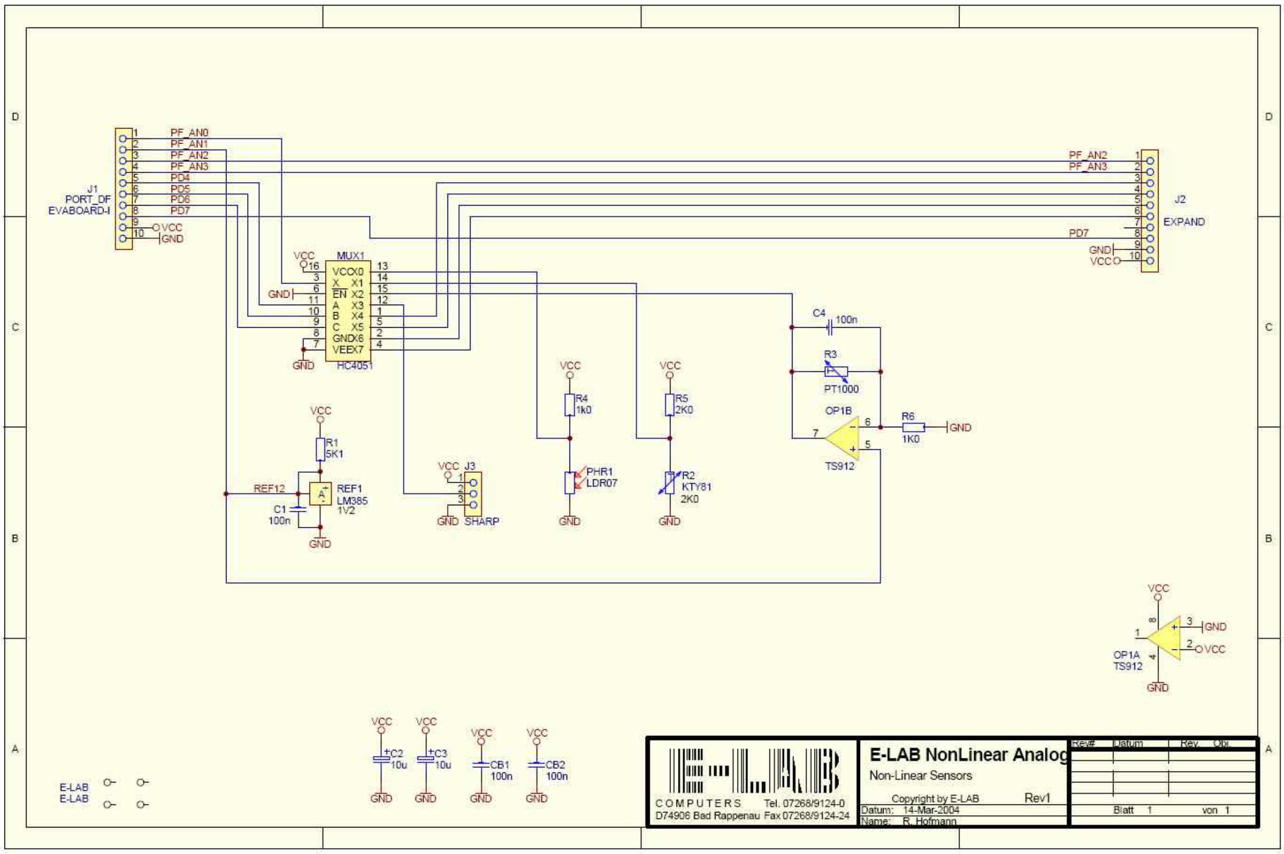
5.3 LookUp Table definition and import

The parameter **const LookUp : pointer** which must be passed to the functions, is a Pointer which must point to a ROM constant table. The table should be defined in this way:

```
const
  IntLookUp   : array[1..(size * 4) + 3] of byte = FileName;
  // size point.x point.y of integer, 3bytes info

  LongLookUp  : array[1..(size * 8) + 3] of byte = FileName;
  // size point.x point.y of longint, 3bytes info

  FloatLookUp : array[1..(size * 8) + 3] of byte = FileName;
  // size point.x point.y of float, 3bytes info
```

The array definition is only a place holder for the binary file which must be loaded. The parameter **Size** defines the count of the x/y-datapairs and the following multiply factor (4 or 8) is the count of bytes needed for one datapair (sizeOf(integer), sizeOf(LongInt) or sizeOf(float)). The three bytes info contain the used data type and the count of data pairs.

Example for non-linear sensors which are predestinated to be used with a Lookup Table linearisation. There is also a sample program in the AVRco installation using this schematic/hardware.

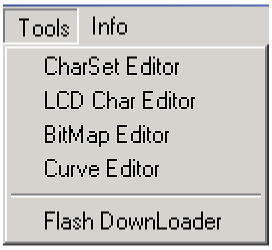


5.4 Creating the LookUp Table with CurveGen

As a support tool for the creation of the lookup table a Table Generator **CurveGen** is included. With it's help one can interactively and graphically create a curve and then store it into a binary file which can be imported into the application.

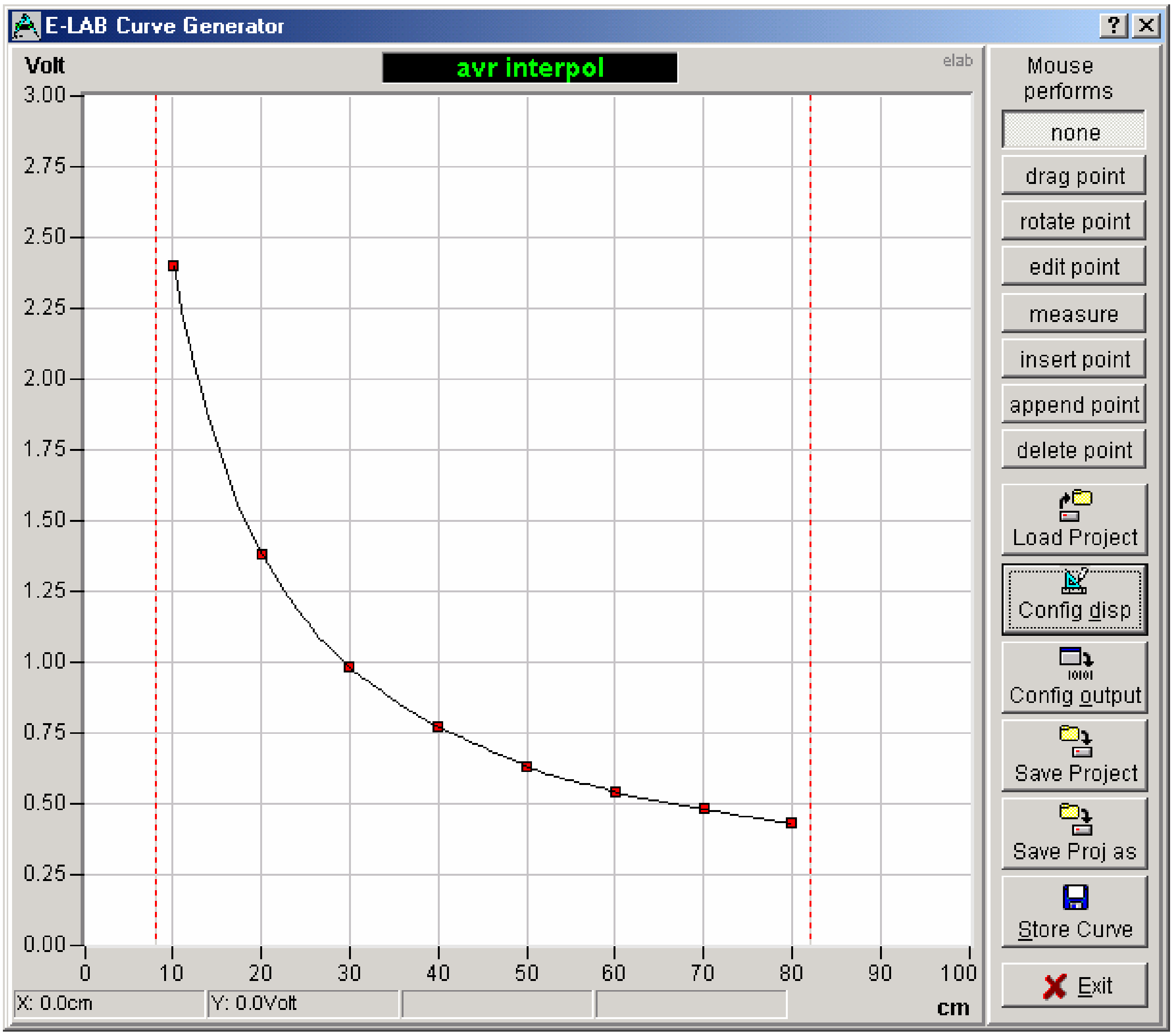
A sample program can be found in the Demos directory as "AVR Interpol". Here an optical distance/proximity sensor (Sharp) is sampled by the ADC and then linearised. The result is displayed in cm. A datasheet can be found in the DOCs directory.

5.4.1 Program start



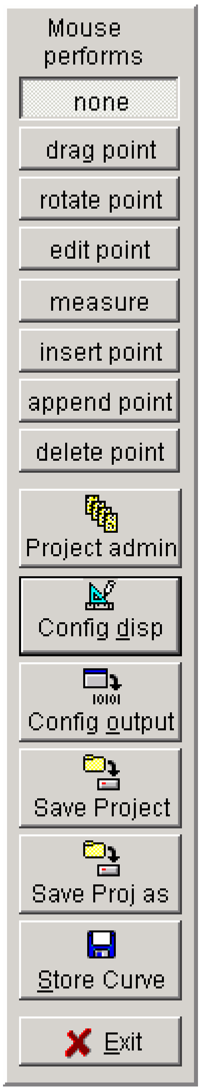
The program *CurveGen.exe* has to be started from the IDE PED32 with the button or with the menu item *curve Editor* on the left. After setting all the needed parameters and setpoints a binary file can be created which is necessary for the interpolation and lookup in the application.

CurveGen is absolutely interactive and graphic based. Because of this there is always an immediately reaction and view when parameters are changed. The curves between the supplied setpoints are calculated with a special spline-like algorithm. So it's also possible to generate a stairway curve which is impossible with a pure spline.



The above picture shows a screenshot of *CurveGen* with the project **AVR Interpol** loaded.
The two vertical red lines define the start and end point for the table generation.

The red squares show the predefined set points.



The upper half of the picture at left shows the properties which can be choosen for mouse clicks and mouse moves.
None means that the mouse clicks and moves have no affect to the display.
Drag point moves a clicked red set point as long as the mouse key is pressed.

Rotate point allows a click to a point and then manipulate the point's direction (vector) and weight by dragging the white attribute circle.
With **edit point** a click to a set point opens a dialog where the exact position (x/y) of a set point can be edited.
Measure allows to move with the left or right mouse button two cross hair cursors which are used as measuring points. The result is shown at the bottom of the display.
With **Insert point** additional set points can be inserted with mouse clicks between two existing set points.
Append point appends with each mouse click a set point to the last right point.
With **delete point** an existing set point can be deleted.

Project admin opens a dialog described later on below.

Config display opens a dialog for defining the display's parameters like scaling, partition etc. See description below.

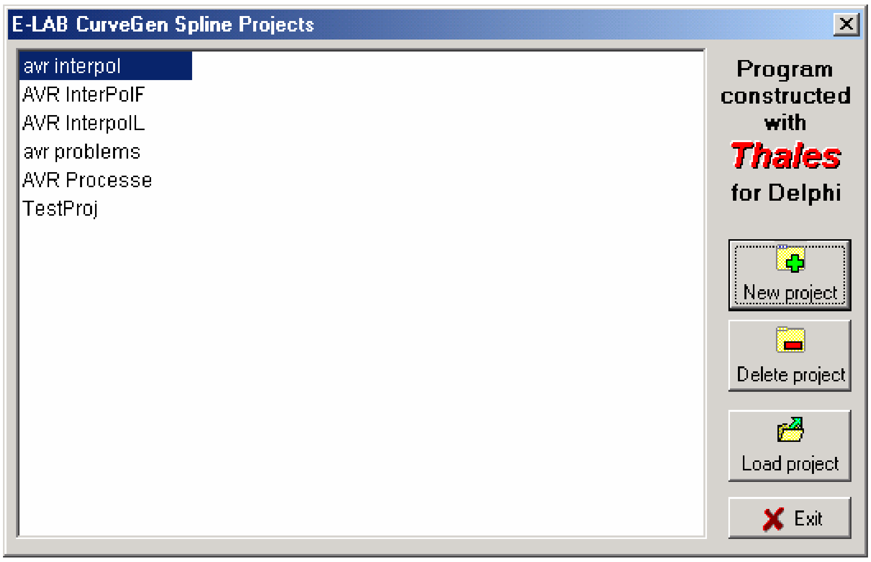
Config output opens a dialog for editing the output/export parameters like data type (integer, longint etc). Description see below.

With **Save Project** all project data and setup is saved in the project file *ProjectName*.inicg

With **Save Proj as** the current project and it's parameters is saved in a new project file.

Store Curve opens the store dialog (see below). A text field shows the generated output in a readable form. So a preview of the generated lookup table is possible.

Exit closes the program and exits.


E-LAB CurveGen Spline Projects
Program constructed with Thales for Delphi
New project
Delete project
Load project
Exit

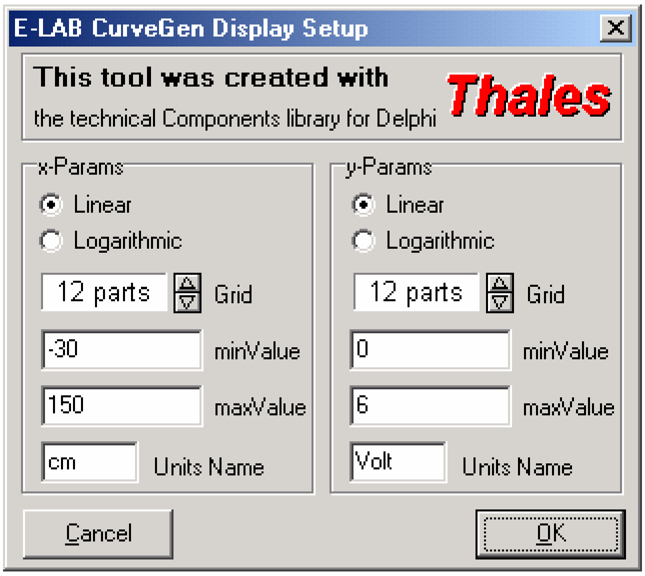
avr interpol
AVR InterPolF
AVR InterpolL
avr problems
AVR Processe
TestProj

The project admin dialog offers besides the loading of existing projects also project deletion and creation of new projects.

With a double click onto a table entry this project is loaded.

With a click to the **New project** button a dialog appers for editing the new project's name and it's working directory.

This new project now has some default settings and a small count of set points. All parameters and setup values now must be changed to fit to the sensor's behaviour.


E-LAB CurveGen Display Setup

This tool was created with Thales

the technical Components library for Delphi

x-Params
- ● Linear
- ○ Logarithmic
- 12 parts Grid
- -30 minValue
- 150 maxValue
- cm Units Name

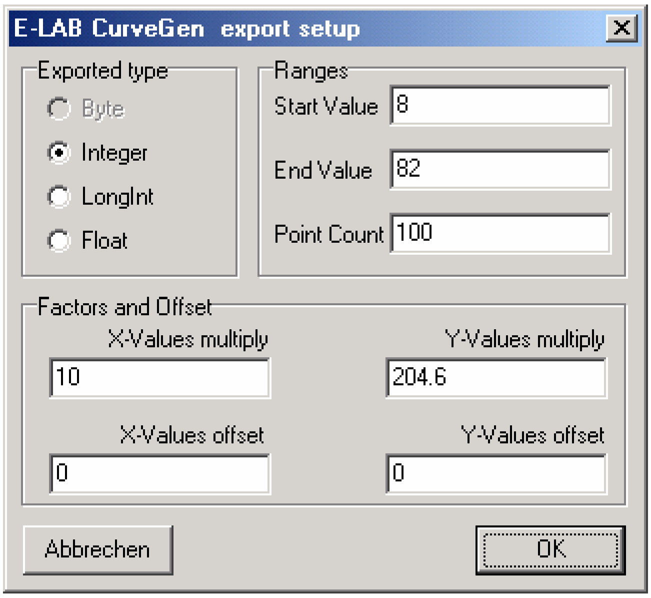
y-Params
- ● Linear
- ○ Logarithmic
- 12 parts Grid
- 0 minValue
- 6 maxValue
- Volt Units Name

The display setup dialog is used to set scalings and partition for the display.

The setting must be done separate for the x and y axis. The scalings can be linear or logarithmic.

The scala partition must be selected so that the text of the scale shows no rounding errors.

The minimal values of the scale are related to the bottom left corner and the maximal values to the upper right corner of the display.


E-LAB CurveGen export setup

Exported type
- ○ Byte
- ● Integer
- ○ LongInt
- ○ Float

Ranges
- Start Value 8
- End Value 82
- Point Count 100

Factors and Offset
- X-Values multiply 10
- Y-Values multiply 204.6
- X-Values offset 0
- Y-Values offset 0

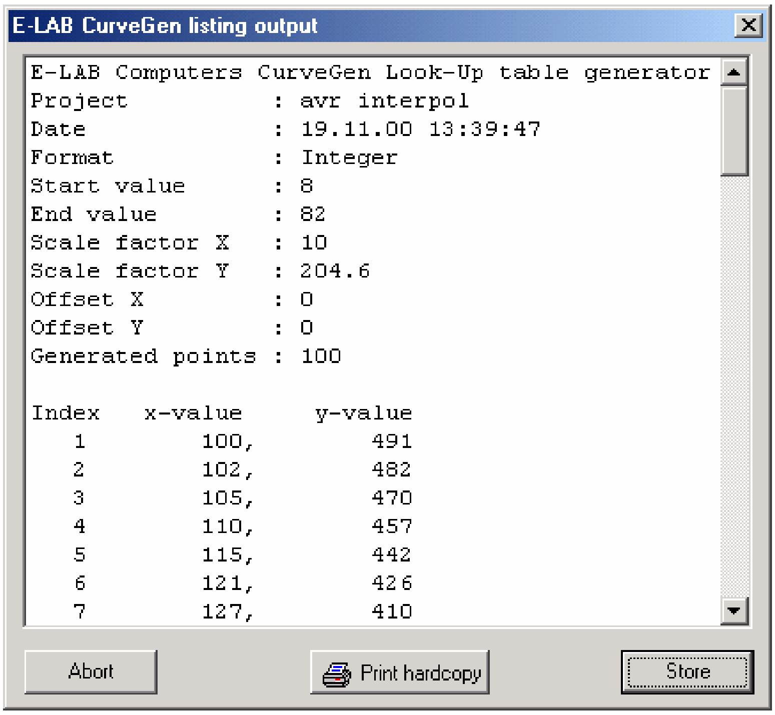
The **Export** dialog on the left serves to edit definitions how the binary file must be build. The data type is defined with **Exported type**

With **Ranges** a part of the display's curve is choosen. This range is displayed by vertical red lines. Start and End Value is only related to the x-axis. Only these points of the curve are exported which reside between these two limiters.

The value **point count** defines value pairs within the selected range should be exported into the binary destination file.

The x and y values can be scaled and offsetted for the export. The calculation will be done with the formula:

```
val:= (val – offset) * factor
```


E-LAB CurveGen listing output

```
E-LAB Computers CurveGen Look-Up table generator
Project           : avr interpol
Date              : 19.11.00 13:39:47
Format            : Integer
Start value       : 8
End value         : 82
Scale factor X    : 10
Scale factor Y    : 204.6
Offset X          : 0
Offset Y          : 0
Generated points  : 100

Index   x-value     y-value
   1        100,        491
   2        102,        482
   3        105,        470
   4        110,        457
   5        115,        442
   6        121,        426
   7        127,        410
```

The **Output** dialog shows all relevant settings, parameters and lists the generated lookup value pairs These list can be printed out.

With the Store button the binary file *ProjectName*.crvg can be written. This file then must be imported into the application source like described above.

5.5 Properties of CurveGen

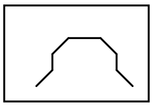
Precision:
This program is based on a special spline function. Spline has the property that the generated curve always touches the set points. This results in the consequence that each error in the coordinates of the **setpoint** atleast at this point becomes effective and also somewhat weaker in it's nearness. Because of this the precision of the interpolated values depend heavily on the precision of the set points.

Less important for the precision is the **count** of the set points. With „normal" quadratic, cubic or logarithmic curves a few points in the range of 5..20 are sufficient. The more komplex a curve is the more set points are necessary. An extreme one could be a logarithmic curve with linear parts and stairways.

Curve shape:
By the count and position of the set points the shape of a curve normally is well defined. But a new curve in a new project still has a strong concentration of the point's attributes and directions to the point itsself. This easily can be seen because the parts between the points are absolutely linear.

These behaviour is not what we expect normally. A curve should have a curve shape ☺. Because of this the attributes of the set points must be adjusted in to proper values. This must be done with the mouse and the selection **rotate point**. With a pressed left mouse button the point must be dragged.

But the point itself isn't move but the visible attributes which are shown by a white curcle. With dragging of such a circle the weight and direction (vector) of the point is changed. The longer the distance between the point and it's white circle is the more the curve is pulled ib this direction. This setup also must be done carefully in order to get a curve which is as similar as possible to the origin curve. But don't fear, the human eye is very well suited to adjust this.



Ambiguousity: Curves must be unambiguous from the point of view of the used LookUp parameter. In the leftside picture there are two x-values for one y-entry value. Which of these values will be returned is uncertain. The similar is true for the x-entry into such a curve. Later implementations of the AVRco LookUp functions support the area limitation for a LookUp so this problem can be handled then.

Generating:
If the curve is completely defined the binary file can be generated. But before this the scaling factors, offsets and the count of the needed curve points must be defined. (**Export** dialog)

The **section** of the curve can be nearly arbitrary. But it makes no sense to set the limits beyond the x-start and end values. In this case only points within the existing x-values of the curve will be exported. If the section is too small propably the interpolation results in inaccurate values.

The offset and scaling factors must be defined in a way that later on the program generates usable values. If for example the input value is read from a 10bit AD-converter, the result from ((curve-y-val – offset) * factor) should be within 0..1023. Similar is true for the x-value. The values of **offset** and **factor** strongly depend of the total gain of the system. This means the gain from sensor to OP-amps upto the AD-converter and its resolution.

The count of points to generate is defined by the dialog field **point count**. Here a compromise is necessary. As an ideal a huge count of points eliminates (or nearly does so) the interpolation and therefore reduces the errors found in the linear interpolation. But this results in huge table with an according long search time. How much points are needed can be found out only with tests with the hardware. An upper limit is the resolution of the data gathering system, eg. AD-converter. With 10bits there are only upto 1024 possibly entry values.

Basically there is one question when the precision has to be checked: which part of the system is the most unprecise one. In most cases the sensor is that one. If this device has large tolerances it makes no sense to trim the software precision to 0.1%.

6 Source-Code-Control-System – SCCS

A problem which is always present with software development is the maintenance of the sources, or better said, the rewinding of the current project state back to the state of date mm.dd. This can become necessary if there is the need to make some changes to an old version of the project which is thought to be history, no more used etc. Also sometimes it is important to switch back to a certain previous version if the current version has accidently moved to a direction, which is obviously wrong (ill, faulty, wrong strategy).

To solve these problems there are some so called Version Control Systems available which are absolutely not cheap. These systems are very complex and working with powerful data bases etc, which results in high costs. But reducing the problem to the essentials, don’t use data bases and complexe operations, this job can also be done with simpler tools by accepting some limitations.

6.1 Overview E-LAB Source-Code-Control-System

The solution integrated into the IDE PED32 from E-LAB is not comparable to the available professional (costly) tools but it does it’s job and is very useful for maintaining the projects build and controlled with the PED32.

The properties of the E-LAB SCCS are:

- The currently loaded version of a project incl. all selected Unit and Include files, the project’s setup etc. can be stored (freezed) with a button click.
- All store operations target a specific directory including datums and time informations.
- The files are not compressed nor keyed so they can be read at any time with a text viewer.
- Each stored version can be viewed in a list and can be restored either into it’s origion directory or any other directory. The project then is useable without any further interventions.
- All operations are done within the IDE PED32. No external programs are necessary or executed.
- Backups are created on demand. There is no automatic protocolling of file changes.

6.2 Strategy of the E-LAB SCCS

All backups (versions) are stored in the subdirectory ***_SCCS_*** below the project’s directory. Here each version has it’s own directory which name consists of ***sccs_*** +***Date*** +***Time***. Example:

sccs_020628_1729

The part ***020628*** is the date 2002.06.28 and ***1729*** the time 17:29. Theoretically it is possible to do a store operation every minute. This directory contains all files which were selected for a version backup of the project.

Files which must be copied for a version backup must be selected in a dialog. An automatic include of all existing files of the project is not possible. It is not necessary that that a selected file is located in the project’s directory, there are no restrictions. But in this case the system has a different strategy in case of a version restore.

If a restore operation is executed and the user decides that the target directory must be the origin directory of the project, all stored files are copied to their origin locations and already existing files with the same name are overwritten.

If the target directory should be different from original directory all stored files are copied into this directory, regardless of the origin location.

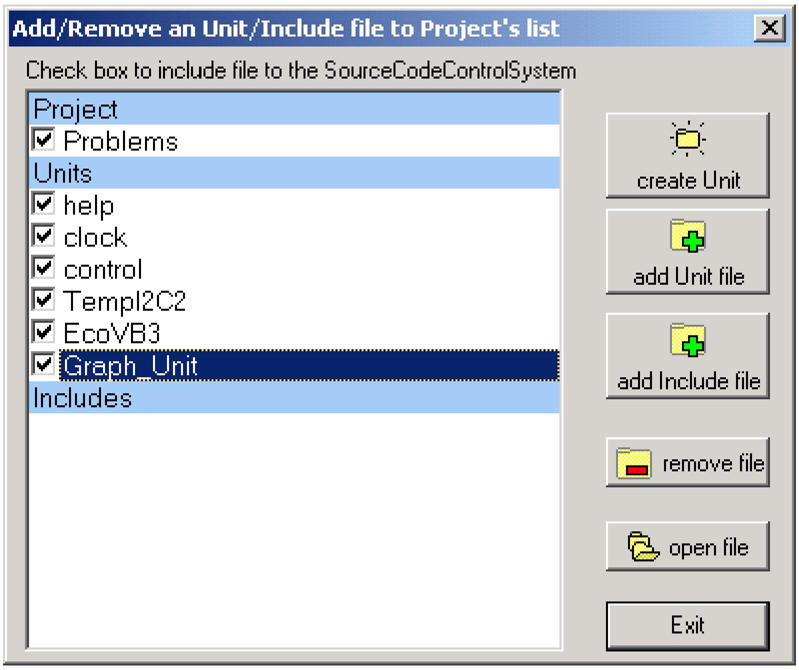
With both cases some ntries in in the project’s control file ‘*dddd.ppro’* will be changed, if necessary

How to...

All operations of the SourceCodeControlSystem (SCCS) are related to actual loaded project.

6.2.1 Version store

If the actual state of a project must be stored as a version the concerned files must be selected at least one time. This must be done with the button which opens the dialog below.



Add/Remove an Unit/Include file to Project's list

Check box to include file to the SourceCodeControlSystem

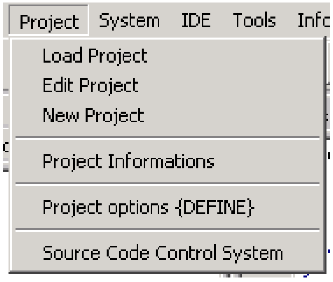
Project
☑ Problems
Units
☑ help
☑ clock
☑ control
☑ Templ2C2
☑ EcoVB3
☑ Graph_Unit
Includes

With ***add Unit file*** or ***add Include file*** a file is added to the project. This file can be reside anywhere, but Unit files must have the extension ***.pas*** and Includes the extension ***.inc***

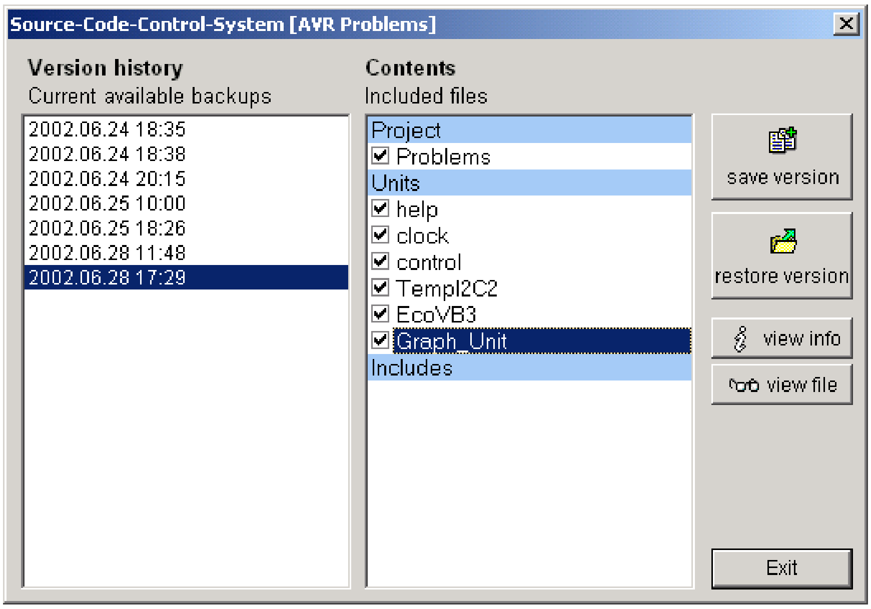
With ***remove file*** the highlighted file is removed from the list.

Now the project administration knows which files belong to this project. Sometimes it makes no sense that all these files are included also into a version backup, so the files which should also included for version backups must be separately selected with their checkboxes.

Now all setups for the version backup of this project are complete. In most cases there is no need to change this setup in the future.



If the current state of the project should be stored as a version the menu item ***Source Code Control System*** opens the SCCS dialog below.



2002.06.24 18:35
2002.06.24 18:38
2002.06.24 20:15
2002.06.25 10:00
2002.06.25 18:26
2002.06.28 11:48
2002.06.28 17:29

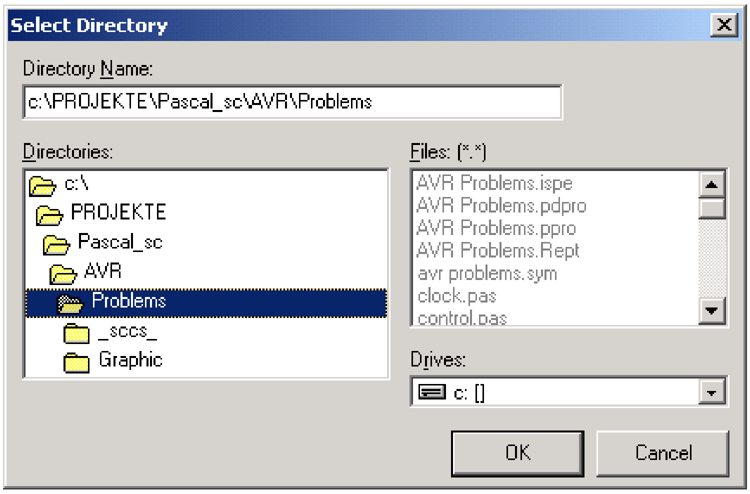
Project
☑ Problems
Units
☑ help
☑ clock
☑ control
☑ Templ2C2
☑ EcoVB3
☑ Graph_Unit
Includes

This dialog shows all known previously stored versionsof the actual loaded projekt. A click onto an entry in the left list field shows it's contained files in the right field. Each backup contains an Info which can be viewed with the ***view info*** button. Each included file can be viewed with ***view file*** button. With the button ***save version*** the actual state (version) of the project is stored into a new sub-directory including date, time and all selected files. Then an editor window pops up where additional informations and user comments can be written which are stored into the info file.

The new additional version is then listed in the version list on the left.

6.2.2 Restore a previous version

In order to restore a previous state (version) of a project the button ***restore version*** in the dialog above must be clicked. The dialog below pops up:



Directory Name: c:\PROJEKTE\Pascal_sc\AVR\Problems

c:\
PROJEKTE
Pascal_sc
AVR
Problems
sccs
Graphic

AVR Problems.ispe
AVR Problems.pdpro
AVR Problems.ppro
AVR Problems.Rept
avr problems.sym
clock.pas
control.pas

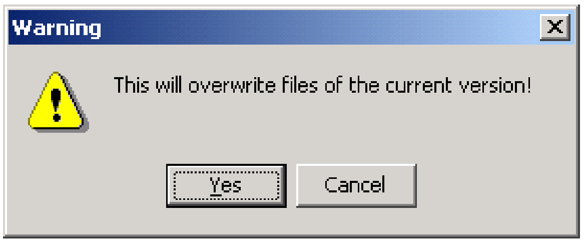
Drives: c: []

It initially shows the origin directory of the project.

Basically there are two ways to proceed with the restoration.

1. Restore into the origin directory
2. Restore into a different existing or new directory.

6.2.2.1 Original Directory

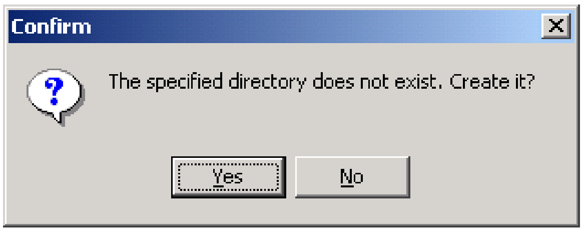


This will overwrite files of the current version!

This is the simplesty way. Click the ***Ok*** button of the directory dialog.

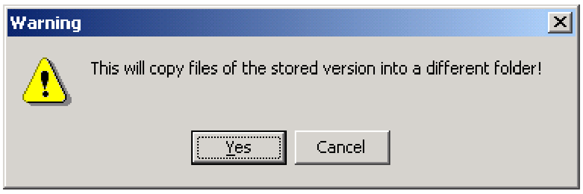
The information dialog on the left appears.
The ***Yes*** button starts the operation, the ***Cancel*** button aborts all.

6.2.2.2 New or different Directory



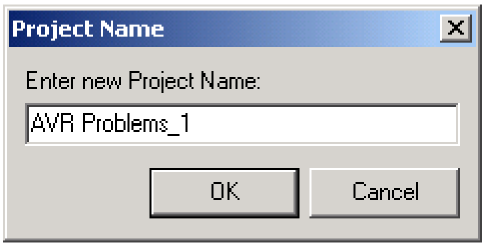
The specified directory does not exist. Create it?

In the topmost edit field of the above directory dialog the desired path and directory of the target must be edited. The quit with the ***Ok*** button. If the selected directory doesn't exist, the dialog on the left appears. Click the ***Yes*** button to accept the directory creation.



This will copy files of the stored version into a different folder!

In both cases the restore and copy operation must be accepted with dialog on the left.



Enter new Project Name:

AVR Problems_1

Because a completely new project is created now using the stored version of the actual project, this new project must have an unique name. The system suggest a new name in the left dialog. This name can be changed by the user.

And then also this operation is finished.

7 Flash Down Loader / Writer

The installation of AVRco contains a PC-based DownLoader Program **FlashLoader.exe**. This is integrated into the IDE PED32.
A click to the button starts the ⚡ tool.

This tool assumes that the Target Monitor communicates through a serial interface. With the help of this tool a new application can be downloaded and programmed into the target system if the Loader Monitor is installed there. It's clear that the Loader in the target must be activated at this time. In the example program **..E-Lab\AVRco\Demos\SelfProg\...** this is accomplished by pressing two special buttons of the keyboard.

The PC-tool can also be used independent of the IDE or Editor. Then there are two possible (optional) commandline parameters:

1. FileName with path (optional). Filename can be a Hexfile (.hex or .eep), a proj-File (.ppro), a Pack file (.pack) or Encr File (.encr).
2. FlashLoader ID (optional). The parameter starts with a % and consists of the decimal ID 0..65536. If this parameter is present the program raises a warning if the preset value differs from the value (ID) in the target system.

If a Flash download is started the tool at first erases the last page below the BootLoader. If there is any ID so it becomes invalid and gets valid again only if a download was successful. The validity can be checked in the Bootloader if the reset of the CPU immediately jumps into the Boot. See the Example below.

The program uses the following communication protocol:

<u>FlashLoader Commando List</u>

? Host requests Loader ID.
Loader responds with
FD FlashDownLoader

A Host sends page adr in word representation. All Flash action rely to this Page
aa1 page addr loByte
aa2 page addr hiByte
aa3 if the target CPU has <u>more than 128kBytes</u> Flash this page extend byte must be send
Loader responds with
CR Command executed

B Host sends EEprom adr in byte representation and EEprom data (byte).
aa1 EEprom adr loByte
aa2 EEprom adr hiByte
data 1 Byte into EEprom
Loader programs this byte into the EEprom and responds with
CR Command executed

C Host sends EEprom adr in byte representation.
aa1 EEprom adr loByte
aa2 EEprom adr hiByte
If an (optional) password is given (see below) the Host must then send the correct password:
pw1 Password loByte
pw2 Password hiByte
Loader reads the EEprom byte from this address in the EEprom and sends it back as the response

D Host stores a new Page of Flash data into the Loader’s buffer
data following (***ps*** x 2) bytes = Pagesize x 2 (mega8..meg16 = 128bytes)
Loader stores this Page into it‘s array and builds an 8Bit checksum by adding all bytes.
Loader sends the computed checksum as a result of the operation to the Host:
cc Checksum

E Host requests erase of the actual Page
Loader erases the actual Page in the Flash and responds with
CR Command executed

I Host requests Loader Info.
Loader responds with
I Info ID = Loader-ID, see below
id1 hiByte Processor ID
id2 midByte Processor ID
id3 loByte Processor ID
ps words pro page
bs1 Bootblock start addr lobyte \
bs2 Bootblock start addr hibyte / = Bootblock start addr in word count
bs3 Bootblock start addr extbyte / = Bootblock start addr in word count if Flash > 128KB

P Host request programming of the Loader’s buffer into the Flash
Loader overwrites the actual Page in the Flash with the content of it’s buffer. A verify is not implemented. This can be done by the Host with reading the Flash page into the Loader’s buffer by the R-Command and then upload the content of this buffer with the U-command.
The Loader responds with:
CR Command executed

R Host requests reading the actual FlashPage into the Loader’s Buffer.
Loader copies the actual Page from Flash into it’s buffer.
The Loader responds with:
CR Command excuted

U Host requests an upload of the content of the Loaders buffer
If an (optional) password is given (see below) the Host must then send the correct password:
pw1 Password loByte
pw2 Password hiByte
The Loader sends (***ps*** x 2) Bytes = Pagesize x 2 (mega8..meg16 = 128bytes).
data
and then an 8Bit checksum, build by the addition of all bytes sent.
cc Checksum

W Host sends a relative page addr in word representation. Then a word follows which the Loader must write into it’s buffer with the use of this address
aa page addr relative (Byte!!)
ww1 loByte
ww2 hiByte
Loader responds with
CR Command executed

X End of communication. The Loader Monitor jumps to the optional user supplied procedure *FlashLoaderExit.* If the system doesn’t find this procedure the loader does a JUMP to 0000h if it terminates.
No response from the Loader.

Loader identification

In most cases the Loader Program is programmed once into the target CPU and remains unchanged at least for a long time. Because of this it's a good idea to place the hardware revision (or similar) of the board or system into it. This 16bit number can be interrogated before a download to make sure that the downloaded firmware is executable on this hardware. This number is included into the machine code as an immediate constant when the BootLoader is generated. This must be done by the definition of a global constant named „DownLoaderID":

```
const
  DownLoaderID : word = 010213;
```

If this constant is not defined this value is stored with $0000 in the Loader.
The ID-number can be recalled at runtime from the loader using the command ‚i'.

i	The Host requests the DownLoader ID.
	Loader responds with
ww1	ID-loByte
ww2	ID-hiByte

There are upto 4 possible ID-words which then are sequential uploaded with the i-command:

```
const
  DownLoaderID   : word = $1234;
  DownLoaderID1 : word = $5678;
  DownLoaderID2 : word = $9ABC;
  DownLoaderID3 : word = $DEF0;
```

Security

In order to disable unauthorized read back of the Flash or EEprom content through the Loader there must be an optional password defined:

```
const
  DownLoaderPWD : word = $1A2B;
```

If a password is defined the upload tool must always provide this password if any upload command is used.

Furthermore it makes sense that a new firmware is not shipped in the hex format but either in the **Pack** or better in the **Encrypt** format of the E-LAB programmer tool. The AVRco Downloader supports also these formats.

7.1.1 BootLoader Example

```
Program SelfProg;

{ $BootRst $01F00}      {reset jumps to here}
{$NOSHADOW}
{ $W+ Warnings}        {Warnings off}
{$DEBDELAY}

Device = mega163, VCC=5;

Import SysTick, FlashWrite, LCDport, MatrixPort;
From System Import ;

Define
        ProcClock       =  8000000;         {Hertz}
        SysTick         =  10;              {msec}
        StackSize       =  $0020, iData;
        FrameSize       =  $0010, iData;
        LCDport         =  PortA;
        LCDtype         =  66712;
        LCDrows         =  4;               {rows}
        LCDcolumns      =  20;              {columns per line}
        MatrixRow       =  PortB, 4;        {use PortB, start with bit4}
        MatrixCol       =  PinB, 0;         {use PinB, start with bit0}
        MatrixType      =  3, 4;            {3 Rows at PortB, 4 Columns at PinB}

Implementation

{$IDATA}
{--------------------------------------------------------------}
{ Type Declarations }
type
  KeyShift =  (F5, F2, arrRight, clear, F4, F1, arrLeft,  point, arrDown, arrUp, F3, shift);

{--------------------------------------------------------------}
{ Const Declarations }
const
  DownLoaderID  :  word = $1234;
//  DownLoaderID1 : word = $5678;
//  DownLoaderID2 : word = $9ABC;
//  DownLoaderID3 : word = $DEF0;

  strC : string = 'Hallo';
  KeyLookUp  : array[key1..key12] of char = ('3', '9', '6', 'E', '1', '7', '4', '0', '2', '8', '5', ' ');

  BootCheck[$3DF8] : word = $AA55;  // last page below bootloader
```

```
{--------------------------------------------------------------}
{ Var Declarations }
{$IDATA}
var
  bb, x, y              : byte;
  bool                  : boolean;
  ch                    : char;
  ww                    : word;
  KeyLookUpV            : array[key1..key12] of char;


{--------------------------------------------------------------}
{ functions }
{$PHASE BootBlock $01F00}
Procedure BootTest;
begin
  // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
  // the following code makes only sense if the Reset jumps to here
  // if the Flash constant "BootCheck" is defined we can check here for a valid last download
  ; check in Flash
  ASM;
    LDI                 _ACCCLO, SelfProg.BOOTCHECKF AND 0FFh
    LDI                 _ACCCHI, SelfProg.BOOTCHECKF SHRB 8
    LPM                 _ACCA, Z+
    CPI                 _ACCA, 055h
    BRNE                BootTestX
    LPM                 _ACCA, Z+
    CPI                 _ACCA, 0AAh
    BRNE                BootTestX
    ;  // normal program start
    JMP                 0000h;
 BootTestX:

  ; // check failed, try to download
  // <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
  FlashDownLoader;
ENDASM;
end;

Procedure FlashLoaderInit;
begin
  ASM;
    ; >> SERPORT Init <<
    ; >> Baudrate 19200Baud <<
    LDI                 _ACCA, 018h                 ; Rx and Tx enable, polling
    OUT                 ucr1, _ACCA                 ;
    LDI                 _ACCA, 019h                 ; 19200 Baud
    OUT                 ubrr1, _ACCA                ;
    SBI                 ucr1, 2                     ; 2 stop bits
    SBI                 ucr1, 0                     ; 2. stopbit = 0
  ENDASM;
end;
```

```
Procedure FlashLoaderRecv;
begin
  ASM;
    SBIS                usr1, 7                          ; Receiver ready?
    RJMP                AVR_SELFPROG.FLASHLOADERRECV   ; if not
    IN                  _ACCA, udr1
  ENDASM;
end;

Procedure FlashLoaderTransm;
begin
  ASM;
    SBIS                usr1, 5                          ; Transmitter ready?
    RJMP                AVR_SELFPROG.FLASHLOADERTRANSM ; if not
    OUT                 udr1, _ACCA
  ENDASM;
end;

Procedure FlashLoaderExit;
begin
  ASM:                  JMP SYSTEM.VectTab;
end;
{$DEPHASE BootBlock}
{--------------------------------------------------------------}
{ Main Program }
{$IDATA}

begin
  LCDcursor (true, false);
  write (LCDout, strC);
  mDelay (1000);
  LCDclr;
  LCDxy (2, 0);
  write (LCDout, 'E-LAB Computers');
  LCDxy (3, 1);
  write (LCDout, 'DownLoad Test');
  LCDxy (3, 2);
  write (LCDout, 'press any Key');
  LCDxy (0, 3);
  write (LCDout, 'Result :');
  LCDxy (9, 3);

  EnableInts;

  loop
    repeat until KeyStatRaised;
    LCDxy (9, 3);
    LCDclrEol;
```

```
if ReadKey(Key12) then
      case KeyShift (GetKeyRaised) of
        F5       : write (LCDout, 'F5');
                 |
        F2       : write (LCDout, 'F2');
                 |
        arrRight : LCDout ('>');
                 |
        clear     : write (LCDout, 'BootTest');
                   BootTest;
                 |
        F4       : write (LCDout, 'F4');
                 |
        F1       : write (LCDout, 'F1');
                 |
        arrLeft  : LCDout ('<');
                 |
        point     : LCDout ('.');
                 |
        arrDown  : LCDout ('v');
                 |
        arrUp     : LCDout ('^');
                 |
        F3       : write (LCDout, 'F3');
                 |
        shift     : write (LCDout, 'Shift');
                 |
      endcase;
    else
      LCDout (KeyLookUp[GetKeyRaised]);
    endif;
  endloop;
end SelfProg
.
```

©1996-2009 ***E-LAB Computers***
Grombacherstr. 27
D74906 Bad Rappenau

Tel. 07268/9124-0
Fax. 07268/9124-24

Internet: www.e-lab.de
e-mail: info@e-lab.de