
BETA Release

Tutorial - Editor - Tools Handbuch

E-LAB AVRco

Pascal Multi-Tasking für Single Chips

Version für

AVR

© Copyright 1996-2009 by E-LAB Computers



Blaise Pascal Mathematiker 1623-1662

3-Jul-2009

Der Inhalt dieses Handbuch ist urheberrechtlich geschützt und ist CopyRight von E-LAB Computers.

Autor Rolf Hofmann
Editor Gunter Baab

E-LAB

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

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

Computers

Wichtige Information

Weltweit wird versucht fehlerfreie Software herzustellen. Die Betonung liegt dabei auf versucht, denn es besteht eine einhellige Meinung, je komplexer eine Software ist, desto grösser die Wahrscheinlichkeit, dass Fehler eingebaut sind.

Wir sind aber nicht der Meinung, dass das ein Grundgesetz ist, und dass man deshalb mit Fehlern und Problemen einfach leben muss (obwohl das bei manchen Software Giganten offensichtlich so ist ☺).

Sollten Sie Fehler feststellen, so wären wir dankbar für jede Information darüber. Wir werden uns bemühen, dieses Problem möglichst kurzfristig zu lösen.

Es ist ebenfalls internationaler Konsens, dass für Folgekosten, die aus fehlerhafter Software entstehen, der Software Hersteller jedwede Haftung ausschliesst, es sei denn es wurde etwas anderes extra vereinbart.

Mit der Benutzung jeglicher Software Produkte von E-LAB Computers schliessen wir als Hersteller sämtliche Haftung aus daraus entstehenden Kosten bei Fehlern der Software aus.

Sie als Anwender bzw. Benutzer der Software erklären Sich damit einverstanden. Sollte das nicht der Fall sein, so dürfen Sie die Software auch nicht benutzen, bzw. einsetzen.

Wie gesagt, dieser Haftungsausschluss ist international Standard und üblich.

Dieses Handbuch und die zugehörige Software ist geistiges Eigentum von E-LAB Computers und damit urheberrechtlich geschützt. Diese Produkte werden dem Erwerber zur Nutzung überlassen. Der Erwerber darf diese Produkte nicht an dritte weitergeben noch weiterveräußern. Weitergabe von Kopien dieser Produkte an Dritte, ob gegen Endgeld oder nicht, ist ausdrücklich untersagt.

Wir meinen dass Sie, als Benutzer der Software, damit Geld verdienen können und damit auch eine Pflege der Produkte erwarten. Ein Produkt, das fast ausschliesslich aus Raubkopien besteht, bringt dem Hersteller/Autor kein Geld ein. Und damit kann ein Produkt auch nicht gepflegt und weiterentwickelt werden.

Es liegt also auch im Interesse des Anwenders, dass das Urheberrecht beachtet wird.

Das wars der Autor

Inhaltsverzeichnis

1	Übersicht	1.1
1.1	AVRco Versionen.....	1.1
1.2	Handbuch Versionen	1.1
1.3	Gliederung der Dokumentation	1.1
2	Tutorial	2.1
2.1	Einführung.....	2.1
2.2	Erstellen und Testen Sie eine Anwendung – eine Einführung: Schritt für Schritt.....	2.2
2.2.1	Herunterladen, Installieren und Starten des AVRco (Demo Version).....	2.2
2.2.2	Erstellen Sie Ihr erstes Projekt	2.4
2.2.2.1	Erstellen Sie ein Programm Gerüst.....	2.5
2.2.2.2	Geben Sie das Programm ein.....	2.8
2.2.2.3	Compilieren und Assemblieren Sie das Programm (= "Make")	2.9
2.2.3	Testen Sie das Programm im Simulator.....	2.10
2.2.4	Geben Sie eigene Texte ein.....	2.12
2.3	Erstellen Sie eine weitere Anwendung – ein etwas tieferer Einblick.....	2.13
2.3.1	Erstellen Sie ein neues Projekt und erzeugen Sie ein Programmgerüst	2.13
2.3.2	Geben Sie das Program ein und führen Sie ein "Make" aus.....	2.17
2.3.3	Einige schöne und nützliche Eigenschaften des Editors.....	2.19
2.3.4	Der Simulator – unverzichtbar auf dem Weg zum Erfolg.....	2.20
2.3.5	wie Sie mehr nützliche Informationen zu erhalten	2.23
2.4	Das interne EEPROM.....	2.25
2.5	Weitere Informationen: Die AVRco Dokumentation	2.26
2.5.1	wie findet man alle verfügbaren Informationen?.....	2.26
2.6	Welche Hardware brauchen Sie?	2.27
2.6.1	Einige zusätzliche Hinweise betreffend die Hardware	2.28
2.7	Was sie brauchen um Ihre Hardware zu programmieren	2.29
2.8	Einsatz des E-LAB Programmers (serieller Programmer, SPI Mode).....	2.31
2.8.1	Das Einstellen der "Programmer Options" für ein neues Projekt.....	2.33
2.9	Nur ganz kurz: Multitasking.....	2.36
2.10	Nützliche Links	2.38
2.11	Anhang.....	2.39
2.11.1	Die AVRco Versionen.....	2.39
2.12	Referenz:.....	2.40
2.12.1	Source File Tutor01.pas.....	2.40
2.12.2	Die Mega8 Konfiguration Bytes	2.42
3	Editor PED32.....	3.1
3.1	Übersicht.....	3.1
3.1.1	Einleitung.....	3.1
3.2	Projekte	3.2
3.3	Controls.....	3.2



AVRco Tools

3.4	Syntax.....	3.2
3.5	Menue.....	3.3
3.5.1	File Menue.....	3.3
3.5.2	Edit Menue.....	3.3
3.5.3	Search Menue.....	3.3
3.5.4	Project Menue.....	3.3
3.5.5	System Menue.....	3.3
3.5.6	IDE Menue.....	3.4
3.5.7	Window Menue.....	3.4
3.5.8	Info Menue.....	3.4
3.6	Dialoge.....	3.4
3.6.1	Project Admin.....	3.4
3.6.2	Project Options.....	3.4
3.6.3	Project Info.....	3.4
3.6.4	General Options.....	3.4
3.6.5	Macro Editor.....	3.5
3.6.6	Zeichen Tabelle.....	3.5
3.6.7	System Admin.....	3.5
3.6.8	File Open.....	3.5
3.6.9	File Save As.....	3.5
3.6.10	Print.....	3.5
3.6.11	Find.....	3.5
3.6.12	Replace.....	3.5
3.6.13	Goto Line.....	3.5
3.6.14	TabSize.....	3.5
3.7	SpeedButtons.....	3.6
3.7.1	FileOpen.....	3.6
3.7.2	Save.....	3.6
3.7.3	Projekt Verwaltung.....	3.6
3.7.4	Application Wizard.....	3.6
3.7.5	Printer Dialog.....	3.6
3.7.6	Cut.....	3.6
3.7.7	Copy.....	3.6
3.7.8	Paste.....	3.6
3.7.9	Find.....	3.6
3.7.10	Replace.....	3.6
3.7.11	Undo.....	3.6
3.7.12	Tile horizontal.....	3.7
3.7.13	Tile vertikal.....	3.7
3.7.14	Cascade.....	3.7
3.7.15	Split Window.....	3.7
3.7.16	Calculator.....	3.7
3.7.17	Projekt Info.....	3.7
3.7.18	Alphabet.....	3.7
3.7.19	Make.....	3.7
3.7.20	Compile.....	3.7
3.7.21	Link.....	3.7
3.7.22	Post Processor.....	3.8
3.7.23	Debugger.....	3.8
3.7.24	Simulator.....	3.8
3.7.25	Assembler.....	3.8
3.7.26	RomSim/Prommer.....	3.8
3.7.27	Tool.....	3.8
3.7.28	Librarian.....	3.8
3.7.29	DisAssembler.....	3.8
3.8	Status Leiste.....	3.8
3.9	Fehler Fenster.....	3.9
3.10	HotKeys und ShortCuts = Tastatur Kommandos.....	3.9

3.10.1	IDE und Syntax Help.....	3.9
3.10.2	Datei und Fenster Operationen	3.9
3.10.3	Cursor bewegen.....	3.9
3.10.4	Editieren	3.9
3.10.5	Suchen/Austauschen	3.10
3.10.6	Cursor Block Befehle.....	3.10
3.10.7	Block bearbeiten	3.10
3.10.8	Diverse	3.10
3.10.9	Keyboard Macros.....	3.10
3.11	Projekte.....	3.11
3.11.1	Arbeiten mit Projekten.....	3.11
3.11.1.1	Load Project	3.12
3.11.1.2	Edit/New Project	3.13
3.11.1.3	Project Path.....	3.14
3.11.1.4	MainFile.....	3.14
3.11.1.5	Application Wizard.....	3.15
3.11.1.6	Template	3.15
3.11.1.7	Project Options.....	3.15
3.11.1.8	System Options	3.16
3.11.1.9	Project Information.....	3.16
3.12	Controls	3.17
3.12.1	Was ist ein Control?	3.17
3.12.2	Control Edit.....	3.18
3.12.3	System Options.....	3.19
3.12.4	Error File define	3.20
3.13	Syntax	3.21
3.13.1	Syntax edit.....	3.21
3.14	Editor Setup.....	3.23
3.14.1	Fonts, Farben, Backup und Schnellhilfe.....	3.23
3.14.2	Zeichentabelle.....	3.24
3.14.3	Keyboard Macros.....	3.25
3.15	Menues.....	3.26
3.15.1	File Menue	3.26
3.15.1.1	History	3.26
3.15.1.2	Insert File.....	3.26
3.15.1.3	Save Block	3.26
3.15.2	Edit Menue.....	3.27
3.15.3	Search Menue.....	3.27
3.15.4	Project Menue.....	3.27
3.15.5	System Menue	3.27
3.15.6	IDE Menue.....	3.28
3.15.6.1	Tabs.....	3.28
3.15.7	Window Menue	3.28
3.15.8	Info Menue.....	3.28
3.15.8.1	Help IDE.....	3.29
3.15.8.2	Help Syntax	3.29
3.15.8.3	Info IDE	3.29
3.15.8.4	Info Syntax	3.29
3.15.8.5	About... und Compiler Registrierung.....	3.29
4	Simulator / Debugger	4.1
4.1	Allgemeines.....	4.1
4.2	Übersicht - die Oberfläche.....	4.1
4.2.1	die Kopf Zeile	4.2
4.2.2	die Menu Zeile	4.2
4.2.3	die Toolbar.....	4.2
4.2.4	der Arbeitsbereich	4.3



AVRco Tools

4.2.5	die Status Zeile	4.5
4.3	die Bedienung des Simulators	4.5
4.3.1	Menu Projekt.....	4.5
4.3.1.1	Open / Save / Save as / Print / Printer Setup / Close	4.5
4.3.1.2	Reload / Reload EEPROM	4.5
4.3.2	Menu Breakpoints	4.6
4.3.2.1	Show list.....	4.6
4.3.2.2	Reset all Breakpoints.....	4.7
4.3.2.3	Stop after	4.7
4.3.2.4	Stop on Schedule, Stop on TASK kill	4.7
4.3.2.5	Memory Write Breakpoints.....	4.8
4.3.2.6	Test I/O	4.10
4.3.3	Menu Watches.....	4.10
4.3.3.1	Add Watch.....	4.10
4.3.3.2	Delete all Watches.....	4.11
4.3.3.3	Popup Raw Display	4.11
4.3.3.4	default Watch representation	4.11
4.3.4	Menu Run	4.11
4.3.4.1	Reset processor Ctrl+F2.....	4.11
4.3.4.2	Go F9.....	4.11
4.3.4.3	Goto cursor pos F4.....	4.12
4.3.4.4	Stop simulation F2.....	4.12
4.3.4.5	Step into F7.....	4.12
4.3.4.6	Step over F8.....	4.12
4.3.4.7	Step out F6.....	4.12
4.3.4.8	Multiple Steps Shift+F9.....	4.12
4.3.4.9	Animate Ctrl+F9.....	4.12
4.3.4.10	Multiple step value	4.12
4.3.4.11	Animation speed.....	4.12
4.3.4.12	Enable Trace ASM / Enable Trace HLL.....	4.13
4.3.4.13	Clear Trace buffer.....	4.13
4.3.4.14	Call Stack Ctrl+F3.....	4.13
4.3.5	Menu Extern	4.13
4.3.5.1	Interrupts	4.13
4.3.6	Menu Search	4.14
4.3.6.1	Show Code at.....	4.14
4.3.6.2	Show Data at.....	4.14
4.3.6.3	Search Code hex pattern.....	4.14
4.3.6.4	Search Data hex pattern.....	4.14
4.3.6.5	Search in Source F3.....	4.14
4.3.7	Menu Configure	4.14
4.3.7.1	Show Hints	4.14
4.3.7.2	Save as default.....	4.14
4.3.7.3	Config with default.....	4.15
4.3.7.4	COMport [ICE..Monitor]	4.15
4.3.8	Menu Properties.....	4.16
4.3.8.1	Short mDelay.....	4.16
4.3.8.2	Fast RTC.....	4.16
4.3.8.3	Short Beep	4.16
4.3.9	Menu Windows	4.16
4.3.9.1	Toobar.....	4.16
4.3.9.2	Arrange icons	4.16
4.3.9.3	Source.....	4.17
4.3.9.4	Work Registers	4.18
4.3.9.5	Processes.....	4.19
4.3.9.6	Disassembler.....	4.19
4.3.9.7	Code memory	4.20
4.3.9.8	Data memory.....	4.20
4.3.9.9	Watches	4.20
4.3.9.10	Ports.....	4.20
4.3.9.11	Peripherals	4.20

4.3.9.12	View Trace	4.21
4.3.9.13	View Process States	4.21
4.3.9.14	SysTick / Scheduler Timings	4.22
4.3.9.15	Terminal I/O	4.22
4.3.9.16	ADC	4.23
4.3.9.17	KeyBoard 4x4	4.24
4.3.9.18	KeyBoard 8x8	4.24
4.3.9.19	LCD display	4.25
4.3.9.20	LCD_M display	4.25
4.3.9.21	7seg display	4.25
4.3.9.22	I2C 7seg display	4.25
4.3.9.23	14seg display	4.26
4.3.9.24	LCD Graphic	4.26
4.3.9.25	File System	4.27
4.3.9.26	FAT16	4.27
4.3.9.27	Incr Counter	4.28
4.3.9.28	Frequ Counter	4.28
4.3.9.29	I2C PortExpand	4.29
4.3.9.30	System Blinker	4.29
4.3.9.31	SwitchPort	4.29
4.3.9.32	RC5receiver	4.30
4.3.9.33	Servos	4.30
4.3.9.34	Heap	4.30
4.3.9.35	Stepper	4.31
4.3.10	Menu Help	4.31
4.4	SysTick und Scheduler Timings	4.32
4.5	Frame und Stack Verbrauch im Simulator ermitteln	4.33
4.6	Frame und Stack Check zur Laufzeit	4.34
4.6.1	Erweiterte Stack/Frame Checks	4.34
4.7	JTAG / OWD Debugging	4.35
4.7.1	UpLoad / DownLoad	4.35
4.7.2	Hardware Breakpoints	4.36
4.7.3	Tips und Bemerkungen zu JTAG Debugging	4.37
4.8	Compiler Schalter und deren Auswirkungen	4.37
4.8.1	{D+} {D-}	4.37
4.8.2	{E+} {E-}	4.37
5	LookUp und Interpolation	5.1
5.1	Nichtlineare Funktion von Sensoren	5.1
5.2	Linearisierung	5.1
5.2.1	System Funktionen:	5.1
5.3	LookUp Table Definition und Import	5.2
5.4	Erzeugung der LookUp Table	5.2
5.4.1	Programm Aufruf	5.2
5.5	Eigenschaften von CurveGen	5.6
6	Source-Code-Control-System - SCCS	6.1
6.1	Einführung Source-Code-Control-System	6.1
6.2	Arbeitsweise des E-LAB SCCS	6.1
6.2.1	Version abspeichern	6.2
6.2.2	Version wieder herstellen	6.3
6.2.2.1	Original Directory	6.4
6.2.2.2	Neue oder andere Directory	6.4



AVRco Tools

7	Flash Down Loader / Writer	7.1
7.1	FlashLoader Beispiel	7.4

1 Übersicht

1.1 AVRco Versionen

alle **AVRco Versionen** unterstützen alle AVR Controller die ein internes RAM (für den Stack) besitzen, also praktisch die gesamte Palette.

AVRco Profi Version:

die Profi Version enthält alle verfügbaren Treiber, darunter auch sehr komplexe wie z.B. ein FAT16 File System oder eine umfangreiche Library für graphische LCDs.

Weiterhin wird die professionelle Programm Erstellung durch den vollen Support von Units unterstützt.

AVRco Standard Version:

in der Standard Version sind nur die besonders komplexen Treiber nicht enthalten.

AVRco Demo Version:

auch die Demo Version unterstützt alle Controller und besitzt alle Treiber der Standard Version. Die **einzige Einschränkung** ist die Limitierung des erzeugten Code auf eine Größe von 4 k.

1.2 Handbuch Versionen

Abschnitte die mit dem Attribut (*P*) gekennzeichnet sind, sind nur in der **AVRco Profi Version** enthalten.

Abschnitte die mit dem Attribut (*4*) gekennzeichnet sind, sind erst ab der **AVRco Revision 4** enthalten.

1.3 Gliederung der Dokumentation

..\E-Lab\DOCs\DocuCompiler.pdf:

enthält die Pascal Sprachbeschreibung und deren Erweiterungen gegenüber dem Standard Pascal

..\E-Lab\DOCs\DocuStdDriver.pdf:

enthält die Beschreibung der Treiber die sowohl in der Standard, also auch in der Profi Version vorhanden sind

..\E-Lab\DOCs\DocuProfiDriver.pdf:

enthält die Beschreibung der Treiber die ausschließlich in der Profi Version vorhanden sind

..\E-Lab\DOCs\DocuReference.pdf :

enthält eine Kurzreferenz (die im wesentlichen mit der Online Hilfe identisch ist)

..\E-Lab\DOCs\DocuTools.pdf:

enthält die Beschreibung der integrierten Entwicklungsumgebung, des Simulators, ein Tutorial usw.

..\E-Lab\IDE\DataSheets\Release-News.txt:

listet die Erweiterungen in chronologischer Reihenfolge auf.

Die Dokumentation der Erweiterungen erfolgt in den oben erwähnten .pdf Files (DocuXXX.pdf)

..\E-Lab\AVRco\Demos\ :

enthält sehr viele Test und Demo Programme

..\E-Lab\DOCs\ :

enthält die Dokumentation sowie weitere Schaltpläne und Datenblätter

2 Tutorial

von Gunter Baab

2.1 Einführung

Alles was Sie brauchen, um dieses Tutorial durchzuarbeiten, ist die kostenlose Demo-Version des AVRco. Der AVRco ermöglicht Ihnen, Applikationen für alle handelsüblichen AVR Controller Typen zu entwickeln. Mit dem AVRco Simulator können Sie Ihre Software "debuggen" und das Verhalten unter einer graphischen Oberfläche überprüfen. Für einen ersten Eindruck schauen Sie sich doch mal die Abbildungen in den Kapiteln den Simulator betreffend an!

Wenn Sie jedoch über diese Tutorial hinausgehen und eine eigene Mikro-Controller Schaltung aufbauen wollen brauchen Sie Kenntnisse in mindestens vier Bereichen:

1. der Programmiersprache Pascal im Allgemeinen und dem Unterschied, den Spezialitäten, sowie den Erweiterungen des AVRco Pascal
2. der internen Struktur des benutzen Controllers
3. der Bedienung des AVRco Pakets um Ihre Programme zu erstellen, zu kompilieren und zu debuggen
4. mehr oder weniger Hardware Kenntnisse, je nachdem Sie eine eigene Schaltung und Programmier-Hardware aufbauen oder auf industriell gefertigte Teile zurückgreifen wollen

Dieses Tutorial behandelt nur Teil 3: *die Bedienung des AVRco*

Die anderen Punkte werden höchstens berührt.

Das erste Kapitel beschreibt die Installation der AVRco Demo Version.
Die einzige Einschränkung der Demo: die Demo erzeugt max. 4k Programm Code.

Vielleicht geht es Ihnen jetzt wie mir am Anfang und Sie fragen:
OK. 4k Programm Code. Nur was heißt das wenn ich eine eigene Anwendung schreiben will?
Was kann ich denn mit gerade mal 4k und (der Hochsprache) Pascal anfangen?
Ist das genug um ein paar Schalter abzufragen und einige LEDs anzusteuern?
Oder reicht das sogar um ein LCD und eine Tastatur-Matrix zu bedienen?
Was bedeutet die Benutzung eines solchen Compilers – wieviel "Overhead" produziert er?

Die Antwort, die ich bekam, war nicht sehr hilfreich: "mach Dir keine Sorgen, der Compiler ist sehr effizient und erzeugt sehr kompakten Code".

Wenn Sie sich auch solche Fragen stellen schauen Sie sich mal das Kapitel "*Multitasking*" an!

Diese Demo benutzt ein LCD, 8 (entprellte) Schalter, ein 6-stelliges 7-Segment-Display, 6 LEDs und den Multi-Tasking Kernel des AVRco und benötigt etwa 85% des verfügbaren Programmspeichers. Damit geht diese Demo an die Grenze des mit 4k Möglichen.
Aber Ihre ersten Anwendungen werden selten so viele Treiber benötigen und jeder Treiber erlaubt Ihnen komplexe Funktionen einfach aufzurufen. Das reduziert den benötigten Platz für Ihr eigenes Programm auf ein Minimum (vielleicht einige Hundert Byte).



AVRco Tools

Ich hoffe, das gibt Ihnen ein gewisses Gefühl für die Grenzen der Demo-Version. Wenn Sie mit kleinen Anwendungen (vielleicht 2,3 Treiber) beginnen, reicht der restliche Speicher sicher locker Sie ein oder zwei Wochenenden zu beschäftigen um diesen mit einem sinnvollen eigenem Programm zu füllen.

Die Dokumentation des AVRco finden Sie in **C:\E-Lab\DOCs\DocuCompiler.pdf**. Insbesondere wenn Sie das 2. Beispiel durcharbeiten, empfehle ich dringend diese Datei auszudrucken und als Referenz zu benutzen. Hier wird häufig darauf Bezug genommen.

Die in der Demo-Version enthaltenen Treiber sind in **C:\E-Lab\DOCs\DocuStdDriver.pdf** beschrieben.

2.2 Erstellen und Testen Sie eine Anwendung – eine Einführung: Schritt für Schritt

Wenn der AVRco neu für Sie ist, sollten Sie dieses Kapitel durcharbeiten. Das Ziel ist, möglichst rasch eine lauffähige Arbeitsumgebung zu schaffen und Ihnen einen schnellen Eindruck über die Möglichkeiten des AVRco zu vermitteln. Sie brauchen dazu nicht eine Menge Pascal Befehle einzugeben. Dafür steht eine Quelldatei zur Verfügung, die sie einfach übernehmen können.

Was soll die Anwendung tun?

mit 8 Tasten werden verschiedene Nachrichten ausgewählt, die auf einem 2x16 LCD dargestellt werden. Tasten und LCD sind an einen Mega8 Controller angeschlossen. Diese Hardware wird mit dem AVRco Simulator simuliert.

2.2.1 Herunterladen, Installieren und Starten des AVRco (Demo Version)

Laden Sie sich die Demo-Version des AVRco von <http://www.e-lab.de> herunter und starten Sie die AVRdemo.exe. Diese selbst-entpackende Datei führt Sie wie üblich durch den Installationsprozeß. (1,2).

Klicken Sie *Install* und wählen Sie die gewünschte Sprache für die Dokumentation aus. Ändern zunächst **nicht** die Vorgabe für das *target directory* C:\ um den Compiler nach C:\E-Lab zu installieren. Klicken Sie OK um eine Programmgruppe AVRco zu erstellen und dann OK um das Hinweis-Fenster zu schließen.

Schließen Sie alle Fenster und klicken Sie *Start – Programme – AVRco – E-LAB PED-32* um die IDE (Integrated Development Environment / Integrierte Entwicklungsumgebung) zu starten. Die PED32 (Programmers Editor für WIN32), die als Plattform für die zahlreichen Funktionen des AVRco (Compiler, Assembler, Simulator u.v.m) dient, wird gestartet.

Bem.:

- (1): machen Sie sich keine Gedanken, wie Sie den AVRco ggf. wieder los werden. Der AVRco schließt einen De-Installer ein, der alle Änderungen an Ihrem PC wieder rückgängig machen kann.
- (2): siehe Anhang für die Unterschied und Einschränkungen der verschiedenen AVRco Versionen

AVRco Tools



The screenshot shows the AVRco Tools software interface. The main window has a menu bar (File, Edit, Search, Project, System, IDE, Tools, Info) and a toolbar. A 'Project Administration' dialog box is open, displaying a list of projects under the 'Project load/delete' tab. The list is as follows:

Project Name	Project Name	Project Name	Project Name
Accu Charger	AVR EvaBoardI	AVR I2C_7s	AVR LinkedList
ACCU checker	AVR FAT16test	AVR I2Cexpand	AVR Mega161
ADC AVR	AVR FATMMC	AVR I2Cloop	AVR Mega163
App 2313	AVR FileFlash	AVR Increment	AVR RC5rx
App 8515	AVR FileSys	AVR Interpol	AVR RC5tx
App Mega	AVR FreqCount	AVR Interrupts	AVR RTC
AVR AVFILTER	AVR Graph1531	AVR IOexpand	AVR RTclock
AVR Banking	AVR Graph61202	AVR Keyboard8	AVR RTclock8
AVR BarGraph	AVR GraphClock	AVR LANmaster	AVR SelfProg
AVR BitSets	AVR GraphCurve	AVR LANslave	AVR Servo
AVR Const_Funcs	AVR GraphTest	AVR LCD_Edit	AVR SHT11
AVR DCF77	AVR GraphTWI	AVR LCDIOS	AVR SleepTest
AVR Disp14	AVR Heap	AVR LCDmulti	AVR SmallMen

Below the list, there is a search field containing 'AVRpas' and buttons for 'Delete', 'Import', 'Load', and 'Exit'. The status bar at the bottom shows 'Errors: 0' and the time '19:38'.

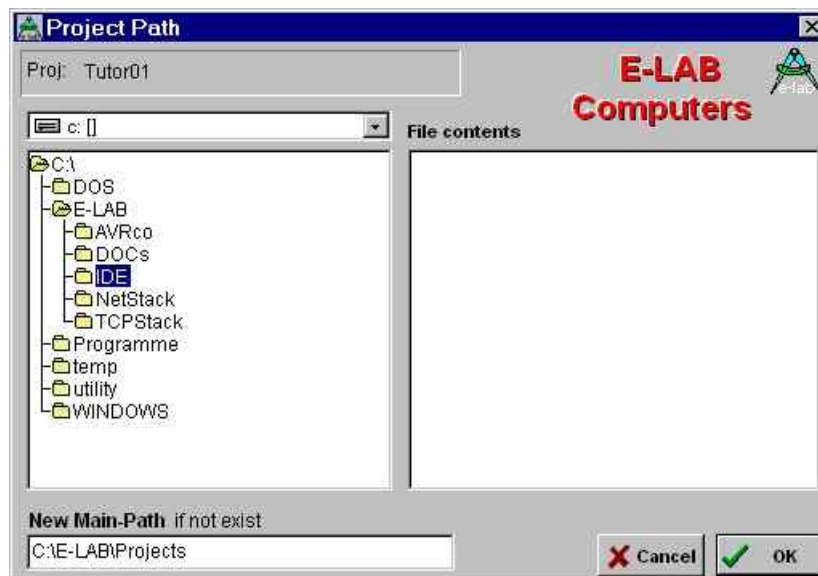


AVRco Tools

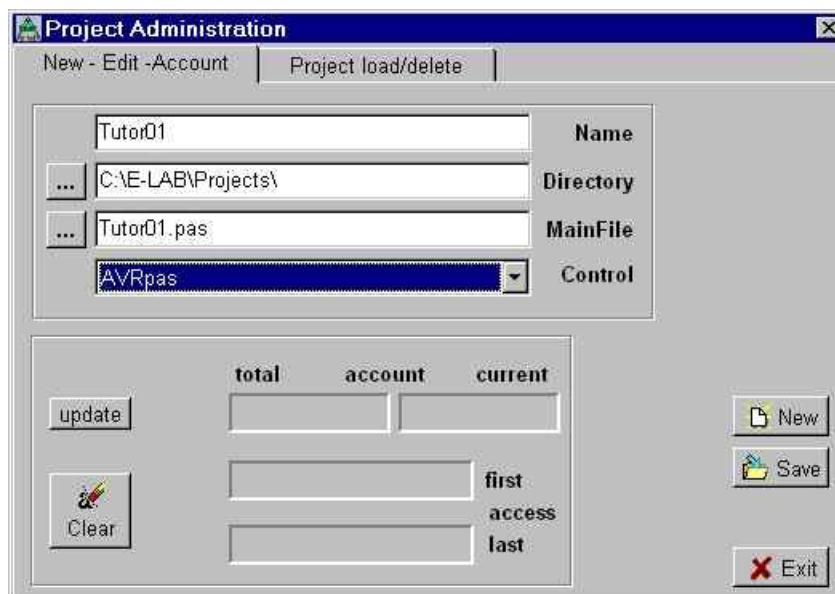
2.2.2 Erstellen Sie Ihr erstes Projekt

Klicken Sie im *Project Administration* Fenster die *New – Edit – Account* Klappe.
Im nächsten Fenster geben Sie

- *Tutor01* im Feld *Name* ein
- klicken Sie die drei Punkte vor dem *Directory* Feld: das *Project Path* Fenster erscheint
- ändern Sie den ***New Main-Path if not exist*** Eintrag wie auf dem folgenden Bild




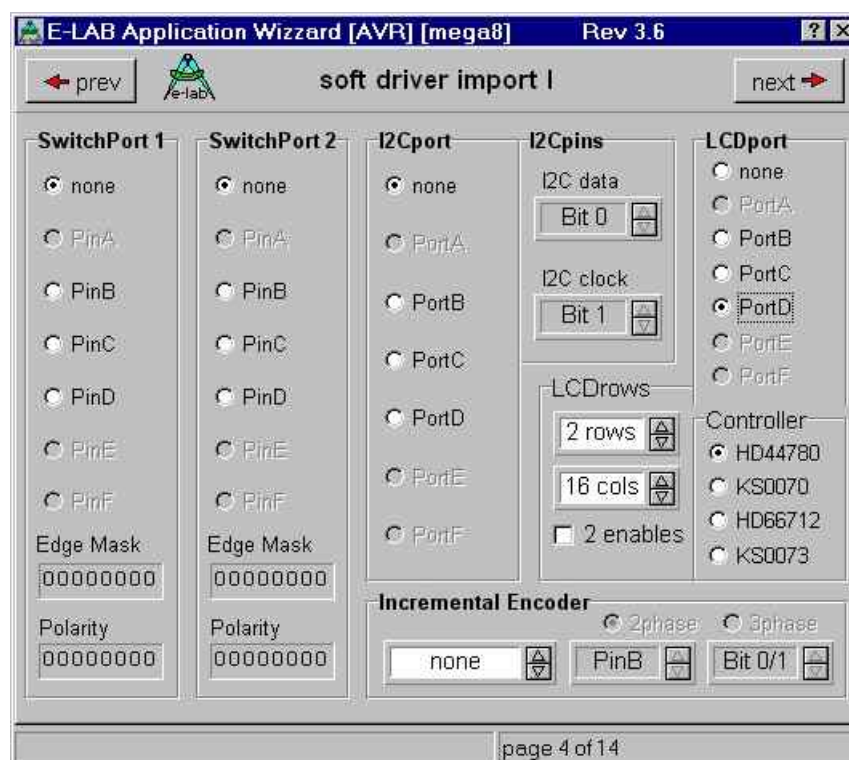
- klicken Sie *OK* und *Yes* in dem *Confirm* Fenster das erscheint
- geben Sie den *MainFile* Name ein und wählen Sie das *Control*



- klicken Sie *Save* und *Exit*

2.2.2.1 Erstellen Sie ein Programm Gerüst

- klicken Sie den "speed button" *Projekt* 
- wählen Sie (bei *Project load/delete*) das *Tutor01* Projekt und klicken Sie *load*
- es dauert einen Augenblick bis der *Application Wizard* geladen ist
- wählen Sie auf der 1. Seite *mega8* und eine *Frequency* von 8 MHz. Ändern Sie **keine** anderen Einstellungen
- klicken Sie wiederholt *next* bis Sie zur Seite *soft driver import* gelangen, die den *LCDport* enthält
- aktivieren Sie den *LCDport* an *PortD* und wählen Sie 2 rows und 16 columns (*cols*) (2 Zeilen mit 16 Spalten)

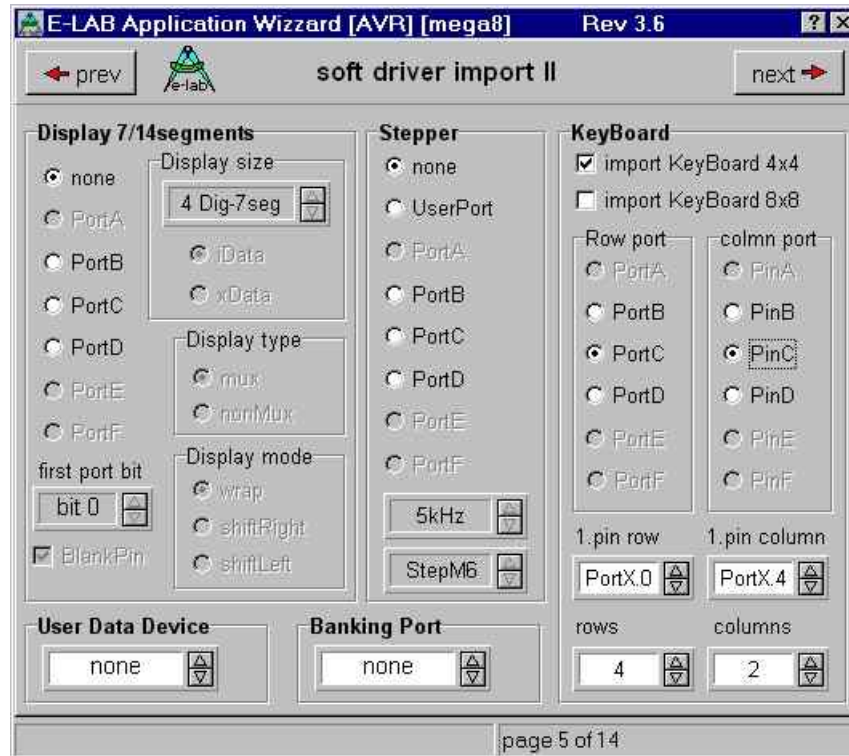


Bem.: die Seitenzahl (oben 4 of 14) kann anders sein. Das hängt von der Compiler Version ab.

- klicken Sie *next* bis zum *soft driver import* Fenster mit dem *KeyBoard* (siehe Abbildung auf der nächsten Seite)
- aktivieren Sie *import KeyBoard 4x4*
- wählen Sie *PortC* und *PinC* als *Row port* bzw. *column port* aus
- wählen Sie *PortX.0* als *1.pin row* bzw.. *PortX.4* als *1.pin column*
- stellen Sie 4 rows und 2 columns ein

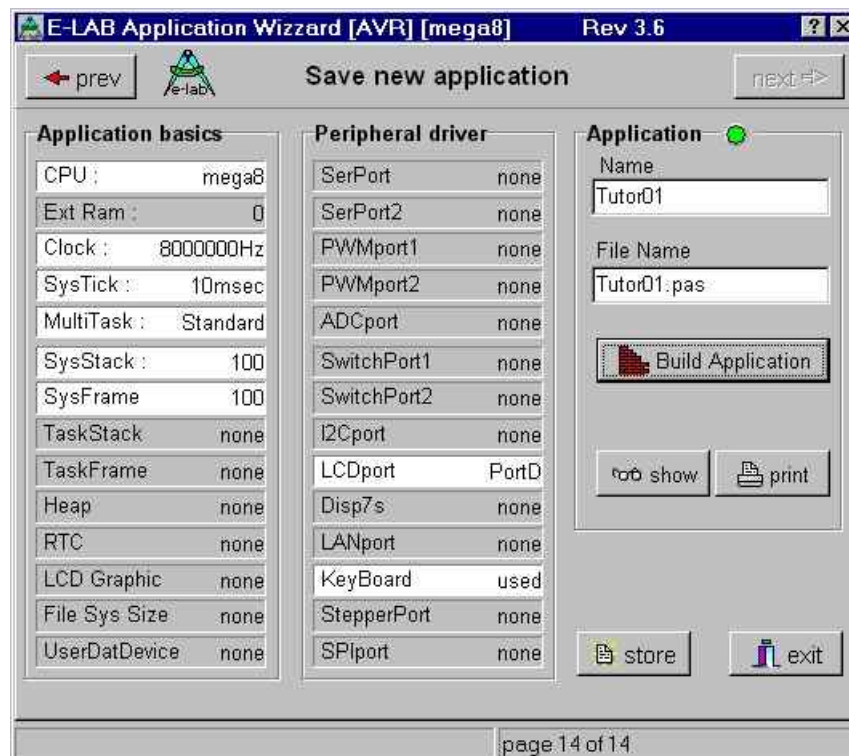


AVRco Tools



Bem.: die Seitenzahl (oben 4 of 14) kann anders sein. Das hängt von der Compiler Version ab.

- klicken Sie wiederholt *next* bis zum letzten Fenster



Bem.: die Seitenzahl (oben 4 of 14) kann anders sein. Das hängt von der Compiler Version ab.

klicken Sie *Build Application*, *store* und dann *exit*
die *Project Administration* wird geschlossen und das generierte Gerüst in den Editor geladen

Sie sollten nun folgendes Programm Gerüst sehen:

```
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

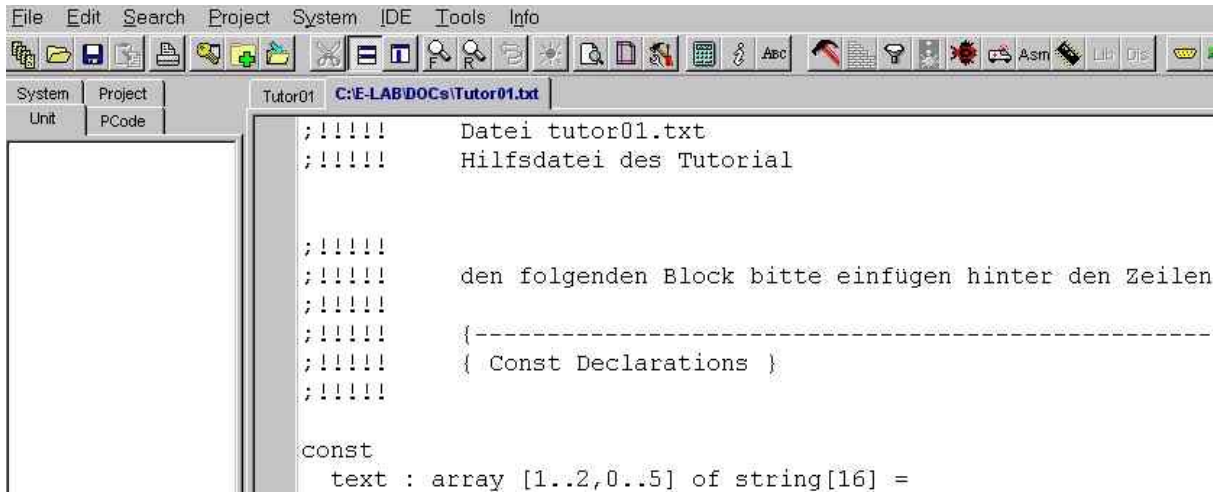
  endloop;
end Tutor01.
```



AVRco Tools

2.2.2.2 Geben Sie das Programm ein

- der Einfachheit halber gibt es eine .txt Datei, die das Programm enthält
- klicken Sie den *Open speed button* 
- ändern Sie den Dateityp von *PASCAL/ASM Source* nach *Text*
- suchen Sie die Datei *tutor01.txt* in *C:\E-Lab\DOCs* und öffnen Sie diese



```

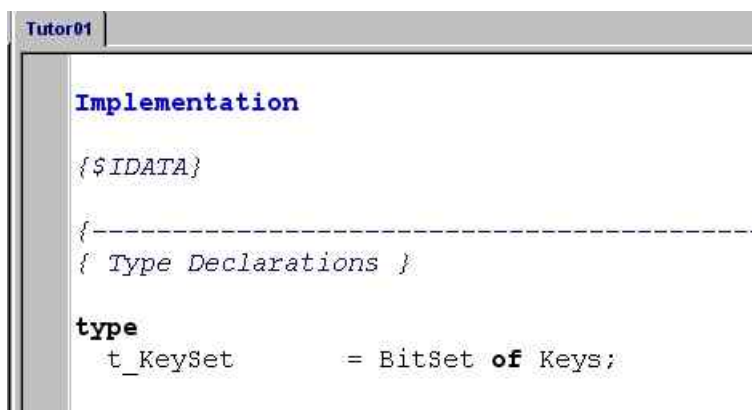
File Edit Search Project System IDE Tools Info
;!!!! Datei tutor01.txt
;!!!! Hilfsdatei des Tutorial

;!!!!
;!!!! den folgenden Block bitte einfügen hinter den Zeilen
;!!!!
;!!!! {-----
;!!!! { Const Declarations }
;!!!!

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

```

- die Textdatei wird in ein zweites Fenster geladen
- beachten Sie die Klappen *Tutor01* (das Programmgerüst) und *C:\E-Lab\DOCs\Tutor01.txt*
- markieren Sie den Block der mit *const* beginnt (bis zur Ende Markierung) und drücken Sie *Ctrl+c* bzw. *Strg+c* (copy)
- merken Sie sich die Stelle wohin der Block kopiert werden soll (die Information über dem Block)
- klicken Sie die *Tutor01* Klappe um das Programmgerüst zu öffnen
- positionieren Sie den Cursor in der Ziel Zeile (in der *Tutor01* Datei) und geben Sie *Ctrl+v* (paste) ein
- kehren Sie zur Text Datei zurück und kopieren Sie den 2. und 3. Block an die richtigen Stellen
- führen Sie einen **Rechts**-Klick auf die Klappe *C:\E-Lab\DOCs\Tutor01.txt* aus und wählen Sie *close file*
- vervollständigen Sie den *type* Abschnitt gemäß dem folgenden Bild (geben Sie die Zeile beginnend mit *t_KeySet* ein)
- Bem.: das "**of**" ist ein Schlüsselwort. Der Editor stellt Schlüsselworte **fettgedruckt** dar
- stellen Sie sicher, daß Sie die Zeile genau wie gezeigt eingeben und vergessen Sie nicht das Semikolon ";" (die Anzahl der Leerzeichen ist egal)




```

Tutor01
Implementation
{$IDATA}
{-----
{ Type Declarations }


type
  t_KeySet      = BitSet of Keys;

```

2.2.2.3 Compilieren und Assemblieren Sie das Programm (= "Make")

- klicken Sie den *make speed button*  um das Programm zu compilieren und assemblieren
- wenn Sie nach einem kurzen Augenblick einen "Beep" hören und die Sanduhr verschwindet wurde das Programm erfolgreich compiliert und assembliert. **Dann weiter mit dem nächsten Kapitel!**
- wenn Sie einen "Chord" hören und ein Fehlerfenster erscheint ist noch was falsch

wenn Sie Fehler bekommen haben:

- klicken Sie *OK* um das Fehler Fenster zu schließen
- versuchen Sie **nicht** einen Fehler zu ignorieren. Der Compiler/Assembler hat dann noch nicht die notwendigen Dateien erzeugt um weitermachen zu können
- sehen Sie sich die (erste) gelb unterlegte Zeile an. Der (erste) Fehler ist in dieser Zeile oder darüber.
Seien Sie sich bewußt, daß der Compiler **nicht** immer die **genaue** Position eines Fehlers finden kann. **Der Fehler liegt häufig eine oder mehrere Zeilen darüber!**
- versuchen Sie **nicht** mehrere Fehler gleichzeitig zu korrigieren! Es ist nicht unüblich, daß ein einziger Fehler mehrere Meldungen und/oder Folgefehler erzeugt
- es handelt sich häufig um Schreibfehler oder vergessene Semikolons
- versuchen Sie –Schritt für Schritt- die Fehler zu beseitigen und führen Sie zwischendurch immer wieder ein "make" aus
- wenn das keinen Erfolg bringt, drucken Sie die Pascal Datei  und die *tutor.txt* Datei aus und vergleichen Sie die Blöcke und deren Position.
- **Sie sollten unbedingt versuchen die Fehler selber zu finden! Auch wenn das eine Menge von Versuchen und einige Zeit beansprucht. Dies ist eine häufige Aufgabe bei der Software-Entwicklung und Sie werden diese Erfahrungen bei Ihren eigenen Projekten brauchen.**

wenn Alles nichts hilft:

drucken Sie den "*tutor01.pas*" Teil im Anhang aus und vergleichen Sie ihn mit Ihrem Programm. Achten Sie besonders auf die Blöcke, die Sie kopiert haben:

- sind diese Blöcke vollständig?
- sind sie an der richtigen Stelle?
- ist der "type" Block OK?


die letzte Rettung:

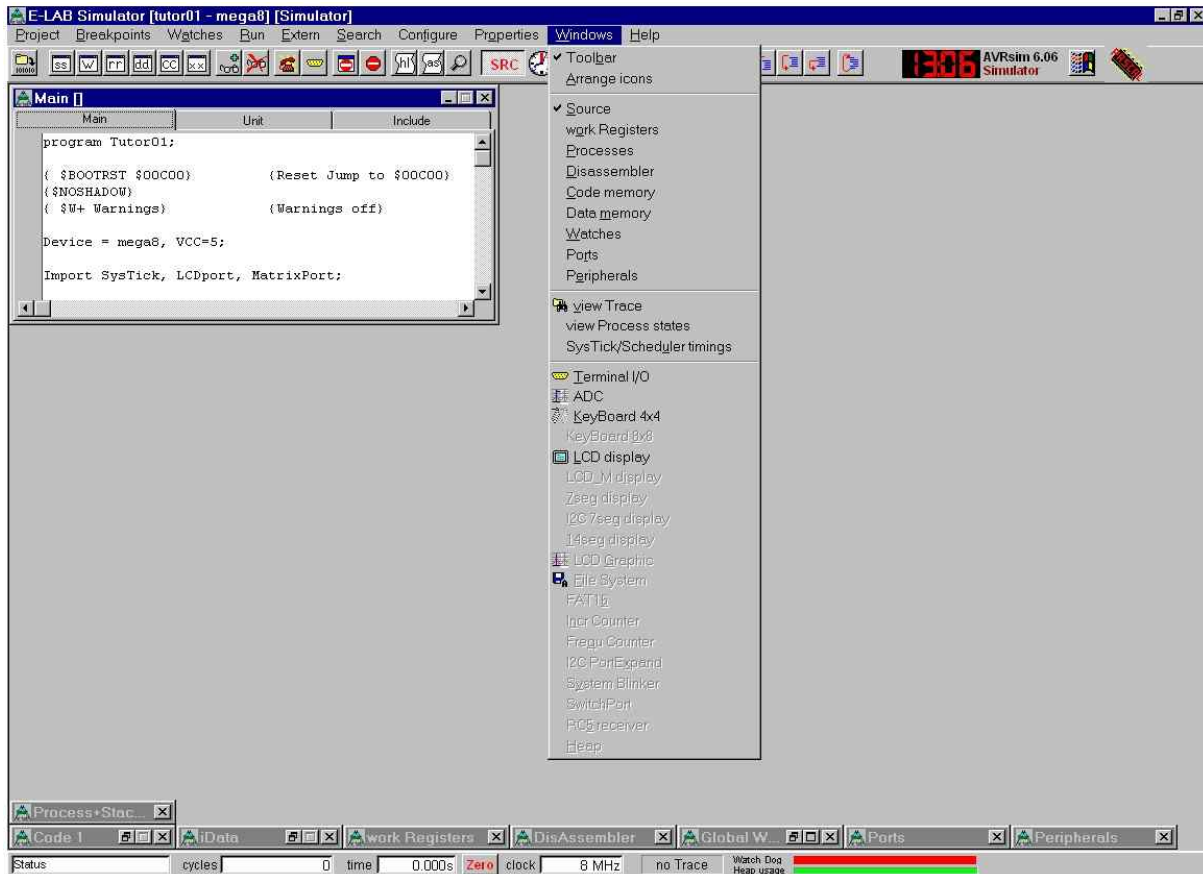
löschen Sie den Inhalt Ihrer gesamten Datei und kopieren Sie den entsprechenden Teil im Anhang mittels cut&paste ("kopieren" und "einfügen")



AVRco Tools

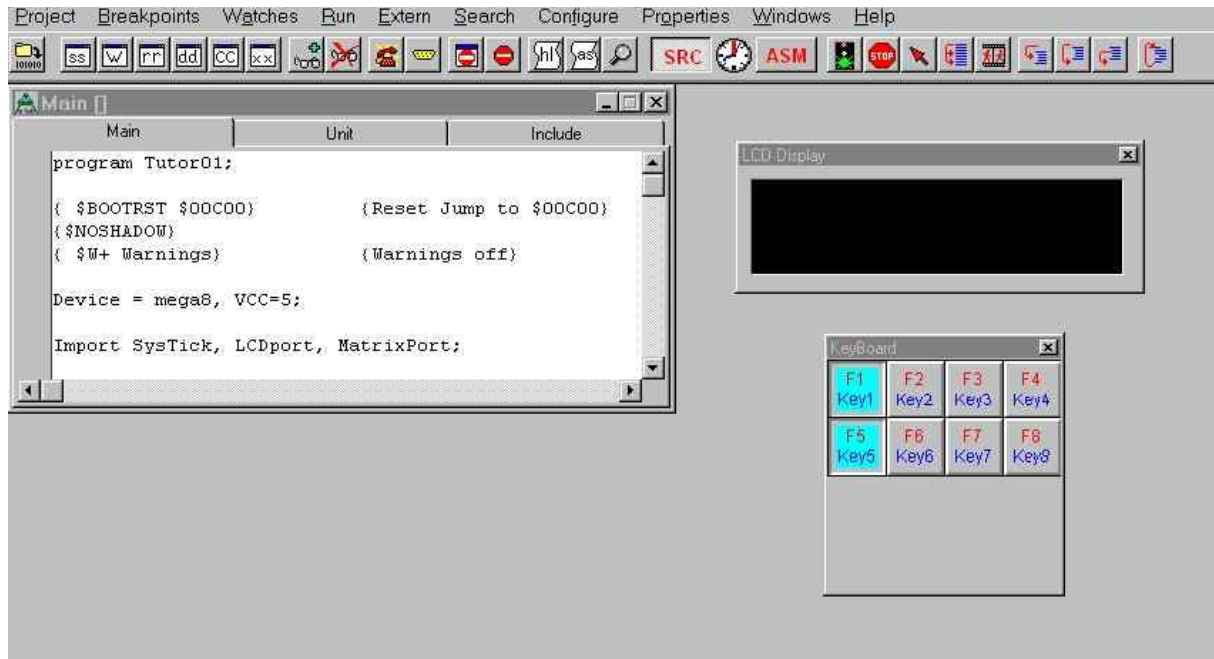
2.2.3 Testen Sie das Programm im Simulator


- starten Sie den Simulator mit dem Speed Button 
- der Simulator startet mit vielen offenen Fenstern. Weil wir (in diesem Beispiel) das Programm nicht "debuggen" (= Fehler suchen) wollen, brauchen wir diese nicht schließen Sie alle Fenster außer *Main []*, das die Source (=Quellcode) enthält, die Sie schon kennen.
- drücken Sie *Alt+w* um das *Windows* Menue zu öffnen
- Sie sollten folgenden Bildschirm sehen:

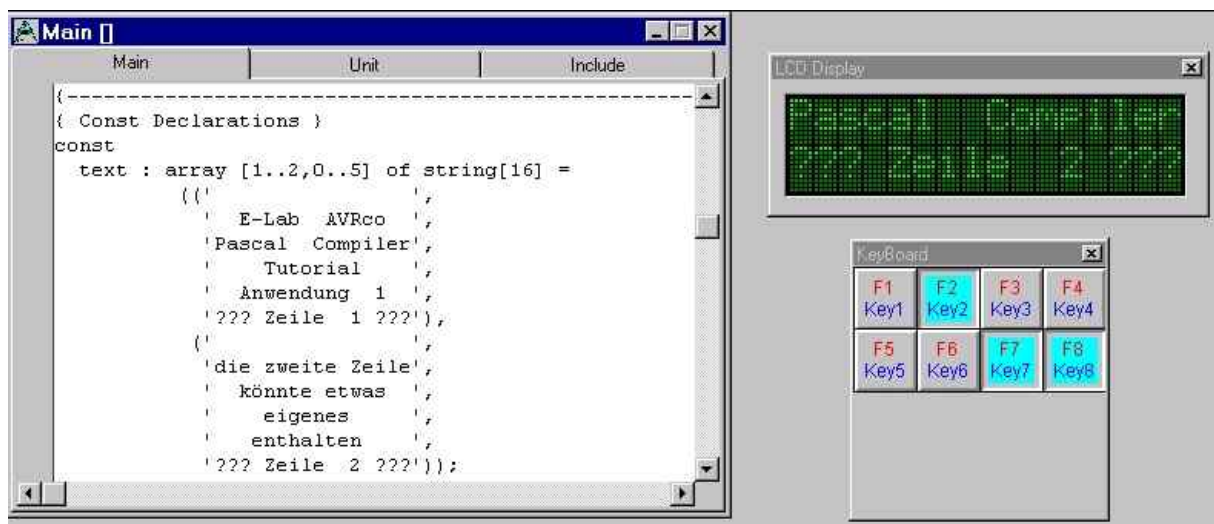


- wählen Sie das *KeyBoard 4x4*
- drücken Sie erneut *Alt+w* und wählen Sie das *LCD Display*
- positionieren Sie die Fenster gemäß dem nächsten Bild
- klicken Sie einige Tasten und sehen Sie, wie diese ein- und ausgeschaltet werden
- schalten Sie *F1/Key1* und *F5/Key5* ein, alle anderen aus

AVRco Tools



- drücken Sie die *run* Taste  und beobachten Sie die LCD Simulation
- schalten Sie die beiden Tasten (*Key1* and *Key5*) aus und testen Sie andere (zunächst immer nur eine aktive Taste pro Zeile!)
- sehen Sie was passiert, wenn mehr als eine Taste pro Zeile gedrückt wird
- vergrößern Sie das *Main []* Fenster und blättern Sie zu den *Const Declarations*
- spielen Sie mit den Tasten und sehen Sie sich den Zusammenhang zwischen den Tasten, dem LCD und der Source Datei an





AVRco Tools

2.2.4 Geben Sie eigene Texte ein

- schließen Sie den Simulator und Sie sind wieder im Editor
- blättern Sie zur Definition der 2. Zeile die mit

```
('          ;  
  'die zweite Zeile';          beginnt.
```

- beachten Sie:
 - jede Nachricht ist in Hochkommata eingeschlossen: 'xxxxxx'
 - die erste Zeile enthält 16 Leerzeichen und wird benutzt um die Zeile zu löschen
 - die folgenden 4 Nachrichten gehören zu Key5 ... Key8
 - die letzte Nachricht wird angezeigt wenn mehr als eine Taste gedrückt wird
 - die Länge aller Texte ist genau 16 Zeichen um sicherzustellen, daß ein langer Text durch einen kurzen komplett überschrieben wird
- ändern Sie **eine** Zeile und geben Sie eine eigene Nachricht ein
- **seien Sie noch vorsichtig ! Ändern Sie nicht zuviel gleichzeitig !**
- behalten Sie für diesen Test die feste Zeilenlänge von 16 bei
- übersetzen Sie das Programm ("make") wie in oben beschrieben und korrigieren Sie ggf. die Fehler
- starten Sie den Simulator und führen Sie das Programm aus (drücken Sie F9)
- registrieren Sie, daß der Simulator die Anzahl, Größe und Position der Fenster gespeichert hat
- kehren Sie zum Editor zurück und geben Sie einen absichtlichen Fehler ein ("vergessen" -löschen- Sie z.B. ein Hochkomma) und übersetzen Sie das Programm und zu sehen, wie der Compiler Fehlermeldungen anzeigt
- korrigieren Sie den Fehler und übersetzen Sie das Programm erneut
- mischen Sie kurze und längere Texte (als 16) und prüfen Sie das Verhalten des Programms
- wenn Sie Fehler bekommen, die Sie nicht selber korrigieren können, schauen Sie oben in *"Wenn Sie Fehler bekommen haben"*

2.3 Erstellen Sie eine weitere Anwendung – ein etwas tieferer Einblick

Dieses Kapitel soll Ihnen zeigen, wie Sie eine sehr einfache Anwendung für eine eigene reale Mikro-Controller Schaltung programmieren und testen können. Das Hauptziel ist zu demonstrieren wie neue Projekte erstellt und wichtige Funktionen der IDE benutzt werden. Weiterhin sollen Sie die Funktionen des Simulators genauer verstehen. Diese helfen Ihnen bei der Suche nach logischen Fehlern in Ihrem Programm.

Was soll die Anwendung tun?

4 LEDs sollen verschiedene Blinkmuster darstellen. Die Muster werden mit einer Taste ausgewählt. Die Taste und die LEDs sind an einem Mega8 Controller angeschlossen: die Taste an PortD-0, die LEDs an PortC-0 ... PortC-3.

Um den Aufwand gering zu halten, wird der interne Oszillator des Mega8 benutzt (1MHz). Das ist die Standard Einstellung eines neuen Controllers und Sie brauchen keine "Fuse Bytes" zu ändern. (1)

In diesem Kapitel wird die Hardware mit dem the AVRco Simulator simuliert.


In folgenden Kapiteln ist Alles beschrieben was Sie brauchen um eine echte Schaltung zu bauen und zu programmieren.

2.3.1 Erstellen Sie ein neues Projekt und erzeugen Sie ein Programmgerüst

sollten Sie unsicher bezüglich der Vorgehensweise sein, schauen Sie sich die entsprechenden Kapitel im ersten Teil und die dortigen Bilder an!

Starten Sie die PED32.exe. Beachten Sie, daß der Editor Ihr letztes Projekt und die offenen Dateien gespeichert hat und automatisch Tutor01 lädt.

es gibt verschiedene Möglichkeiten ein neues Projekt zu erstellen:

- z.B. der Speed Button  *project*
 - das Menu *File – New Project*
 - das Menu *Project – New Project*
 - Rechtsklick auf die Klappe *Tutor01* und Auswahl von *Load Project*
- und wahrscheinlich noch mehr ...

egal welche Art Sie benutzen – die *Project Administration* wird geladen (2)

geben Sie in der Klappe *New – Edit - Account* folgendes ein:

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

klicken Sie *save* und *exit* um den *Application Wizard* zu starten

Bem.:

(1): für mehr Informationen über die Fuse Bytes laden Sie sich das Mega8 Manual herunter (Links am Ende des Tutorials).

Seien Sie vorsichtig und experimentieren Sie nicht mit den Fuse Bytes! Es gibt einige Bits die den Controller sichern und ein neues Programmieren verhindern.

(2): benutzen Sie die immer die *Project Administration* wenn Sie ein Projekt laden / erstellen oder löschen wollen.

(3): für mehr Informationen über "Controls" und "Projects" siehe Kapitel "PED32"!



AVRco Tools

Der *Application Wizard* erzeugt ein Programmgerüst für Ihre Pascal Quelldatei. In einer ganzen Reihe von Seiten können Sie Ihr System und die gewünschten Eigenschaften und Optionen, wie Datentypen, On-Chip Optionen und zusätzlich benötigte Treiber einstellen. Weiterhin können Sie auch die benutzten I/O-Ports angeben.

Je mehr Informationen Sie hier angeben, umso vollständiger ist das erzeugte Gerüst. Der Wizard erzeugt die Befehle um die Treiber zu importieren, die notwendigen Definitionen, die Initialisierung der Ports und ein leeres Hauptprogramm.

Wichtig:

**benutzen Sie immer die *Project Administration* um eine neue Applikation zu erzeugen!
Sie dürfen nicht den *Application Wizard* starten um eine neue Applikation zu generieren!**

Sobald der AVRco die "main program" Datei nicht findet, startet er automatisch den Wizard. Ein manueller Start des Wizard ist für spezielle Zwecke vorgesehen und wird normal nicht gebraucht.

Es ist in diesem Tutorial bei Weitem nicht möglich alle Themen abzudecken. Deshalb werden nur die notwendigen Teile beschrieben. Alle Funktionen sind standardmäßig ausgeschaltet. Sie können sich somit auf die benötigten konzentrieren und diese gezielt einschalten.

Keine Sorge, wenn Sie was vergessen oder einen Fehler gemacht haben:

der Wizard ist "nur" eine Hilfe und Sie können die betreffenden Teile mit dem Editor auch später noch hinzufügen, löschen oder ändern.

wählen Sie den *mega8* und eine *Frequency* von *1 MHz*. Belassen Sie die anderen Einstellungen auf Standard (1)

klicken Sie next.

Die folgenden Fenster hängen von der Compiler Version ab: der AVRco wird ständig erweitert und die Anzahl der Seiten wächst somit. Derzeit ist das 2. Fenster "*System and Types Import*". Diese Einstellungen sind im *DocuCompiler* Manual erklärt.

Das 3. Fenster heißt "*OnChip driver import I*". Derzeit passen alle "OnChip driver" auf eine Seite.

Abhängig von der zukünftigen Entwicklung der Controller wird es vielleicht bald eine Seite "*OnChip driver import II*" geben. Diese Einstellungen sind im AVRco- and dem Controller Manual erklärt.

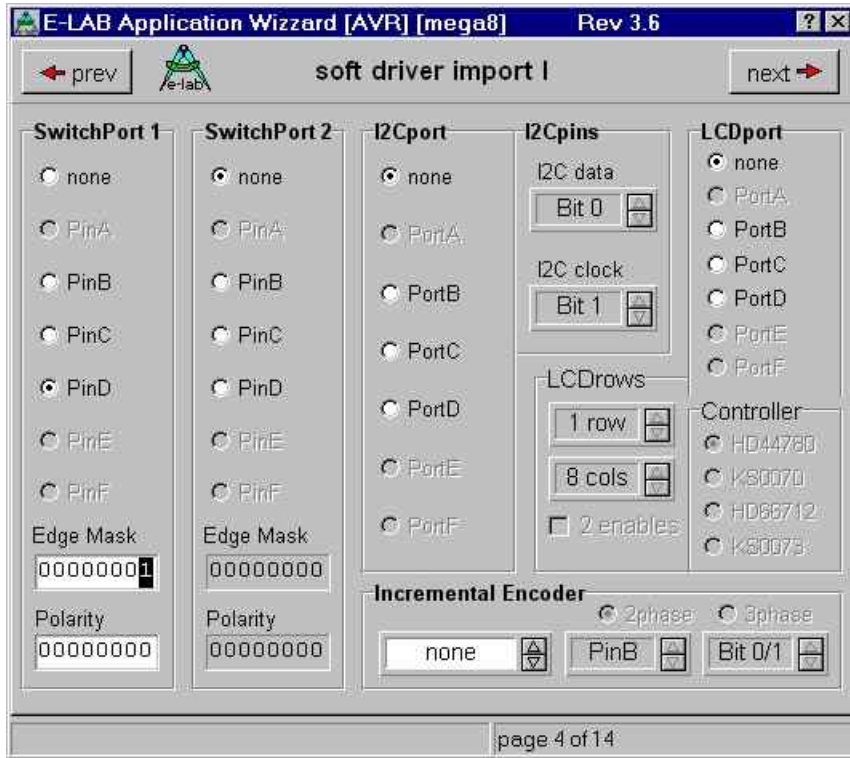
machen Sie die nächste Eingabe in der *soft driver import* Seite, die den *Switch Port 1* enthält:

wählen Sie PinD und ändern Sie die Edge Mask gemäß folgendem Bild

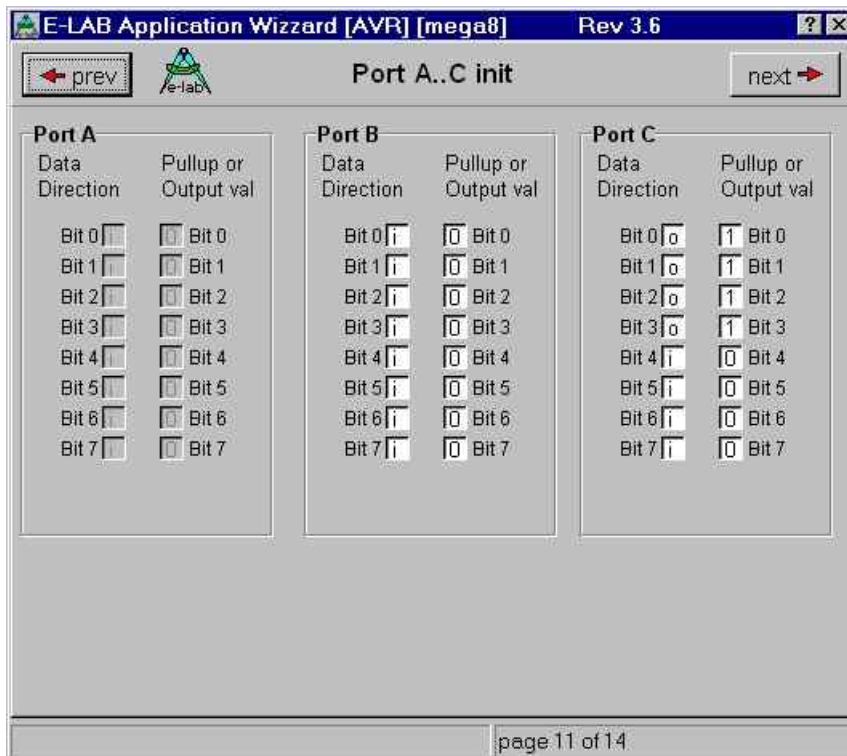
Bem.:

(1): abhängig von dem ausgewählten Controller Typ und den ausgewählten Optionen sind einige Funktionen nicht auszuwählen. Dies ist auch bei den folgenden Seiten so.

AVRco Tools



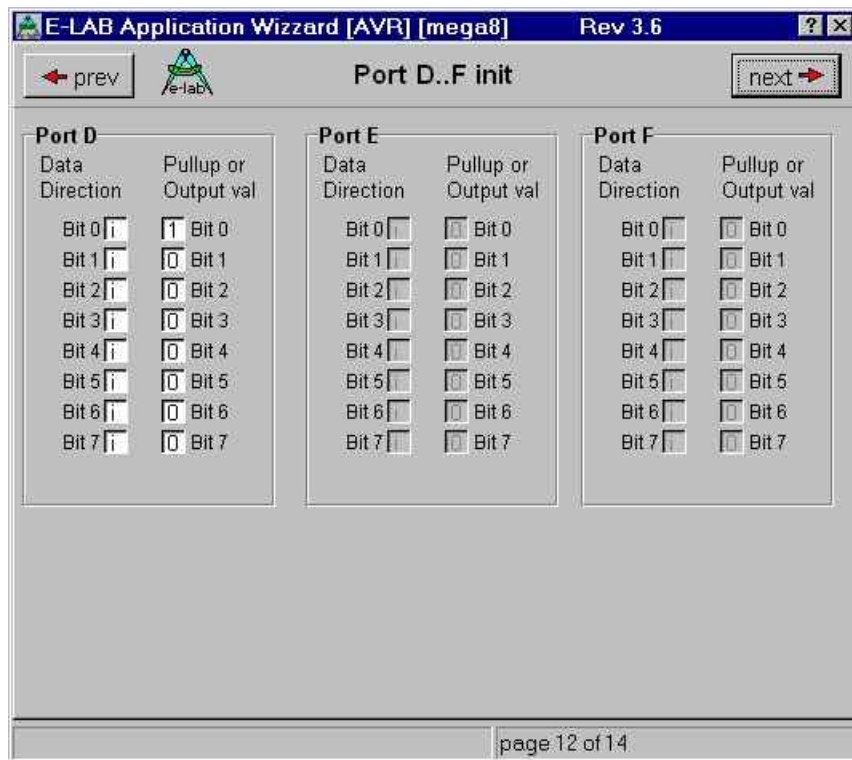
die nächsten Seiten heißen *special driver import*. Die Beschreibung dieser Treiber befinden sich im "DocuStdDriver" Manual
 klicken Sie wiederholt *next* bis zum *Port A..C Init* Fenster
 ändern Sie *Data Direction* und *Pullup or Output val* von *Bit 0* bis *Bit 3* von *Port C*





AVRco Tools

klicken Sie *next* zum *Port D..F init* Fenster und ändern Sie *Port D Bit 0* wie folgt



Die Bedeutung der *Data Direction* und *Pullup or Output val* Einträge steht im Controller Manual.

klicken Sie wiederholt *next* bis zur letzten Seite des Application Wizard
klicken Sie *Build Application*, *Store* und *Exit*

Sie sollten sich danach im Editor befinden und das Programmgerüst von Tutor02 sehen.

drücken Ctrl+F9 (=Strg+F9) um ein "make" (compilieren und assemblieren) des Programmgerüsts auszuführen. Dies sollte keine Fehlermeldungen erzeugen.

2.3.2 Geben Sie das Program ein und führen Sie ein "Make" aus

Geben Sie die Definitionen der Konstanten "pattern", der Variable "sel" und das Hauptprogramm zw. den Schlüsselworten *loop* und *endloop* gemäß folgendem Listing ein. Beachten Sie, daß der größte Teil des Programms vom *Application Wizard* erstellt wurde. Sie müssen nur die **blauen Zeilen** eingeben.

Geben Sie das Programm Schritt für Schritt ein!

Beginnen Sie mit dem "const" Block und führen Sie ein "make" durch. Korrigieren Sie etwaige Fehler bevor Sie fortfahren.

Machen Sie mit der "var" Definition weiter und lassen Sie den Compiler die Syntax überprüfen (drücken Sie Ctrl+F9).

Im nächsten Schritt geben Sie den **fettgedruckten blauen** Block hinter *loop* ein.

Die restlichen Zeilen können einzeln eingegeben (und überprüft) werden.

Denken Sie daran, daß der Compiler häufig **nicht** die genaue Position einer Fehlers bestimmen kann. Wenn Sie zu viel gleichzeitig ändern, kann es schwierig werden die Fehler zu finden.

Versuche Sie **nicht** mehrere Fehler gleichzeitig zu korrigieren (außer gleichartige an der gleichen Stelle, z.B. mehrere vergessene Kommata). Häufig können die Anweisungen hinter dem ersten Fehler vom Compiler nicht mehr richtig interpretiert werden.

Wenn Sie mehrere Fehler bekommen positioniert ein Klick auf die Fehlermeldung den Cursor auf die betreffende Zeile des Quellcode.

Wenn das "make" erfolgreich war probieren Sie mal folgendes aus:

- entfernen Sie die geschweifte Klammer am **Ende** von { Const Declarations }
- führen Sie ein "make" durch und sehen Sie wo der Compiler den Fehler vermutet
- vergessen Sie nicht, den Fehler wieder zu korrigieren

Sie sehen: je weniger Sie gleichzeitig ändern (oder hinzufügen) um so leichter sind Fehler zu finden.

```
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
```



AVRco Tools

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;

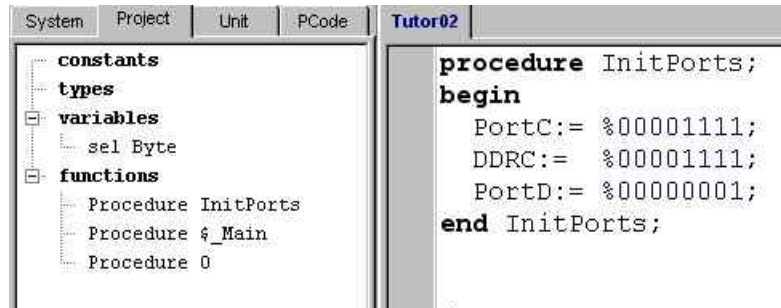
    PortC := not (Pattern[sel] shr 4);
    mDelay(250);
    PortC := not (Pattern[sel] and $0F);
    mDelay(250);

  endloop;

end Tutor02.
```

2.3.3 Einige schöne und nützliche Eigenschaften des Editors


- klicken Sie auf der linken Seite (im Editor) mal die *Project* Klappe und expandieren Sie alle Einträge



- Doppel-Klicken Sie *sel* und die Prozeduren. Der Cursor springt sofort zur entsprechenden Definition. Besonders bei großen Programmen ist diese Funktion sehr nützlich.
- positionieren Sie den Cursor auf ein Schlüsselwort (z.B. hinter das "c" im obigen "procedure InitPorts;")
- drücken Sie F1. Beachten Sie die Möglichkeit, die Beispiele über die Zwischenablage in Ihre Quelldatei zu kopieren.
- entfernen Sie die Formatierung einiger Zeilen, indem Sie die führenden Leerzeichen löschen und führen Sie ein "make" durch

z.B.:

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

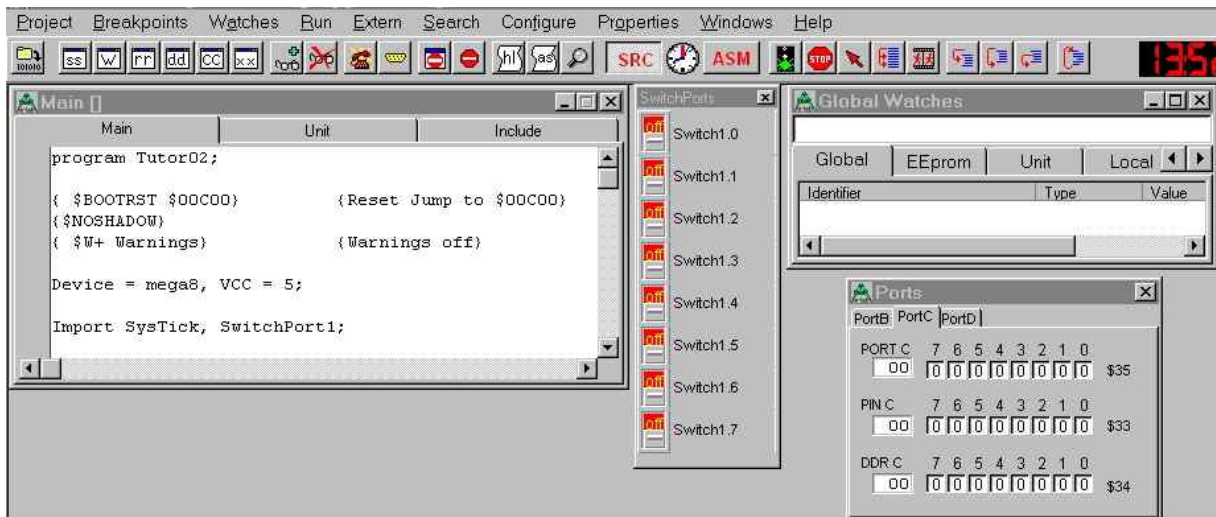
- klicken Sie den *Beautifler* Speed Button 
- positionieren Sie den Cursor hinter die "end Tutor02." - Zeile (alle Zeilen hinter dieser Anweisungen werden vom Compiler ignoriert. Hier ist ein sicherer Platz um mit dem Editor "herumzuspielen").
geben Sie die ersten Buchstaben einer Anweisung ein z.B. *md*
drücken Sie Ctrl + <leer> und dann <Eingabe> um die mDelay Prozedur zu übernehmen
- versuchen Sie einen einzelnen Buchstaben und Ctrl + <leer>
- drücken Sie Shift + Ctrl + p. Das ist ein Keyboard Makro und erzeugt ein leeres Gerüst für eine Prozedur.
- öffnen Sie das Menue *IDE – Edit keyboard macros* um alle vordefinierten Makros zu sehen
- entfernen Sie alle Testzeilen und führen Sie erneut ein "make" aus



AVRco Tools

2.3.4 Der Simulator – unverzichtbar auf dem Weg zum Erfolg

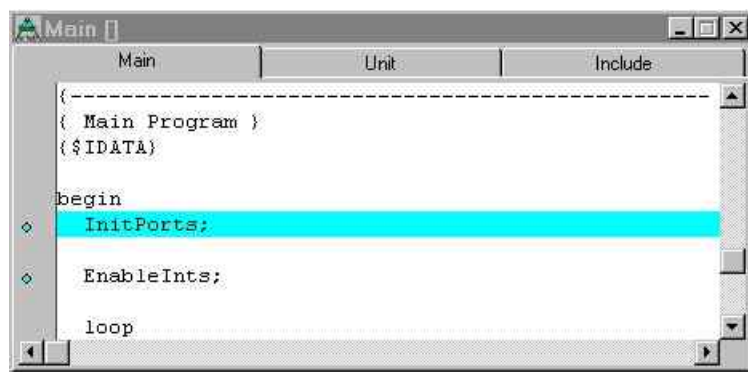
wenn das "make" erfolgreich war öffnen Sie den Simulator
zur besseren Übersicht schließen Sie alle Fenster außer *main[]*, *Global Watches* and *Ports*
schalten Sie den *SwitchPort* im *Windows* Menue ein
wählen Sie die *PortC* Klappe im *Ports* Fenster. Ordnen Sie die Fenster nach Position und Größe



drücken Sie F7 um einen Einzelschritt auszuführen (beachten Sie das Menue *Run* und die 3 Speed Buttons zur Einzelschritt-Ausführung)





Damit führen Sie die Initialisierung der Variablen und der importierten Treiber aus und der Cursor zeigt auf den ersten Befehl Ihres Programms: dem Aufruf der Prozedur *InitPorts*.

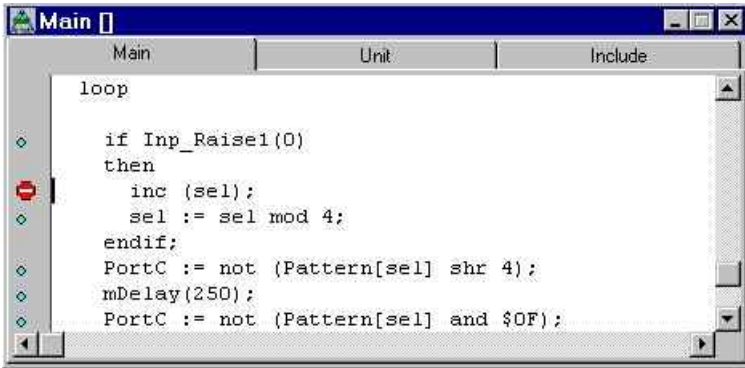


- drücken Sie F7 um diese Prozedur in Einzelschritten zu durchlaufen:
- führen Sie den nächsten Befehl `PortC:= %0001111;` aus (drücken Sie wieder F7)
- beobachten Sie die 1. Zeile im *Ports* Fenster (stellen Sie sicher, daß Sie Klappe *PortC* ausgewählt haben)

AVRco Tools



- führen Sie den nächsten Befehl aus und beobachten Sie, wie sich das Register *DDR C* ändert (die 3. Zeile)
- da der nächste Befehl ein Register von PortD ändert, klicken Sie die *PortD* Klappe bei *Ports*
- führen Sie den Befehl aus und drücken Sie 2 x F7. Sie befinden sich wieder im Hauptprogramm in der 2. Zeile: *EnableInts*;
- bevor Sie weitermachen spielen Sie mit den Schaltern im Fenster *SwitchPorts* und beobachten Sie die Anzeige von *Pin D*. Beachten Sie, daß ein offener Schalter als "1" und ein geschlossener als "0" dargestellt wird.
- falls Sie was verpaßt haben, können Sie mit *Reset processor* leicht "durchstarten" 
- versuchen Sie ein *Reset processor* und führen Sie dieses Mal die *InitPorts* Prozedur mit F8 aus (in einem einzigen Schritt)
- stellen Sie alle Schalter auf "off" und wählen Sie die *PortC* Klappe bevor Sie weitermachen
- drücken Sie wiederholt F7 und beobachten Sie die Programmschleife
- Erinnerung: wir wollen die 4 LEDs an *Port C0* .. *Port C3* anschließen. Diese Port Pins bleiben während der Programmschleife auf "1". Eine "1" bedeutet: die LED ist **aus** ! Halten Sie F7 gedrückt und beobachten Sie das *Ports* Fenster
- beachten Sie, daß die *if*-Abfrage niemals *true* wird und die Anweisungen innerhalb des Blocks nie ausgeführt werden
- die Variable *sel* wird benutzt um das Anzeigemuster auszuwählen. Welchen Wert hat diese? Suchen Sie nach einer Anweisung mit dieser Variable und doppel klicken Sie die Variable. Dies öffnet ein Fenster um die Variable zu editieren oder sie in die "WatchList" aufzunehmen. Klicken Sie *Add to WatchList* und schließen Sie das Fenster.
- beobachten Sie die Variable in *Global Watches* während Sie das Programm ausführen
- klicken Sie den blauen Kreis vor der Zeile *inc (sel)*; um einen *Breakpoint* zu setzen 



```



Main [
Main      Unit      Include
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);


```

- drücken Sie F9 um den Simulator frei laufen zu lassen. Beachten Sie das Uhrensymbol und die ausgegrauten Speed Buttons. Das Programm läuft jetzt bis Sie es anhalten oder ein Breakpoint erreicht wird.



AVRco Tools

- simulieren Sie einen Tastendruck indem Sie *Switch1.0* auf *on* und zurück auf *off* setzen (nicht zu schnell!)
- die if-Abfrage wird true, das Programm erreicht den Breakpoint und stopt
- fahren Sie mit einem Einzelschritt (F7) fort und beobachten Sie den Wert der Variable *sel*
- halten Sie F7 gedrückt und beobachten Sie *Port C*: das erste wechselnde Muster wird angezeigt
- wiederholen Sie die letzten Schritte (F9, Tastendruck, F7) und beobachten Sie die Muster
- halten Sie den Simulator an (wenn er läuft)  und entfernen Sie den Breakpoint indem Sie darauf klicken.
- drücken Sie Ctr+l+F9 um die *Animate* Funktion zu starten oder benutzen Sie  dafür
- spielen Sie mit den Schaltern. Beachten Sie, daß nur *Switch1.0* das Muster ändert. Der simulierte Controller läuft jetzt mit sehr geringer Geschwindigkeit um die Änderungen sichtbar zu machen. Da der *SwitchPort* entprellt ist, dauert es einige Zeit (einige Prozessor Takt Zyklen) um einen Schalter als *on* bzw. *off* zu erkennen.

- halten Sie den Simulator an (wenn er läuft) und stellen Sie den Cursor in die Zeile *inc (sel)*; im Source Fenster.
Drücken Sie F4 *Goto cursor pos* oder den Speed Button  und simulieren Sie einen Tastendruck

Der Simulator ist ein unverzichtbarer Teil des AVRco und nicht nur "nice-to-have".


Kein Programm funktioniert sofort wie gewünscht. Und wenn es das dennoch tut, seien Sie umso vorsichtiger – die Erfahrung zeigt, daß in so einem Fall die Fehler nur gut versteckt sind.

Die oben beschriebenen Techniken reichen aus um viele einfache Programme zu "debuggen". Wenn Sie anfangen komplexere Funktionen wie Interrupts oder MultiTasking zu benutzen gibt es noch viel mehr nützliche Funktionen im Simulator. Die meisten sind selbst-erklärend wenn Sie den betreffenden Hintergrund verstanden haben: z.B. die Benutzung des internen EEPROM. Andere Funktionen, wie ein Debuggen auf Assembler Ebene, sind selten notwendig.

Leider ist die eingebaute Hilfefunktion des Simulators noch nicht verfügbar.

Sollten Sie auf größere Probleme stoßen, folgen Sie den Anweisungen im Kapitel "*Weitere Informationen: Die AVRco Dokumentation*".

2.3.5 wie Sie mehr nützliche Informationen zu erhalten

- wählen Sie im Editor *Project – Project Informations* or klicken Sie  um das folgende Fenster zu öffnen



The screenshot shows the 'Project information [Tutor02]' window with the following data:

Flash used

Available:	Used:	Free:
\$02000 8kB	\$00228 1kB 7%	\$01DDA 7kB 93%

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

Register used [DATA]

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

Internal RAM used [DATA]

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

Internal EEPROM used [EEPROM]

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

Buttons: Symbols, Exit

Eine Einschränkung der Mikrocontroller sind die begrenzten Ressourcen.
Benutzen Sie die obige Funktion für eine Übersicht über die benutzten Ressourcen:

- Flash
 - der Programm Speicher
 - Sie sehen 8kB *available* (verfügbar). Das ist richtig für den Mega8 Controller. Aber denken Sie daran, daß die AVRco-Demo-Version auf 4kB beschränkt ist
 - achten Sie besonders auf den *Used* (benutzt) Eintrag: einschließlich aller System Programme benötigt Tutor02 etwa 1kB. Es sind also noch 3kB für Ihr Programm (und die im Flash gespeicherten Konstanten) frei. (1)

Bem.:

- (1): seien Sie sich besußt, daß der AVRco nur die notwendigen Teile einer System-Library importiert: z.B.: wenn Sie "float" importieren und nur Fließkomma-Multiplikationen benutzen wird das (Assembler-) Programm für Fließkomma-Division **nicht** importiert!
Testen Sie das mal wenn Sie das Tutorial beendet haben:
versuchen Sie $a := a * 1.5$; $a := a / 0.5$; und überprüfen Sie die Größe des Programms.
Starten Sie neu und versuchen Sie $a := a * 1.5$; $a := a * 2.0$; und schauen Sie sich die Größe an.
Öffnen Sie die .lst Dateien. Blättern Sie ans Ende und vergleichen Sie die Abschnitte "Imported Library Routines" in beiden Fällen.



AVRco Tools

- [DATA]
-die Prozessor Register (1)
-ist (derzeit) nicht wichtig. Nur für fortgeschrittene Programmierung.
- [IDATA]
-die Speichergröße (Ram) die für Variablen benutzt wird (Ihre eigenen und die vom AVRco intern benutzten). Hier sind noch 80% frei: also kein Grund zur Sorge
- [EEPROM]
-nur wichtig, wenn Sie das interne EEPROM benutzen
- wählen Sie (im Editor) *Info – Internet update* um die Version Ihres AVRco zu sehen.
Schauen Sie regelmäßig auf der E-LAB Home Page nach erweiterten Versionen und beachten Sie besonders die neuesten *DokuAddOnV-x*

öffnen Sie die Symbol Datei *C:\E-Lab\Projects\Tutor02.sym* um alle Symbole zu sehen, die der AVRco definiert hat. Vergleichen Sie diese mit dem Controller Manual und beachten Sie, daß Sie alle Prozessor Register mit deren Namen ansprechen können

lesen Sie die Kapitel "*types – BIT*" und "*System Library – BIT Processing*" im "*DocuCompiler*" Manual um einzelne Bits mit deren symbolischen Namen anzusprechen

z.B. um das *Carry Flag* (bit 0) des *Status Register (SREG)* als Variable "CFlag" anzusprechen, können Sie folgendes definieren

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

oder (2)

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

Bem.:

- (1): die Anzahl der available Prozessor Register hängt von den importierten Treibern ab. Benutzen Sie diese Informationen wenn Ihr Projekt beendet ist um die Programmgröße zu minimieren und die Ausführungsgeschwindigkeit zu erhöhen (siehe "*DocuCompiler*" Manual).
- (2): schauen Sie im Kapitel *Compiler Switches* des "*DocuCompiler*" Manuals für die Erklärung von `{$PDATA}` und `{$IDATA}`.
`{$PDATA}` weist den Compiler hier an, die internen Prozessor Register zu adressieren. Vergessen das `{$IDATA}` nicht, um für die folgenden Definitionen wieder den Ram Bereich des Controllers zu benutzen

2.4 Das interne EEPROM

das interne EEPROM stellt einen Permanent-Speicher dar, auf den folgendermaßen zugegriffen werden kann:

entweder zur Laufzeit

z.B. um Benutzer Einstellungen oder Eichwerte zu speichern, die beim nächsten Start wieder verfügbar sein sollen

oder schon während der Compilierung

z.B. um Menues, Tabellen usw. zu speichern

Obwohl alle Werte die schon bei der Compilierung bekannt sind im Flash gespeichert werden können, kann die Speicherung im EEPROM Platz im Flash für mehr Programmcode schaffen.

Es gibt verschiedene Möglichkeiten um mit dem AVRco auf das EEPROM zuzugreifen. Dafür gibt es ein eigenes Kapitel im "*DocuCompiler*" Manual! (1)

Wenn Sie in Ihrer Source EEPROM Daten spezifizieren (siehe {\$EEPROM} Compiler Switch) indem Sie die **StructConst** Anweisung benutzen, erstellt der AVRco eine zusätzliche Datei mit einer *.eep*-Erweiterung. (2)

Diese Datei muß, zusammen mit der Datei für den Flash Speicher (Erweiterung *.hex*), in den Controller übertragen werden.

Üblicherweise hat die Programmier Software dazu folgende Optionen (3)

- read / write / verify all (Flash und EEPROM)
- read / write / verify Flash
- read / write / verify EEPROM
- erase all (Flash und EEPROM)

Bem.:

- (1): ein Zugriff auf das EEPROM ist bei Weitem nicht so einfach wie ein Zugriff auf das Ram. Dazu ist eine spezielle Vorgehensweise unter Beachtung von Zeitvorgaben notwendig. Der AVRco übernimmt das: Sie können das EEPROM fast wie das Ram behandeln. Sie sollten nur beachten, daß der Zugriff auf das EEPROM um Einiges langsamer als ein Zugriff auf das Ram ist!
- (2): beachten Sie, daß Sie wirklich Daten definieren müssen, um diese Datei zu erstellen. Sie können auch Variablen im EEPROM Bereich definieren. Damit wird aber nur der Platz reserviert und es wird keine *.eep*-Datei erstellt. Schauen Sie sich die STRUCTCONST Anweisung an um Konstanten im EEPROM zu erstellen
- (3): manche Programmier Software benutzt standardmäßig andere Erweiterungen (z.B. *.e2p* für die EEPROM Datei) und Sie müssen diese Software extra so einstellen, daß sie nach *.EEP*-Dateien sucht und diese benutzt



AVRco Tools

2.5 Weitere Informationen: Die AVRco Dokumentation

Die Compiler Dokumentation ist in **C:\E-Lab\DOCs\DocuCompiler.pdf** enthalten.

Die Treiber sind in **C:\E-Lab\DOCs\DocuStdDriver.pdf** beschrieben.

Komplexere Treiber, die nur in der Profi Version des AVRco enthalten sind, finden Sie in **C:\E-Lab\DOCs\DocuProfiDriver.pdf**.

Die zusätzlichen Programme in diesem Manual **C:\E-LAB\DOCs\DocuTools.pdf**.

Die aktuellen Erweiterungen sind in **C:\E-Lab\DOCs\DocuAddOn** dokumentiert.

Das aktuelle *DocuAddOn.pdf* kann auch von der E-LAB Homepage geladen werden.

Diese ergänzen die jeweils aktuellen Handbücher.

2.5.1 wie findet man alle verfügbaren Informationen?

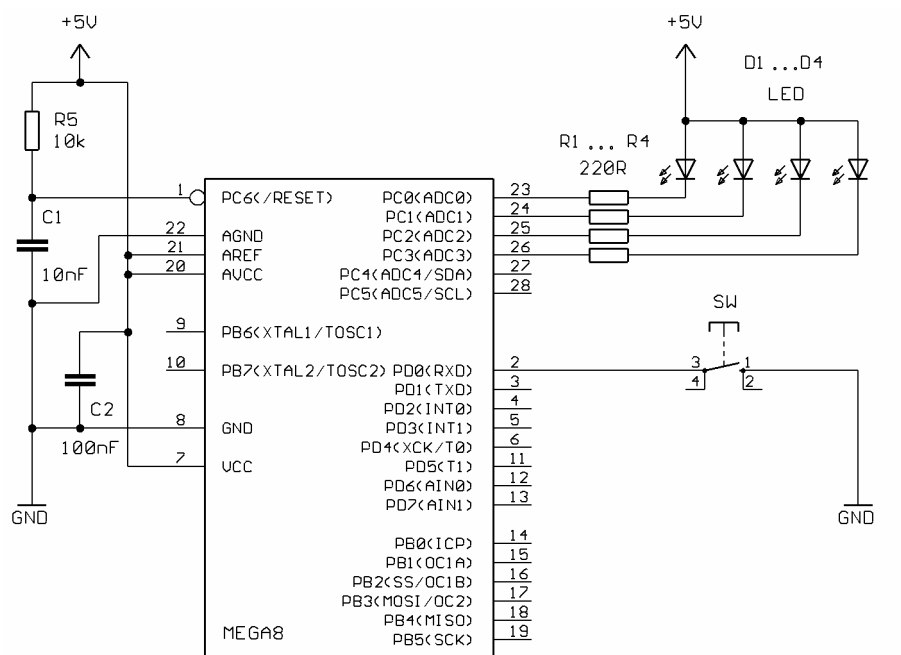
- die erste Quelle sind die AVRco Manuals
- neue Treiber werden häufig zusammen mit einem Beispiel veröffentlicht.
Die entsprechenden Dateien sind in *C:\E-Lab\AVRco\Demos* zu finden
- wenn Sie ein sehr spezielles Interesse oder Problem haben, kann es hilfreich sein, das ganze *C:\E-Lab* Verzeichnis (einschließlich den Unterverzeichnissen) nach typischen Schlüssel-worten zu durchsuchen
- wenn Sie keinen Erfolg haben, durchsuchen Sie das AVRco Forum auf der E-LAB Home Page nach den Schlüsselworten
- die letzte Rettung:
stellen Sie Ihre Frage im AVRco Forum

2.6 Welche Hardware brauchen Sie?

Wie oben erwähnt ist die Hardware sehr einfach.

Alles was Sie außen dem Controller brauchen sind ein Taster, 4 LEDs, 5 Widerstände und 2 Kondensatoren.

Der Stromverbrauch liegt weit unter 50mA und hängt vom ein/aus-Zustand der LEDs ab.



Beachten Sie, daß die LEDs mit gemeinsamer Anode angeschlossen sind. Eine "0" am Port schaltet die LED "ein".

Der Taster ist an GND angeschlossen. PD0 wird "0" wenn der Taster gedrückt wird.

LEDs und Taster sind "aktiv low".

Indem wir die entsprechenden *Pullup or Output val* Bits gesetzt haben, stellen wir sicher, daß die LEDs zunächst aus sind und ein interner "Pullup-Widerstand" an PD0 aktiviert ist.

Wenn Sie die Hardware selber bauen wollen ist es am Einfachsten ein Steckbrett zu benutzen.

Achten Sie besonders auf die Spannungsversorgung! Eine Spannung über 5.5V kann den Controller zerstören.

Sollten Sie nicht die notwendigen Hardware Kenntnisse haben, empfehle ich die Benutzung von kommerziellen Mini-Modulen und Spannungsversorgung (siehe Links am Ende des Tutorials).

Wenn Sie eine kommerzielle Schaltung benutzen editieren Sie die Zeile *ProcClock* und geben Sie die richtige Frequenz ein!



AVRco Tools

2.6.1 Einige zusätzliche Hinweise betreffend die Hardware

1. denken Sie daran: Controller mit einer "*internal oscillator*" Option (wie der Mega8) werden mit "internal oscillator at 1 MHz" programmierten Fuse Bytes ausgeliefert. Die XTAL 1/2 Pins sind dann "ganz normale" I/O Anschlüsse.
Um einen externen Oszillator zu benutzen müssen Sie einige Fuse Bits ändern. Ein Beispiel für einen Mega8 mit 8 bis 16 MHz finden Sie im Anhang.
Dringende Empfehlung: dokumentieren Sie immer die Standard- und die neuen Werte!

nochmal: seien Sie extrem vorsichtig mit den Fuse Bytes!

Bevor Sie irgendwelche Änderungen vornehmen: lesen und verstehen Sie die entsprechenden Kapitel im Controller Manual!

Achten Sie auch besonders darauf, wie die Bits von der Programmier-Software dargestellt werden! Die Controller interpretieren einen Wert von "1" als nicht programmiert und einen Wert von "0" als programmiert.

Manche Programmier-Software stellt dies Bits "invertiert" dar.

z.B. angekreuzt=programmiert="0" und nicht angekreuzt=nicht programmiert="1".

Das kann sehr verwirrend sein!

Die Fuse Bytes werden, wie das Flash und das EEPROM, mit einem Programmer und einer Programmier Software gesetzt.

2. Achtung: die neueren Controller mit 40 Pins und mehr (wie der Mega16) haben ein eingebautes JTAG Interface. Dieses Interface benutzt 4 Port Pins und ist bei Auslieferung enabled. Diese Port Pins können ohne Änderung der Fuses **nicht** als normale I/Os benutzt werden.
z.B.:
das JTAG des Mega16 benutzt PortC-2 bis PortC-5. Um diese Ports als normale I/O Pins zu benutzen müssen Sie das JTAGEN-Bit "disablen" (auf "1" setzen).
3. eine häufige Frage betrifft den Wert der externen Kondensatoren für einen externen Quarz.
Dieser Wert hängt vom Quarz selbst ab (und nicht vom Controller) und wird vom Hersteller des Quarzes spezifiziert.
Montieren Sie den Quarz und die Kondensatoren so nahe als möglich beim Controller.
Der Wert der Kondensatoren ist nicht sehr kritisch und 22pF paßt meistens.
4. um das Takt Signal mit einem Oszilloskop darzustellen benutzen Sie einen Tastkopf mit einer Abschwächung von 1:10. Nehmen Sie das Signal nur vom Pin XTAL2 ab. **Nicht** vom XTAL1 Pin (wegen der zusätzlichen Belastung durch den Tastkopf kann sonst die Schwingung abreißen)
5. ein andere Quelle der Verunsicherung ist oft das Verhalten von "Character LCDs" die nur an die Betriebsspannung angeschlossen sind. Solange diese Displays nicht initialisiert sind, zeigen die meisten ungerade Zeilen als schwarze Balken. Die geraden Zeilen bleiben hell.
z.B.:
-auf einem 2-Zeilen Display ist die 1. Zeile schwarz, die 2. hell
-auf einem 4-Zeilen Display sind die 1.&3. Zeile schwarz, die 2.&4. hell
Kein Grund zur Beunruhigung. Das LCD ist **nicht** defekt. Das ist "normal".
Aber Achtung bevor Sie ein Display kaufen: stellen Sie sicher, daß der AVRco den Controller-Typ des LCD unterstützt (der LCD-Controller ist ein –oder mehrere- SMD Chip, der auf der Rückseite des LCD montiert ist). Die unterstützten Controller sind auf der *LCD port* Seite des *Application Wizard* zu finden.
Leider sind einige der sogenannten "kompatiblen" Controller nicht "komaptibel genug" und verursachen Probleme.

Sollten Sie die schwarzen Balken nicht sehen, überprüfen Sie die Kontrastspannung! LCDs mit einem erweiterten Temperaturbereich (z.B. für die Automobilindustrie) brauchen eine negative Kontrastspannung (genauso wie graphische LCDs).

Achten Sie darauf, den richtigen Abschnitt im Datenblatt zu erwischen. Die meisten LCDs werden in verschiedenen Versionen (für verschiedene Temperaturbereiche) hergestellt und benutzen verschiedene Spannungsbereiche für die Kontrast-Einstellung.

2.7 Was sie brauchen um Ihre Hardware zu programmieren

der AVRco erzeugt eine Programm Datei (z.B. ...\\Projects\\TutorXX.**hex**). Diese Datei muß in den internen Flash Speicher Ihres Controllers übertragen werden.

Evtl. erzeugt er auch eine EEPROM Datei (z.B. ...\\Projects\\TutorXX.**eep**). Dies Datei muß in den internen EEPROM Speicher Ihres Controllers übertragen werden.

Bislang stehen diese Dateien auf Ihrer Festplatte. Um diese in Ihre Hardware zu laden brauchen Sie

1. irgendeine Verbindung zw. Ihrem PC und Ihrer Controller Hardware
2. irgendeine Software, welche die Datei(en) übertragen und überprüfen (zurück lesen) kann

Die Controller bieten verschiedene Möglichkeiten die internen Speicherbereiche zu schreiben, zu lesen oder zu löschen.

Diese Kapitel beschreibt die verbreitetste Möglichkeit, das SPI (Serial Peripheral Interface).

Die anderen Möglichkeiten sind

- der "Parallel Programming Mode" der ziemlich kompliziert ist und
- der "JTAG Mode" der nur bei Controllern mit ≥ 40 Pins verfügbar ist und eine Menge erweitert Funktionen hat (wie "OnChip Debugging")

SPI ist eine synchrone Schnittstelle, die den Datentransfer von/zu peripheren Geräten erlaubt. Während einem Reset (solange der Reset Pin auf "0" gehalten wird) geht das SPI in einen besonderen "Serial Programming Mode", der den Zugriff auf die internen Speicher erlaubt.

Der *Serial Programming Mode* benutzt die Pins SCK, MOSI and MISO zur Datenübertragung. (1)
Diese Pins (+ Reset) müssen vom PC gesteuert werden.

Auf der PC Seite gibt es derzeit 3 Typen von Schnittstellen um so etwas anzuschließen: einen parallelen Port, einen seriellen Port (= V24 = RS232) oder eine USB Schnittstelle.

Damit es keine Verwirrung gibt:

- die Programmierung erfolgt immer seriell (auch mit dem Paralle Port!)
- der "Parallel Programming Mode" hat nichts mit dem parallelen PC Port zu tun (sondern dabei ist das "program" und "verify" parallel möglich)

Bem.:

- (1) da gibt es auch Ausnahmen - wie beim Mega128. Schauen Sie bitte ins entsprechende Controller Manual!



AVRco Tools

Um den Controller an den PC zu hängen brauchen Sie eine Programmier Hardware (einen sog. ICP - In Circuit Programmer oder ISP – In System Programmer).

Es gibt alle Arten von derartigen Programmern:
mit seriell/parallel/USB Anschluß, professionelle und Selbstbau Versionen.

Und Sie brauchen eine Programmier Software die zu Ihrem Programmier paßt.
Auch da gibt es kommerzielle und frei verfügbare Versionen.

Weil die Selbstbau-Programmer (normalerweise zusammen mit einer freien Software veröffentlicht) sehr oft Probleme machen empfehle ich die Benutzung einer kommerziellen Version.

Links zu Selbstbau-Programmieren sind am Ende des Tutorial zu finden.

Weiterhin bieten alle Programmer / jede Programmier Software die Möglichkeit auf die Configuration Bytes (Fuses) der Controller zuzugreifen.

E-LAB bietet verschiedene Typen mit seriellem (V24) und USB Anschluß an und der AVRco beinhaltet eine Programmier-Software für diese Programmer. Die Software ist Bestandteil der Entwicklungsumgebung. Somit können Sie einen Speed Button benutzen um Ihre Dateien in die Hardware zu übertragen.

Das ist der sicherste und komfortabelste Weg Ihre Hardware zu programmieren.

Die kommerziellen Versionen des AVRco (Standart und Profi) beinhalten einen Programmer.

Die Standard Version beinhaltet einen V24/RS232 Typ "ISP-V24" und die Profi Version einen USB Typ "ISP-USB" der auch als JTAG-ICE Debugger dient.

Alle E-LAB Programmer unterstützen sowohl JTAG als auch SPI Programmierung.

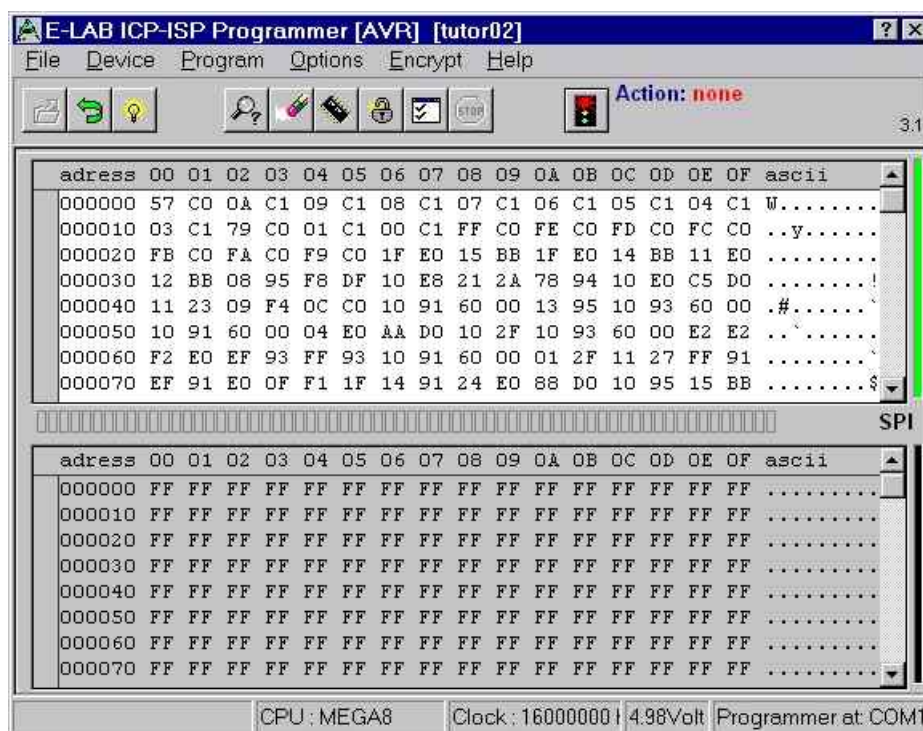


2.8 Einsatz des E-LAB Programmers (serieller Programmierer, SPI Mode)

verbinden Sie den Programmierer mit Ihrem PC und Ihrer Hardware und schalten Sie die Spannung ein. Beachten Sie das Programmiers-Manual für die verschiedenen Möglichkeiten der Spannungsversorgung.

Compilieren Sie Ihr Projekt mit der gewünschten Frequenz und Takt Quelle. (1)

Klicken Sie den *Prommer Speed* Button  in der IDE um die Programmier Software *AVRprog.exe* zu starten:



Alle notwendigen Informationen werden automatisch von der IDE zum AVRProg übertragen:

- der Name der Flash Datei
- der Name der EEPROM Datei (falls verfügbar)
- der Controller Typ
- die Takt Frequenz

und die Datei(en) werden geladen.

Bem.:

- (1): es ist **unerheblich** daß neuere Controller mit aktiviertem internem 1MHz Takt ausgeliefert werden. Im ersten Schritt werden die Fuse Bytes programmiert und die richtige Takt Quelle ausgewählt. Alle weitere Programmierung erfolgt dann mit dieser Einstellung für den Takt.



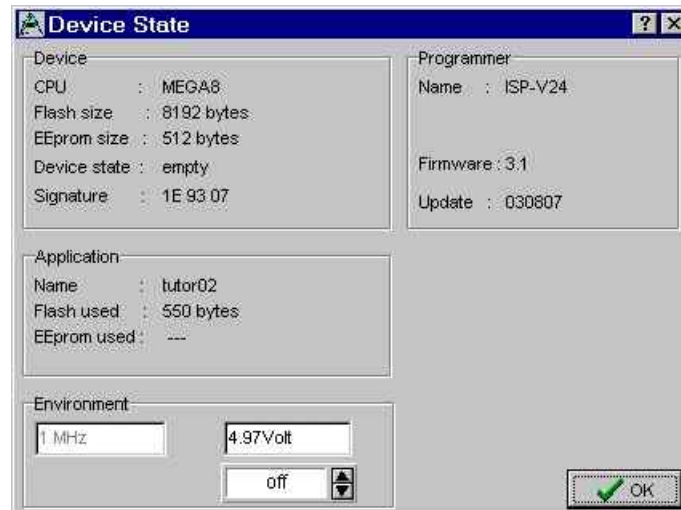
AVRco Tools

Beachten Sie die "*Programmer at COM1*" Anzeige in der unteren rechten Ecke. Diese zeigt an, daß der Programmer erfolgreich erkannt wurde.

Klicken Sie den *Device Check Speed* Button



Bei einem neuen Mega8 sollten Sie ein ähnliches Fenster erhalten:



WICHTIG:

wenn Sie den AVRprog zum ersten Mal mit einem neuen Projekt starten MÜSSEN Sie zunächst die *Programmer Options* anwählen !!! Dies müssen Sie normalerweise nur ein Mal tun (weil der AVRprog Ihre Auswahl für jedes Projekt speichert). Beachten Sie die Anweisungen auf den folgenden Seiten!

DIES MÜSSEN SIE (einmal) FÜR JEDES PROJEKT TUN!

Der Standard des AVRprog ist "program Flash, EEProm and Fuse Bytes". Der Standard für alle Fuse Bits ist inaktiv (binär "1").

z.B.:

ohne korrekte Auswahl wird der Mega8 auf "External Crystal / Ceramic Resonator" (CKSEL 3..0 = "1111") gesetzt.

Um auf den Mega8 wieder zugreifen zu können und diese Einstellung ggf. zu ändern, muß ein externer Takt angeschlossen sein.

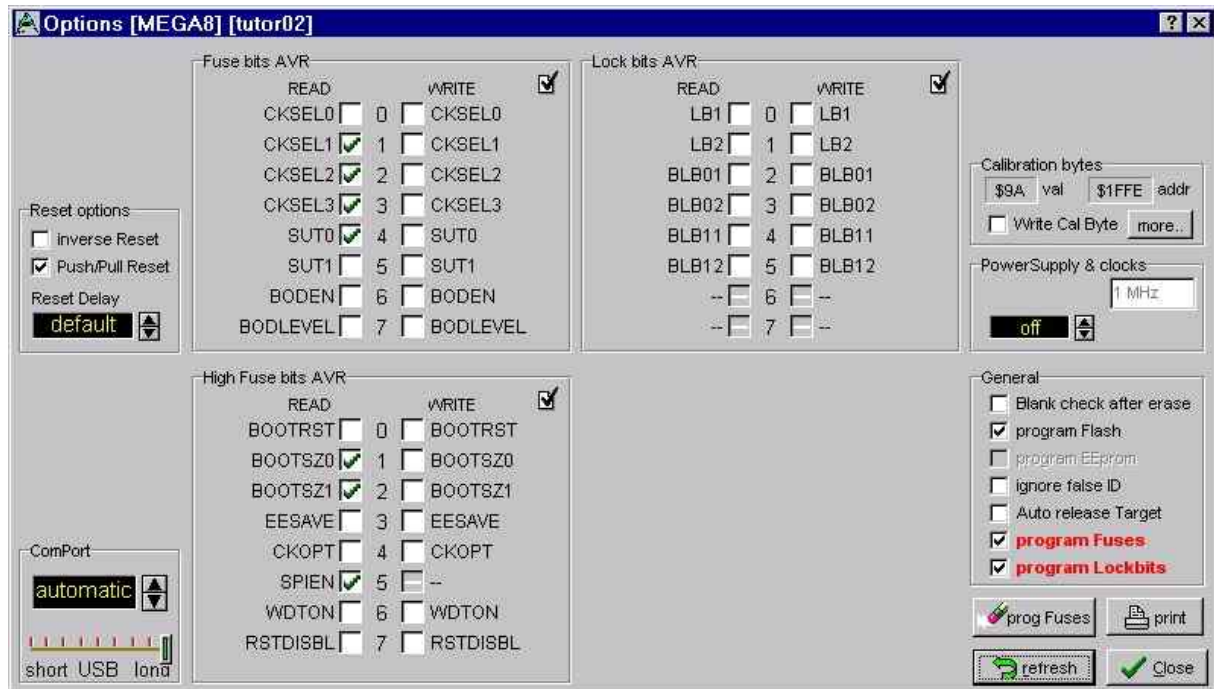
2.8.1 Das Einstellen der "Programmer Options" für ein neues Projekt

Klicken Sie "Options" – "Programmer Options"

da der Programmer die Fuse Bits noch nicht gelesen hat, ist kein Bit angekreuzt

klicken Sie "Refresh" um die Bits vom Controller zu lesen (beachten Sie den ControllerTyp in der Kopfzeile)

Sie sehen dann den aktuellen Zustand der Fuses (z.B. unten: die Werkseinstellung)



- wenn Sie eine Fehlermeldung wie die folgende bekommen



sind die Fuse Bits (schon) falsch gesetzt und Sie müssen eine korrekte Taktquelle anschließen um den Chip neu zu programmieren. Nur gibt es leider keinen Hinweis welcher Art eine korrekte Taktquelle ist

- wenn Sie den Controller mit dem AVRprog "zerschossen" haben, sind sehr wahrscheinlich alle Fuses disabled. Um auf einen Mega8 zuzugreifen muß ein externer Quarz oder ein Resonator angeschlossen werden ("CKSEL3..0" sind "1111"). Bei anderen Controllern schauen Sie ins Manual was "alle Fuses=1" bedeutet.

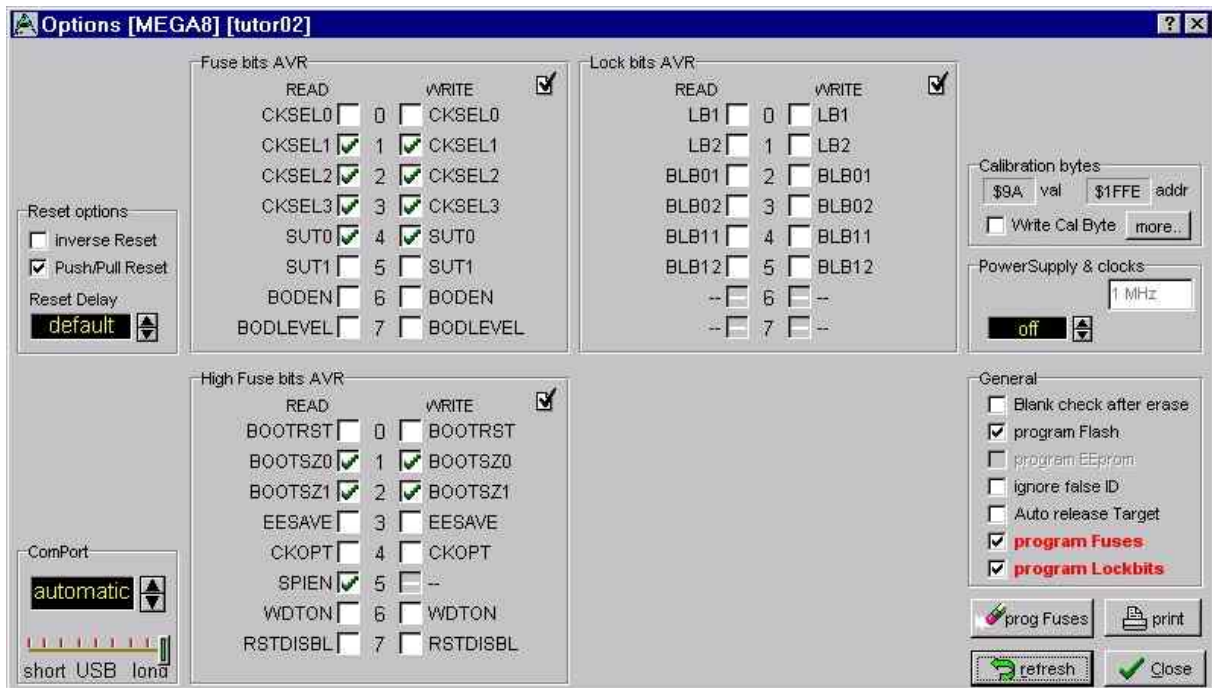


AVRco Tools

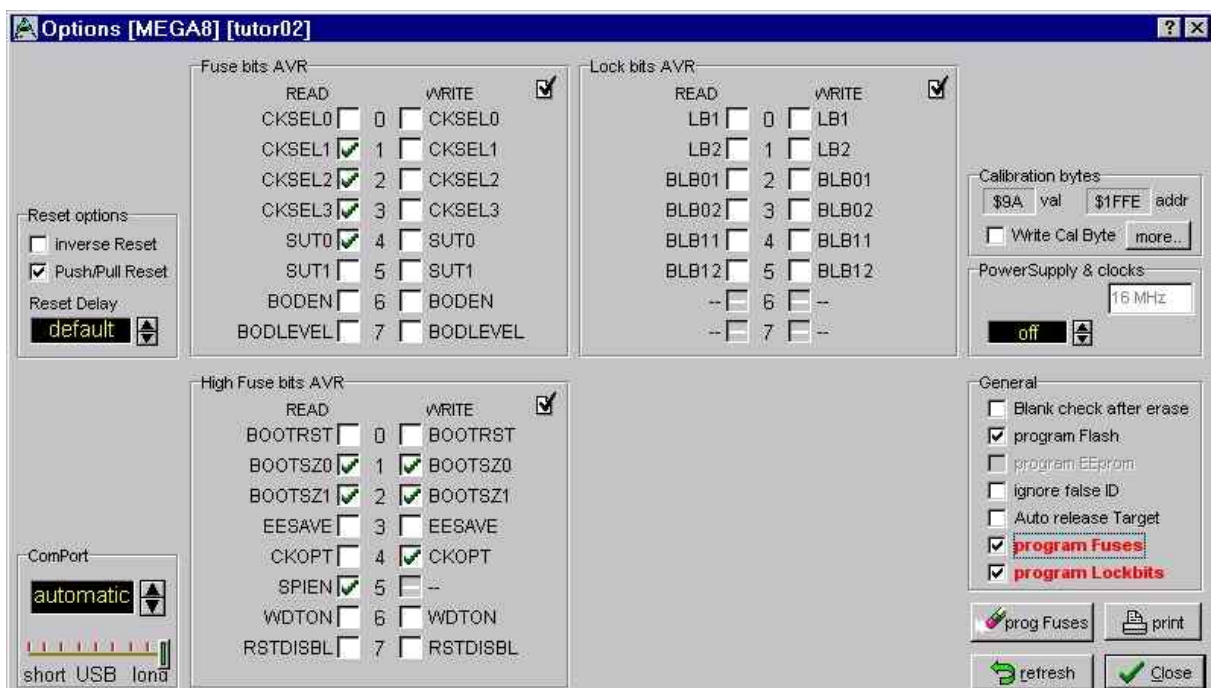
- beachten Sie, daß der JTAG Programmier Modus **keinen** Takt am Controller voraussetzt
- wenn Sie keine Fehlermeldung bekommen haben, wählen Sie die Bits, die geschrieben werden sollen: **Fuse bits WRITE**.

Beispiele:

für die Standardeinstellung (interner Oszillator mit 1 MHz) kreuzen Sie die gleichen Bits an



oder, wenn Sie einen externen Takt von 8 bis 16 MHz benutzen (siehe Anhang) stellen Sie die Bits wie folgt ein:



AVRco Tools

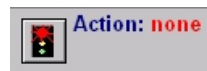


zum "Download" klicken Sie den *Program Speed* Button des AVRprog

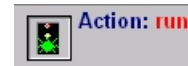


nach dem Programmieren bleibt Ihre Hardware gesperrt. Das heißt, der *Reset* Ihres Controllers bleibt aktiv (low).

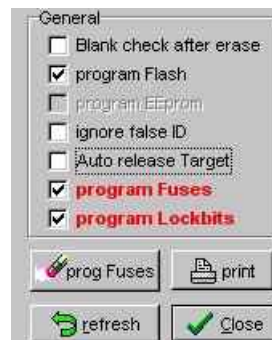
um Ihr Programm zu starten klicken Sie die Ampel



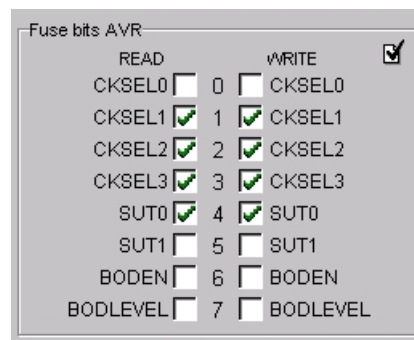
der *Reset* Pin wird freigegeben und Ihre Anwendung startet



wenn Sie Ihre Anwendung nach dem Laden automatisch starten wollen aktivieren Sie in den "Programmer Options" die Schaltfläche "*Auto release Target*"

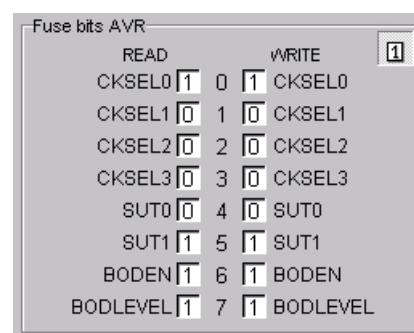


- beachten Sie, daß –abhängig vom Projekt und dem Controller- einige Optionen nicht verfügbar sind ("ausgegraut" sind).
z.B.:
im o.a. Beispiel ist "program EEPROM" nicht möglich, da keine EEPROM Datei erstellt wurde.
- beachten Sie, daß der AVRprog die logischen Zustände der Bits darstellt:



im obigen Beispiel: CKSEL0 ist nicht aktiviert, CKSEL1 ist aktiviert

Sollten Sie verwirrt sein oder diese Bits einfach nur mit dem Controller Manual vergleichen wollen, klicken Sie das Haken-Symbol oben rechts. Dann werden alle Bits als **binäre** Werte dargestellt.





AVRco Tools

2.9 Nur ganz kurz: Multitasking

Der folgende Absatz ist ein Auszug des AVRco Manuals:

Bei einer sog. Embedded Applikation (Single-Chip Anwendung) stellt sich sehr häufig das Problem, dass eigentlich mehrere Aufgaben gleichzeitig erfüllt werden sollten. Z.B. sollten die Zeichen einer seriellen Schnittstelle abgeholt, geprüft und evtl. von Hex zu einem Integer konvertiert werden. Gleichzeitig sind Ports mit Endschaltern zu Überwachen oder eine Leuchtdiode soll blinken. Zusätzlich ist ein Messwert von einem Poti zu erfassen und dieser Wert als Stellgröße einem Regler zu übergeben. Der Regler wiederum soll in einem festen Zeitraster eine Ausgangsgröße errechnen.

Alle diese Vorgaben stellen den Programmierer vor das Problem, möglichst alles gleichzeitig zu erledigen. Der Programmierer ist also in der schwierigen Lage, mehrere Vorgänge gleichzeitig überwachen zu müssen, wobei er damit rechnen muss, dass alle Funktionen **gleichzeitig und unabhängig** voneinander laufen müssen.

Mit einfachen Zeitschleifen etc. ist dieses Problem nicht mehr zu lösen oder nur noch mit Tricks, die das ganze Programm schwerfällig und unflexibel machen.

Man braucht also eine Lösung, die es ermöglicht, bestimmte Aufgaben so zu verteilen, dass sie möglichst oft zum Zug kommen, jedoch andere Aufgaben nicht blockieren. Ein solches System nennt man **Multi-Tasking**, wobei Task hier für Aufgabe/Job steht.

Um die Vorteile von Multi-Tasking aufzuzeigen benutze ich hier Funktionen, die sich gut im Simulator darstellen lassen. Diese stehen stellvertretend für oben erwähnte, die in einer echten Anwendung denkbar sind.

Die Demo benutzt ein 1*16 LCD um einen (43 Zeichen langen) Text als Laufschrift anzuzeigen.

Eine 7-Segment-Anzeige zeigt gleichzeitig den Start-Index des angezeigten Text-Teils.

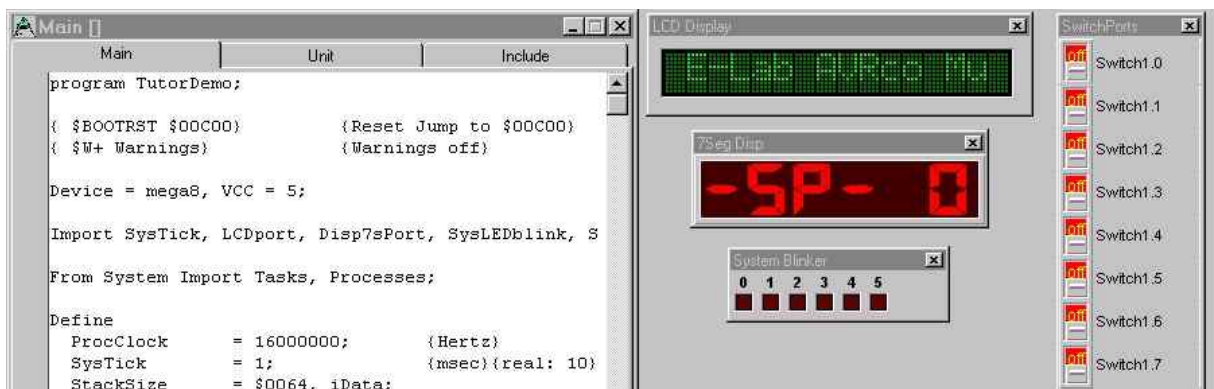
Weiterhin sollen 6 LEDs periodisch ein- und ausgeschaltet werden (blinken).

All dies sind Hintergrund Jobs, die parallel zum "Haupt-Job" ausgeführt werden sollen.

Der "Haupt-Job" liest einen Taster ein und setzt eine Variable, die den Status der LEDs bestimmt (blinken / aus / ein).

Öffnen Sie das *Project Management* und laden Sie das *TutorDemo* Projekt!

Starten Sie den Simulator:



Starten Sie das Programm (F9) und testen Sie *Switch1.0* (denken Sie daran: nicht zu schnell).

Schauen Sie sich mal das "*Main Program*" an: es liest nur den Schalter ein und setzt dann den Status der LEDs.

Die Laufschrift, die "-SP-Anzeige" (StringPointer) und das Ein/Ausschalten des LEDs passiert ohne eine einzige Anweisung im Hauptprogramm.

Überlegen Sie sich mal welche Vorteile Ihnen das bringen kann:

Sie erstellen und testen einen einzigen Teil Ihrer Anwendung, z.B. die Laufschrift.

Sobald dieser Teil funktioniert definieren Sie ihn als (Hintergrund-)Job, Sie können die Laufschrift vergessen und sich auf die weiteren Aufgaben konzentrieren.

Oder der "*System Blinker*":

egal, wo Sie in Ihren Programm den *LEDstate* zu *FlashOn* setzen – das Blinken wird durch einen Hintergrund-Prozess erledigt. Darum brauchen Sie sich nicht mehr zu kümmern, Sie brauchen noch nicht mal einen Task dafür. Der *System Blinker* ist Teil des Scheduler, der nur importiert werden muß (der Scheduler ist die Steuerung, die das Aktivieren der Jobs regelt).

Aber denken Sie daran:

in diesem Beispiel werden nur unkritische Tasks verwendet. Es macht nichts aus, wenn es einige Millisekunden länger dauert, einen Schalter zu erkennen oder eine LED umzuschalten.

Sobald Sie "Echtzeit-Anwendungen" als Tasks formulieren und gewisse Antwortzeiten garantieren müssen, kann Multitasking sehr kompliziert werden. In solchen Systemen müssen Sie die ganzen Multitasking Techniken wie Prioritäten, Semaphore, Pipes usw. verstehen.



AVRco Tools

2.10 Nützliche Links

E-LAB Homepage: <http://www.e-lab.de/index.html>
AVRco: <http://www.e-lab.de/AVRco/index.html>
enthält auch Links zu Pascal Tutorials !
Programmer: siehe Hardware / Programmiers
Mini Boards: <http://www.e-lab.de/diverse/components.html>
Datenblätter: http://www.e-lab.de/AVRco/avr_sheets.html (1)
Forum: <http://www.e-lab.de/phpBB2/>

Englisches Forum: <http://www.avrfreaks.net/>
Controller Manuals: siehe Devices

Eigenbau Programmer und Programmiersoftware:
<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>
http://ln.com.ua/~real/avreal/index_e.html (Kommandozeilen-Programm)

WARNUNG:

es gibt sehr einfache Programmer, die nur aus einigen Widerständen bestehen. Diese sollte man NICHT für den "Normalbetrieb" benutzen, da ein Fehler in Ihrer Schaltung den Parallel Port des PCs zerstören kann. Außerdem bringen diese auch andere Einschränkungen mit sich.

Manchmal können sie jedoch nützlich sein um herauszufinden ob man ein Problem mit der Programmier-Hardware oder –Software hat.

Viele verschiedene Links für alle möglichen Themen stehen bei

<http://www.mikrocontroller.net/links.htm>

Bem.:

(1): Datenblätter und Controller Manuals sind fast ausschließlich nur auf Englisch verfügbar. Bestenfalls werden Sie in seltenen Fällen mal einen Auszug von einem Kapitel finden, den jemand ins Deutsche übersetzt hat.

2.11 Anhang

2.11.1 Die AVRco Versionen

E-LAB bietet zwei Versionen des AVRco an:

- die Standard Version
die Standard Version unterstützt alle gängigen AVR Controller und bietet Treiber für nahezu alle Controller-internen Funktionen ("*On Chip Drivers*") und eine große Anzahl Treiber für zusätzliche Hardware ("*Soft Drivers*").
Sie beinhaltet einen V24/RS232 Programmieradapter "ISP-V24", der SPI und JTAG Programmierung unterstützt.
- die Professional Version
zusätzlich zur Standard Version, bietet die Professional Version fortgeschrittene Funktionen wie die Unterstützung von:
Units, Graphischen LCDs, File System, IP Stacks (UDP and TCP), Heap und JTAG Debugging. Sie beinhaltet einen USB Programmieradapter "ISP-USB", der SPI und JTAG Programmierung, sowie auch JTAG Debugging unterstützt.

Wie oben erwähnt gibt es auf der E-LAB Home Page auch eine kostenlose Demo Version. Der einzige Unterschied zwischen der Standard- und der Demo-Version ist die Beschränkung der Demo auf 4k Programmgröße. Ansonsten hat die Demo den vollen Funktionsumfang der Standard Version.



2.12 Referenz:

2.12.1 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;

```



AVRco Tools

```

if 12 <> 12a                                {new text ?}
then
  12a := 12;
  LCDxy(0,1);
  write (LCDout,Text[2,12]);
endif;

endloop;
end Tutor01.

```

2.12.2 Die Mega8 Konfiguration Bytes

(Default ist "internal oscillator at 1 MHz")

Mega 8	default		>= 8 MHz	
Bit Name	binary value	P / U	binary value	P / U
		(Un/Programmed)		(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 Übersicht

3.1.1 Einleitung

PED32 ist eine sogenannte IDE (Integrated Development Environment = Integrierte Entwicklungs Umgebung).

PED32 (Programmers Editor für WIN95) dient als Plattform für diverse Compiler, Assembler und Debugger.

PED32 enthält eine komfortable Projekt-Verwaltung, wie sie selten zu finden ist.

PED32 enthält einen Multi-Window Editor (MDI) vergleichbar mit dem von Borland Delphi.

PED32 kann, bei korrekter Einstellung, alle vom Compiler, Assembler und Linker erfasste Fehler anzeigen.

PED32 lädt im Fehlerfall automatisch die jeweilige Datei und positioniert den Cursor an die Fehlerposition

PED32 führt ein Fehlerprotokoll, das mit der Maus eingeschaltet und durchsucht werden kann.

PED32 erfasst projektbezogen die aufgelaufene Zeit, die das Projekt in Anspruch genommen hat.

PED32 kann praktisch unbegrenzt grosse Dateien bearbeiten.

PED32 erkennt automatisch durch Compiler usw. veränderte Dateien und lädt sie neu.

PED32 erkennt automatisch durch andere Programme veränderte Dateien und lädt sie neu.

PED32 hat eine konfigurierbare Syntax-Highlight.

PED32 unterstützt Syntax-bezogene Help-Files.

PED32 unterstützt Hervorhebung der Syntax in Schriftart und Farbe

PED32 ist in weiten Grenzen konfigurierbar und damit an unterschiedlichste Tools anpassbar.

PED32 ist damit, (nach Meinung des Programmierers!), eine der besten, nicht an bestimmte Tools gebundene IDE



AVRco Tools

3.2 Projekte

Im Gegensatz zu vielen anderen IDEs und Editoren ist **PED32** Projekt-bezogen und weniger Datei-bezogen. Das bedeutet, dass in erster Linie mit einem Projekt gearbeitet wird und nicht mit einzelnen Dateien.

Ein **Projekt** umfasst dabei ein oder mehrere Quelldateien, darunter das **MainFile**, den **ProjektPfad**, die aufgewendete Arbeits- bzw. **Programmierzeit** sowie ein projektbezogenes Steuerfile, das sog. **Control**.

Bei der Erstellung eines neuen Projekts über den Menüpunkt **Project/Edit Project** und dem Dialog **Project Admin** oder dem entsprechenden SpeedButton muss, neben den projektspezifischen Einstellungen wie Projekt Name, MainFile, Projekt Pfad etc., auch eine passende Steueranweisung (Control) mit angegeben werden. Der Dialog bietet eine Auswahl der vorhandenen Controls mit an.

Sind alle Angaben korrekt und komplett, so wird z.B., falls nicht schon vorhanden, im Falle von Control = PICpas automatisch ein Mainfile erstellt, das als Gerüst (Template) verwendet werden kann. Beim AVR wird hier jedoch der Application Wizard aufgerufen, der bei der Erstellung des Programm Gerüsts hilft.

Weitere Einstellungen sind normalerweise nicht erforderlich, da das verwendete Control alle weiteren Anweisungen für die IDE enthält.

3.3 Controls

Wie weiter oben unter **Projekte** schon zu ersehen war, spielt die Steueranweisung **Control** eine wesentliche Rolle beim Arbeiten mit einem Projekt. Control ist im grossen ganzen eine Batch Beschreibung.

Einmal mittels dem Menüpunkt **System/System Admin** und dem Dialog **System Admin** korrekt und komplett erstellt, braucht das Control in aller Regel nicht mehr zu verändert werden.

Fast jeder Punkt bzw. Zeile in diesem Dialog hat ein zugehörigen Knopf in der oberen Knopfleiste, den sogenannten SpeedButtons. Die Make-Zeile und der Simulator/Debugger können neben den jeweiligen SpeedButtons auch durch CTRL + F9 bzw. F9 aufgerufen werden.

Jede Zeile besteht aus Anweisungen, in der Regel EXE-Dateien, die von links nach rechts abgearbeitet werden.

Weiterhin können in einer Zeile auch Zahlen stehen, von Text und anderen Zahlen durch ein Separatorzeichen „|“ getrennt. Damit werden andere Zeilen innerhalb des Dialogs einbezogen. Auf Rekursionen achten!!

Ein Beispiel: Das Control PICpas ist eine Steueranweisung für den E-LAB Pascal Compiler für die MicroChip PIC Prozessor Familie. Für alle zu erstellende Projekte mit PICs und dem Pascal Compiler muss nur das Control PICpas bei der Projekterstellung angegeben werden, und man kann sofort editieren, compilieren, debuggen usw.

An dieser Stelle soll jedoch nicht verschwiegen werden, dass die Erstellung eines neuen Controls sehr sorgfältiger Überlegung bedarf. Ausserdem muss man sich mit den zugehörigen Dialogen intensiv beschäftigen. Der später daraus resultierende Automatismus bedarf etwas Vorarbeit.

3.4 Syntax

Die Spracheinstellung (Compilersprache wohlgemerkt) ist in der Regel auf ein bestimmtes Control bezogen und ist deshalb in einem Unterdiallog von System Admin angesiedelt. Es ist aber unter Umständen, z.B. bei mehreren unterschiedlichen Pascal Compilern, sinnvoll auch mehrere Pascal „Sprachen“ zu haben. Diese müssen separat erstellt und in das jeweilige Control unter Syntax einbezogen werden.

Die Erstellung einer jeweiligen Sprach-Syntax Liste ist allerdings kein muss. Diese dient nur zur Hervorhebung von bestimmten Sprachelementen innerhalb des Editors.

3.5 Menue

Übersicht der Menue-Punkte von PED32

3.5.1 File Menue

New		Eine neue Datei eröffnen.
Open		Ruft den Datei-Open Dialog auf.
Open Mainfile		Öffnet das MainFile des Projekts
Open Mapfile		Öffnet das MapFile des Projekts
Save	Ctrl + S	Sichert das aktuelle Editorfenster in die entspr. Datei
Save As		Speichert das aktuelle Editorfenster unter einem neuen Namen
Close File		Schliesst das aktuelle Editorfenster mit evtl. Dateisicherung
History		Zeigt die 10 zuletzt bearbeiteten Projekte in zeitlicher Reihenfolge.
		Durch Anklicken wird das gewählte Projekt geladen.
Insert File		Eine Datei an die aktuelle Cursor Position einlesen.
Save Block		Den markierten Block als Datei wegschreiben.
Print		Die aktuelle Datei (Fenster) oder Block ausdrucken.
Exit	Alt + F4	PED32 beenden.

3.5.2 Edit Menue

Undo	Ctrl + Z	Letzte Änderung widerrufen (Undo)
Redo	Shift + Ctrl + Z	Letztes Undo widerrufen (Redo)
Cut	Ctrl + X	Markierter Block ausschneiden und in die Windows-Ablage kopieren
Copy	Ctrl + C	Markierter Block in die Windows-Ablage kopieren
Paste	Ctrl + V	Windows-Ablage an die Cursor Position kopieren
Delete		Zeichen rechts vom Cursor oder ganzer Block löschen
Select All		Block markieren von Datei Anfang bis Ende

3.5.3 Search Menue

Find	Ctrl + F	Wort (am Cursor) suchen
Replace	Ctrl + R	Wort (am Cursor) suchen und ersetzen
Find Next	F3	Letztes Suchen/Ersetzen wiederholen
Replace Tabs by Sofftabs		Ersetzt harte Tabs durch Sofftabs
Replace Tabs by Spaces		Ersetzt harte Tabs durch Leerzeichen
Goto Line	F4	Cursor auf Zeilennummer x setzen
Clear all Markers		Setzt alle Marker zurück

3.5.4 Project Menue

Load Project		Öffnet Dialog zum Laden eines Projekts
Edit Project		Öffnet Dialog zum Erstellen bzw. Verändern eines Projekts
Information		Zeigt den aktuellen Stand des Projekts an
Project Options		Editieren von Compiler Schaltern für Conditional Compile. Einstellen von projektbezogenen Suchpfaden

3.5.5 System Menue

System Options		Compiler/Assembler Window anzeigen bzw. verstecken. Einstellen von systembezogenen Suchpfaden
System Admin		Spezielle Einstellungen der Controls (Steueranweisungen für div. Compiler)



AVRco Tools

3.5.6 IDE Menue

General Options	Öffnet den Dialog zur Font- und Farbeinstellung und weiterer Optionen
Tabs	Einstellen der TABs
Popup delay	Verzögerung der Syntax Hilfe
Edit Keyboard Macros	Öffnet Dialog zum Editieren von Keyboard Macros

3.5.7 Window Menue

Arrange Icons	Auf Icons verkleinerte Editorfenster neu anordnen
Minimize all	Alle Editor Fenster auf Icongröße verkleinern
Window ...	Ein bestimmtes Editorfenster in den Vordergrund bringen

3.5.8 Info Menue

Help IDE	Ctrl + F1	Die IDE bzw. Editor Hilfe im Kontext aufrufen
Help Syntax	F1	Die Compiler bzw. Syntax Hilfe im Kontext aufrufen
Info IDE		Die Übersicht der IDE bzw. Editor Hilfe aufrufen
Info Syntax		Die Übersicht der Compiler bzw. Syntax Hilfe aufrufen
About		Anzeige der aktuellen Version etc. von PED32

3.6 Dialoge

Übersicht der Dialoge von PED32

3.6.1 Project Admin

Dient zum Laden oder Verändern eines vorhandenen Projekts, als auch zum Erstellen eines neuen Projekts. Anzeige der bisjetzt aufgelaufenen Arbeitszeit der einzelnen Projekte.

Untergeordnete Dialoge:

Project Path	Dient zum Einstellen des Pfades des gewählten Projekts
Main File	Dient zur Auswahl des Main Files .

3.6.2 Project Options

Hier können Compiler Schalter für das Conditional Compile definiert werden. Diese werden dem Compiler übergeben, der sie wie ein **{\$DEFINE Label}** behandelt. Weiterhin können projektbezogene Suchpfade angegeben werden, die der Compiler etc. auswerten kann. Die Pfade werden im *.ppro File abgelegt.

3.6.3 Project Info

Anzeige des Status des aktuellen Projekts sowie allgemeiner Daten wie RAM und ROM Verbrauch und Belegung. Lage und Grösse der Stacks und Frames.

3.6.4 General Options

Dient zum Einstellen der Font/Zeichengrösse sowie der Farben für normalen Text und dessen Hintergrund, markierter Text und Hintergrund, Fehlertext und dessen Hintergrund sowie Kommentar und Strings. Das allgemeine Verhalten des Editors (Cursor, Caret, Backup, schnell-Hilfe etc.) werden hier ebenfalls eingestellt.

3.6.5 Macro Editor

Dient zum Erstellen von Tastatur Macros. Hiermit können ganze Code-Blöcke erstellt und editiert werden. Diese Blöcke werden mittels Hotkeys in den laufenden Text eingefügt.

3.6.6 Zeichen Tabelle

Dient zum Einfügen von Sonderzeichen in den Source Text. Ausserdem kann hier der Hex- oder Dezimalwert eines Zeichens festgestellt werden.

3.6.7 System Admin

Dient zum Erstellen der **Controls** (Steuerdateien) für die einzelnen Tools wie z.B. Compiler, Assembler etc. Weiterhin wird hier die Syntax der verwendeten Sprachen beschrieben, welche für die Syntax-Hervorhebung des Editors benötigt wird. Ebenso muss hier auch die Fehlerauswertung (Fehler-Datei Aufbau) der Tools deklariert werden.

Untergeordnete Dialoge:

Syntax + FileMasks	Dient zur Auswahl der Sprache, z.B. Pascal, der Limiter für Kommentare sowie der Datei Masken, z.B. *.PAS, *.ASM
Error Definitions	Deklaration der Fehlerdateien und deren interner Aufbau
Help File	Name und Pfad des zum Compiler gehörigen Help Files

3.6.8 File Open

Dient zum Auswählen und Laden der gewünschten Datei inklusive deren Pfad

3.6.9 File Save As

Dient zum sichern der aktuellen Datei unter einem anderen Namen und/oder Pfad

3.6.10 Print

Dient zur Auswahl eines Druckers, dessen Einstellung und Drucken des aktuellen Editor Files.

3.6.11 Find

Dient zur Einstellung der Such Optionen und zum Suchen Start.

3.6.12 Replace

Dient zur Einstellung der Suchen/Ersetzen Optionen und zum Suchen/Ersetzen Start.

3.6.13 Goto Line

Positioniert den Cursor in die angegebene Zeile.

3.6.14 TabSize

Stellt die Tabulator Länge ein.

3.7 SpeedButtons



Übersicht der Speedbuttons von PED32

3.7.1 FileOpen



Ruft den Datei-Open Dialog auf.

3.7.2 Save



Ctrl + S

Sichert das aktuelle Editorfenster in die entspr. Datei

3.7.3 Projekt Verwaltung



Öffnet den Projekt Dialog

3.7.4 Application Wizard



Startet den Programm Generator

3.7.5 Printer Dialog



Öffnet den Drucker Dialog

3.7.6 Cut



Ctrl + X

Schneidet den markierten Block aus und kopiert ihn in die Windows Ablage

3.7.7 Copy



Ctrl + C

Kopiert den markierten Block in die Windows Ablage

3.7.8 Paste



Ctrl + V

Kopiert den Inhalt der Windows Ablage an die Cursor Position

3.7.9 Find



Ctrl + F

Öffnet den Such Dialog

3.7.10 Replace



Ctrl + R

Öffnet den Suchen/Austauschen Dialog

3.7.11 Undo



Ctrl + Z

Macht die letzte Text Änderung rückgängig

3.7.12 Tile horizontal



Teilt das vorhandene PED32 Window gleichmässig unter den geöffneten Editoren auf. Die Fenster stehen dabei untereinander.

3.7.13 Tile vertikal



Teilt das vorhandene PED32 Window gleichmässig unter den geöffneten Editoren auf. Die Fenster stehen dabei nebeneinander.

3.7.14 Cascade



Die geöffneten Editoren werden hintereinander absteigend plaziert

3.7.15 Split Window



Das aktuelle Editorfenster wird in zwei Fenster aufgeteilt. Beide Fenster beinhalten dasselbe File, jedoch mit unterschiedlichen Positionen im File. Damit kann innerhalb einer Datei an zwei unterschiedlichen Stellen gleichzeitig editiert werden. Für manche Operationen sehr hilfreich!

3.7.16 Calculator



öffnet den Taschenrechner

3.7.17 Projekt Info



Zeigt den Projekt Status an

3.7.18 Alphabet



Öffnet den Alphabet Dialog

3.7.19 Make



Ctrl + F9

Die in dem eingestellten Control in der Make-Zeile stehende Kommando-Zeile wird abgearbeitet. Evtl. Fehler werden anschliessend ausgewertet.

3.7.20 Compile



Die in dem eingestellten Control in der Compile-Zeile stehende Kommando-Zeile wird abgearbeitet. Evtl. Fehler werden anschliessend ausgewertet.

3.7.21 Link



Die in dem eingestellten Control in der Link-Zeile stehende Kommando-Zeile wird abgearbeitet. Evtl. Fehler werden anschliessend ausgewertet.



AVRco Tools

3.7.22 Post Processor



Die in dem eingestellten Control in der PostProc-Zeile stehende Kommando-Zeile wird abgearbeitet. Keine Fehlerauswertung.

3.7.23 Debugger



Die in dem eingestellten Control in der Debugger-Zeile stehende Kommando-Zeile wird abgearbeitet. Keine Fehlerauswertung.

3.7.24 Simulator



F9

Die in dem eingestellten Control in der Simulator-Zeile stehende Kommando-Zeile wird abgearbeitet. Keine Fehlerauswertung.

3.7.25 Assembler



Die in dem eingestellten Control in der Assembler-Zeile stehende Kommando-Zeile wird abgearbeitet. Evtl. Fehler werden anschliessend ausgewertet.

3.7.26 RomSim/Prommer



Die in dem eingestellten Control in der RomSim-Zeile stehende Kommando-Zeile wird abgearbeitet. Keine Fehlerauswertung.

3.7.27 Tool



Zur Zeit noch ohne Funktion

3.7.28 Librarian



Die in dem eingestellten Control in der Library-Zeile stehende Kommando-Zeile wird abgearbeitet. Evtl. Fehler werden anschliessend ausgewertet.

3.7.29 DisAssembler



Die in dem eingestellten Control in der DisAsm-Zeile stehende Kommando-Zeile wird abgearbeitet. Keine Fehlerauswertung.

3.8 Status Leiste

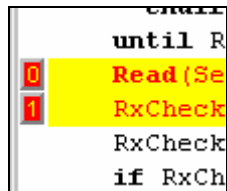
238:1	Modified	Total: 537	Top: 238	Bytes: 13974	Insert	Errors: 0
-------	----------	------------	----------	--------------	--------	------------------

Cursor Pos Zeile Spalte	Text ver- ändert	Gesamt Zeilenzahl	Oberste Zeile im Fenster	Datei Grösse	Insert bzw. Overwrite Modus	Anzahl zuletzt auf- getretener Fehler
-------------------------------	------------------------	----------------------	-----------------------------	-----------------	-----------------------------------	--

3.9 Fehler Fenster



Treten bei einer Compilierung oder Assenblierung Fehler auf oder Warnungen sind im Compiler freigegeben und diese treten auf, so wird am unteren Rand des Editors ein Fehlerfenster geöffnet. Dieses enthält alle Fehler und evtl. Warnungen mit einer Erklärung. Durch einen Doppelclick auf eine dieser Zeilen scrollt der Editor an die zugehörige Stelle in der Source..



In der Source werden die fehlerhaften Zeilen mit der entsprechenden Farbe markiert und am linken Rand befinden sich die Fehlernummern. Ein Click auf diese Nummer scrollt das unten stehende Fehlerfenster so, dass der zugehörige Fehler und dessen Erklärung sichtbar ist. Diese Zeile erhält zusätzlich ein HighLight.

Wird die Source Zeile wesentlich geändert oder gelöscht, wird das Fehler Highlight und die Nummer in der Source gelöscht.

3.10 HotKeys und ShortCuts = Tastatur Kommandos

3.10.1 IDE und Syntax Help

Ctrl + F1 Help der IDE aufrufen. Help ist abhängig vom aktuellen Dialog bzw. Focus
F1 Syntax-Help aufrufen. Help ist abhängig vom Wort unter dem Cursor.

3.10.2 Datei und Fenster Operationen

Ctrl + S Aktuelles Editor Fenster speichern
Ctrl + TAB Umschalten zum nächsten Editor Fenster

3.10.3 Cursor bewegen

← Cursor 1 Zeichen nach links
→ Cursor 1 Zeichen nach rechts
↑ Cursor 1 Zeile nach oben
↓ Cursor 1 Zeile nach unten
Ctrl + ← Cursor 1 Wort nach links
Ctrl + → Cursor 1 Wort nach rechts
HOME Cursor an Zeilen Anfang
END Cursor an Zeilen Ende
PageUp 1 Bildschirmseite Richtung Dateianfang
PageDown 1 Bildschirmseite Richtung Dateieinde
Ctrl + PageUp Cursor an aktuelle Bildschirmseite Anfang
Ctrl + PageDown Cursor an aktuelle Bildschirmseite Ende
Ctrl + HOME Cursor an Datei Anfang
Ctrl + END Cursor an Datei Ende
F4 Cursor auf Zeilennummer x setzen
Ctrl + n Sprung zum Merker "n" (0..9)
Shift + Ctrl + n Merker "n" (0..9) setzen

3.10.4 Editieren

Insert Einfügen/Überschreiben Ein/Aus
Delete Zeichen rechts vom Cursor löschen oder ganzer Block
BackSpace Zeichen links vom Cursor löschen oder ganzer Block
Ctrl + Enter Leere Zeile an Cursor Position einfügen



AVRco Tools

Ctrl + N	Leere Zeile an Cursor Position einfügen
Ctrl + Y	Zeile an Cursor Position komplett löschen
Ctrl + T	Wort rechts von Cursor Position komplett löschen
Ctrl + P	Steuerzeichen an Cursor Position einfügen
Ctrl + Z	Letzte Änderung widerrufen (Undo)
Alt + BackSpace	Letzte Änderung widerrufen (Undo)
Shift + Ctrl + Z	Letztes Undo widerrufen

3.10.5 Suchen/Austauschen

Ctrl + F	Wort (am Cursor) suchen
Ctrl + R	Wort (am Cursor) suchen und austauschen
F3	Suchen bzw. Austauschen wiederholen

3.10.6 Cursor Block Befehle

Shift + ←	Cursor 1 Zeichen nach links mit Blockerweiterung/Verkürzung
Shift + →	Cursor 1 Zeichen nach rechts mit Blockerweiterung/Verkürzung
Shift + ↑	Cursor 1 Zeile nach oben mit Blockerweiterung/Verkürzung
Shift + ↓	Cursor 1 Zeile nach unten mit Blockerweiterung/Verkürzung
Shift + Ctrl + ←	Cursor 1 Wort nach links mit Blockerweiterung/Verkürzung
Shift + Ctrl + →	Cursor 1 Wort nach rechts mit Blockerweiterung/Verkürzung
Shift + HOME	Cursor an Zeilen Anfang mit Blockerweiterung/Verkürzung
Shift + END	Cursor an Zeilen Ende mit Blockerweiterung/Verkürzung
Shift + PageUp	1 Bildschirmseite Richtung Dateianfang mit Blockerweiterung/Verkürzung
Shift + PageDown	1 Bildschirmseite Richtung Dateiende mit Blockerweiterung/Verkürzung
Shift + Ctrl + PageUp	Cursor an aktuelle Bildschirmseite Anfang mit Blockerweiterung/Verkürzung
Shift + Ctrl + PageDown	Cursor an aktuelle Bildschirmseite Ende mit Blockerweiterung/Verkürzung
Shift + Ctrl + HOME	Cursor an Datei Anfang mit Blockerweiterung/Verkürzung
Shift + Ctrl + END	Cursor an Datei Ende mit Blockerweiterung/Verkürzung

3.10.7 Block bearbeiten

Shift + Delete	Block ausschneiden, Kopie des Blocks in die Windows-Ablage
Ctrl + X	Block ausschneiden, Kopie des Blocks in die Windows-Ablage
Ctrl + Insert	Block in die Windows-Ablage kopieren. Block unverändert.
Ctrl + C	Block in die Windows-Ablage kopieren. Block unverändert.
Shift + Insert	Block aus der Windows-Ablage an Cursor Position kopieren.
Ctrl + V	Block aus der Windows-Ablage an Cursor Position kopieren.
Shift + Ctrl + I	Block nach rechts einrücken.
Shift + Ctrl + U	Block nach links ausrücken.

3.10.8 Diverse

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

3.10.9 Keyboard Macros

Mit dem Dialog **Edit Keyboard Macros** kann der Anwender Textbausteine erstellen, z.B. Programmsequenzen, die einem zugehörigen ShortCut bzw. HotKey an die Stelle des Carets (Einfüge Cursor) in den Text eingefügt werden können.

3.11 Projekte

3.11.1 Arbeiten mit Projekten

Im Gegensatz zu vielen anderen IDEs und Editoren ist **PED32** Projekt-bezogen und weniger Datei-bezogen. Das bedeutet, dass in erster Linie mit einem Projekt gearbeitet wird und nicht mit einzelnen Dateien. Ein einmal erstelltes Projekt ist, bis es gelöscht wird, für immer der IDE bekannt. Der Programmierer braucht sich nicht weiter um Verzeichnisse und zugehörige Tools kümmern. Einfach das Projekt laden und schon kann gearbeitet werden.

Ein Verknüpfen von z.B. Pascal-Dateien mit PED32 ist nicht notwendig und auch nicht sinnvoll. Die einzigste sinnvolle Verknüpfung mit PED32 ist die von *.ppro Dateien.

Ein **Projekt** umfasst ein oder mehrere Quelldateien, darunter das **MainFile**, den **ProjektPfad**, die aufgewendete **Arbeits-** bzw. **Programmierzeit** sowie eine projektbezogene Steueranweisung, das sog. **Control**.

Bei der Erstellung eines neuen Projekts über den Menüpunkt **Edit Project** und dem Dialog **Project Admin** oder dem entsprechenden SpeedButton muss, neben den projektspezifischen Einstellungen wie Projekt Name, MainFile, Projekt Pfad etc., auch eine passende Steueranweisung (Control) mit angegeben werden. Der Dialog bietet eine Auswahl der vorhandenen Controls mit an.

Sind alle Angaben korrekt und komplett, so wird z.B., falls nicht schon vorhanden, automatisch ein Mainfile erstellt, das als Gerüst (Template) verwendet werden kann. Das Template ist ein normales Text-File. Der Name und der Pfad dieses Templates muss beim Erstellen des jeweiligen **Controls** eingetragen werden. Ist im Control ein Application Wizard (Programm Generator) eingestellt, so wird statt dem Template der Programmgenerator gestartet.

Weitere Einstellungen sind normalerweise nicht erforderlich, da das verwendete Control alle weiteren Anweisungen für die IDE enthält.

Wie schon erwähnt, speichert die IDE PED32 die Daten aller Projekte. Ein zukünftiger Aufruf eines angelegten Projekts beschränkt sich auf 2 Clicks. Es ist zwar möglich, auch Einzel Dateien zu bearbeiten, der eigentliche Zweck ist aber die Dateibearbeitung innerhalb eines Projekts.


Nach einer gewissen Umgewöhnungszeit vom normalen Editor auf Projekte lernt der Anwender die Vorzüge sehr schnell schätzen.

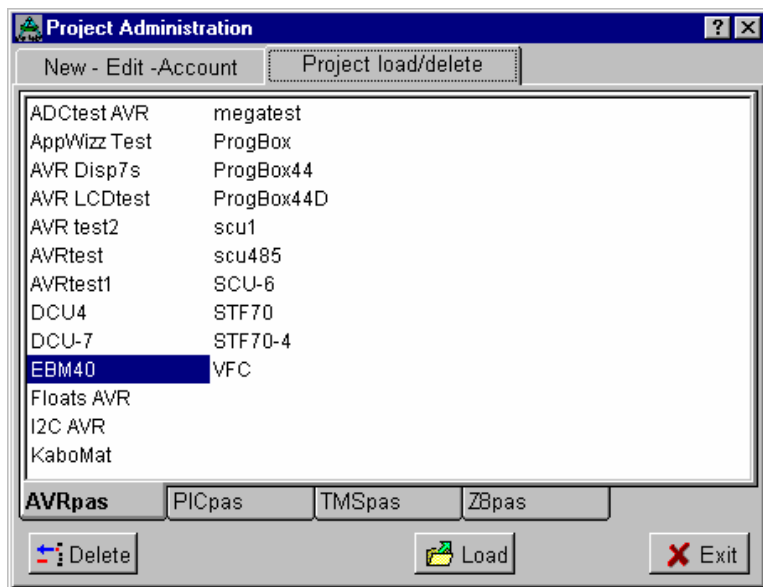
Dabei wird auch der geringe Mehraufwand für die Erstellung der Projekt-Parameter eines neuen Projekts gerne in Kauf genommen.



AVRco Tools

3.11.1.1 Load Project

Um ein vorhandenes Projekt zu laden, muss der Dialog **Project Admin** geöffnet werden. Das kann über den Menü-Punkt **Project .. Edit/Load Project** oder über den SpeedButton  geschehen. Daraufhin wird der folgende Dialog eröffnet. Die Dialogseite **Project load/delete** wird durch ein Click auf die jeweilige Klappe gewählt.



Durch Doppelclick auf den gewünschten Eintrag oder Click auf die Taste Load wird das gewählte Projekt geladen.

Eine weitere Möglichkeit besteht im File-Menue über den Menüepunkt **History**. Hier wird eine Auflistung der zuletzt bearbeiteten Projekte in zeitlicher Reihenfolge angezeigt. Auch hier mit einem Click das gewünschte Projekt Laden.

Obsolete Projekte werden mit der **Delete** Taste gelöscht.

Die den einzelnen Controls (Compilern) zugeordneten Projekte werden in separaten Seiten angezeigt.

Nach der Auswahl des gewünschten Projekts werden alle projekt-relevanten Daten aus der Datei **PED32.ini** geladen. Diese Daten enthalten u.a. auch das Heimat Verzeichnis des Projekts und das verwendete Control.

PED32 schaltet jetzt in Windows die **Current Directory** auf dieses Verzeichnis. Alle weiteren Datei Operationen innerhalb der IDE, die ohne Pfad Angaben gemacht werden, gehen deshalb jetzt in dieses Verzeichnis.

Als nächstes lädt die IDE die Projekt Steuerdatei **xxx.ppro**. Diese Datei enthält jetzt alle weiteren Projektdaten, so auch z.B. die zuletzt geöffneten Dateien.

Das zugehörige Control wird jetzt geladen und anschliessend die zuletzt geöffneten Dateien. Zuletzt wird noch geprüft, ob eine Fehlerdatei vorliegt. Ist das der Fall, wird diese geöffnet und ausgewertet. Damit ist der Ladevorgang abgeschlossen und es kann gearbeitet werden.


Ein Starten oder Laden mittels DragAndDrop ist nicht sinnvoll und auch nicht vorgesehen.

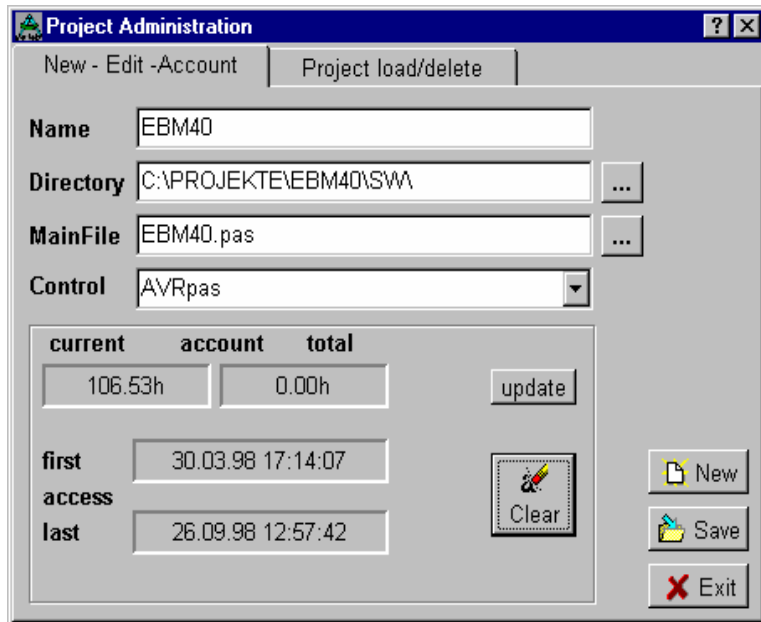
Eine Verknüpfung von *.ppro Dateien mit PED32 jedoch ist möglich. Wurde diese Verknüpfung einmal erstellt, so kann jederzeit innerhalb des Explorers und ähnlichen Programmen durch einen Doppelclick auf eine Datei im Format xxx.ppro die IDE PED32 gestartet werden. Diese lädt dann wiederum das geklickte Projekt.

Ein Doppelstart von PED32 ist nicht möglich, da beide Programme die gleichen INI-Dateien konkurrierend pflegen müssten, was garantiert zu seltsamen Ergebnissen führen würde.

Der Editor in PED32 überwacht alle geladenen Dateien. Wird extern, z.B. durch einen Editor eine geladene Datei verändert, kommt von der IDE die Information dass die Datei verändert wurde und die Abfrage ob sie neu geladen werden soll. Das Neuladen erfolgt automatisch, wenn z.B. der Compiler eine veränderte Assembler Source generiert.

3.11.1.2 Edit/New Project

Um ein vorhandenes Projekt zu verändern oder ein neues zu erstellen, muss der Dialog **Project Admin** geöffnet werden. Das kann über den Menue-Punkt **Project .. Edit/Load Project** oder über den SpeedButton  geschehen. Daraufhin wird der folgende Dialog eröffnet. Die Dialogseite **New-Edit-Account** wird durch ein Click auf die jeweilige Klappe gewählt.



current	account	total
106.53h	0.00h	

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

Soll ein neues Projekt angelegt werden, so ist jetzt die Taste **New** zu klicken. Das weitere Vorgehen ist für New und Edit identisch.

Im Editierfenster **Name** wird der gewünschte Projekt-Namen eingetragen. Dieser Namen wird auch als Projekt Dateinamen verwendet. Er kann jedoch alle Zeichen enthalten, die in WIN95 etc. erlaubt sind, z.B. Leerzeichen.

Durch editieren des Fensters **Directory** wird die Arbeits-Directory angegeben. Ein Doppelclick auf dieses Fenster oder den nebenstehenden Button eröffnet den Dialog **Project Path**. Hiermit kann eine vorhandene Directory als Arbeits-Directory ausgewählt werden. Durch editieren des Fensters **MainFile** wird das **Haupt-File** angegeben.

Eine weitere Möglichkeit besteht durch einen DoppelClick auf dieses Fenster oder Click auf den nebenstehenden Button den Dialog **MainFile** zu eröffnen und hiermit ein in der Arbeitsdirectory schon vorhandenes File zum MainFile zu erklären.

Existiert das eingetragene MainFile nicht, sow wird, falls im ausgewählten Control angegeben, ein Template-File unter dem Namen *MainFile* in die Arbeitsdirectory kopiert, falls kein Programm Generator vorgesehen ist.

Das Fenster **Control** zeigt das z.Zt. ausgewählte Control an. Durch ein Click auf die rechte kleine Pfeiltaste dieses Fensters werden die z.Zt. bekannten Controls zur Auswahl angeboten. Das gewünschte Control wird durch einen Click ausgewählt. Ein neues Control, falls notwendig, muss zuerst über das Menue **System\System Admin** und dem Dialog **SysAdmin** erstellt werden. Ein einmal erstelltes Control, z.B. PICpas, kann dann in Zukunft für alle Projekte mit dem Pascal-Compiler PICco32 herangezogen werden.

Durch einen Rahmen getrennt, werden links unten die Zeit-relevanten Projektdaten angezeigt. Dies sind die Zugriffsdaten, getrennt nach Projekt-Erstellung (**first access**) und letztem Aufruf (**last access**).

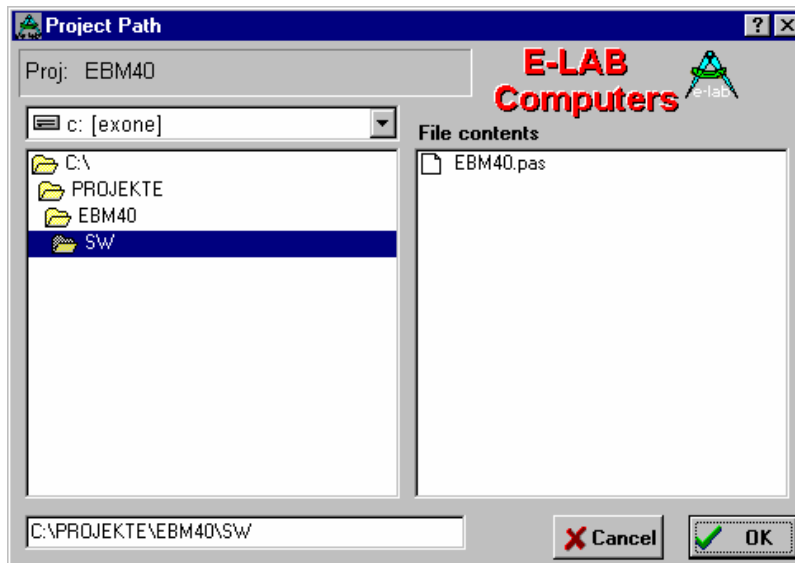
Darüber werden die aufgelaufenen Arbeitszeiten angezeigt. **Total account** bezeichnet den zuletzt abgerechneten Zeitaufwand. **Current account** bezeichnet die nach der letzten Abrechnung aufgelaufene Zeit. Die Summe beider Zeiten ist die tatsächlich bisjetzt aufgewendete Zeit für dieses Projekt.

Wird eine Zwischenabrechnung der Zeit gemacht, so ist die Taste **Update** zu klicken. Jetzt wird current account zu total account hinzuaddiert und current account auf Null gesetzt. Total account enthält jetzt die absolute Zeit.

Die Zeiten können mit **Clear** zurückgesetzt werden. Damit werden die access-Daten ebenfalls auf das momentane Datum und Uhrzeit gesetzt. Das Projekt wird sozusagen neu begonnen.

Alle Änderungen müssen mit **Save** bestätigt und abgespeichert werden. Wird das nicht gemacht, so wird beim Schliessen des Dialogs nachgefragt, ob eine Abspeicherung der Veränderungen gewünscht wird. Wird die Abfrage verneint, werden die Änderungen verworfen.

3.11.1.3 Project Path



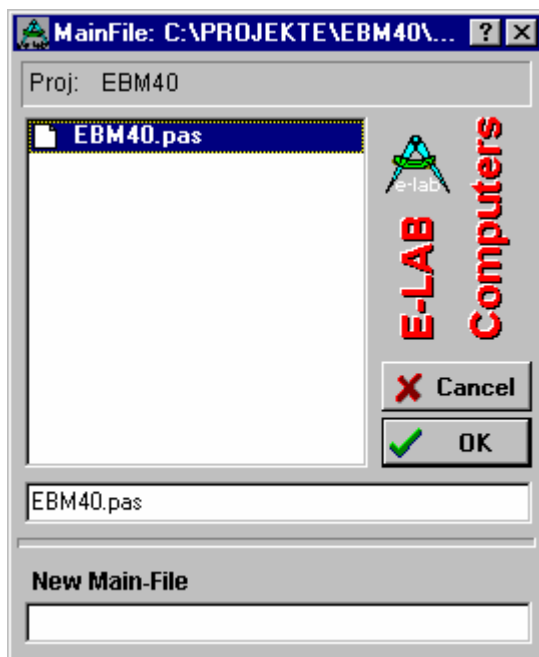
Unterdialog zum Dialog **Edit/New Project** .

Beim Neu-Anlegen oder Editieren eines Projekts muss dem Projekt eine Heimat- bzw. Arbeitsdirectory zugewiesen werden. Durch ein Doppelclick im Dialog **Edit/New Project** auf das Editfeld **Directory** oder den rechten Button wird der Dialog **Project Path** geöffnet.

Hier kann ein vorhandenes Verzeichnis zum Arbeitsverzeichnis des gewählten Projekts erklärt werden. Die rechte Hälfte enthält die relevanten Dateien dieser Directory zur Information. Bei der Auswahl muss darauf geachtet werden, dass der gewünschte Pfad

komplett im Feld ganz unten erscheint. Dieses Feld kann auch editiert werden. Existiert der veränderte Pfad nicht, wird anschliessend gefragt, ob die Directory neu erstellt werden soll.

3.11.1.4 MainFile



Unterdialog zum Dialog **Edit/New Project** .

Beim Neu-Anlegen oder Editieren eines Projekts muss dem Projekt eine Haupt Datei zugewiesen werden. Bei der Neuerstellung eines Projekts wird im Dialog **Edit/New Project** im Editfeld **Main File** der gewünschte Dateinamen eingegeben.

Existiert die Datei jedoch schon, kann durch ein Doppelclick auf das Feld oder Click auf den Button der **Dialog Main File** geöffnet werden.

Hier kann eine vorhandene Datei zum MainFile des gewählten Projekts erklärt werden. Existiert noch kein entsprechendes File, so muss im untersten Feld der Namen des neuen Mainfiles eingetragen werden. Um kompatibel mit anderen Tools zu sein, sollte der Datei Namen den DOS Konventionen entsprechen.

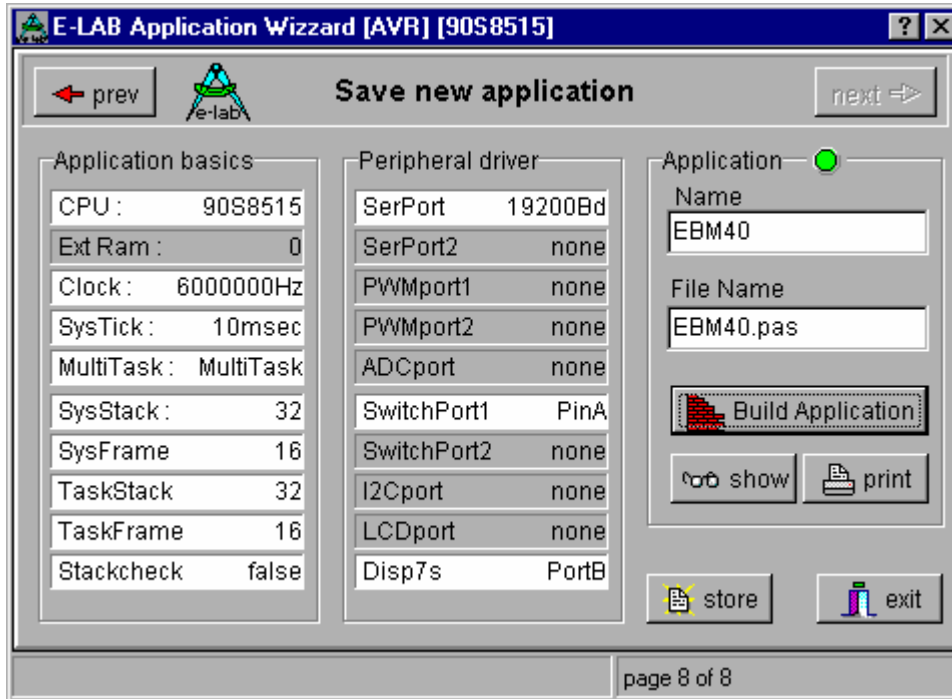
Ist ein Projekt einmal mit obigen Dialogen erstellt worden, so kann es jederzeit wieder aufgerufen werden.

In dem Arbeitsverzeichniss des Projekts wird dazu automatisch ein Projekt-Steuerfile erstellt und gepflegt. Diese Datei enthält alle Projektbezogenen Daten. Der Filename dieser Datei ist "*ProjektName.ppro*". Der Aufbau entspricht einer Windows-Ini-Datei und ist damit ein normales Textfile. Dieses Textfile sollte nur zu Reparaturzwecken verändert werden.

Am Ende des gesamten Vorgangs (Neuerstellung oder Editieren eines Projekts) prüft die IDE ob das eingetragene Projekt MainFile existiert. Ist das nicht der Fall, so wird im zugewiesenen Control nachgeprüft, ob ein Application Wizard oder ein Template spezifiziert worden ist und entsprechend verfahren.

3.11.1.5 Application Wizard

Ist im zugehörigen Control ein Application Wizard definiert, so wird dieser, falls bei einem neuen Projekt noch kein Mainfile existiert, aufgerufen. Mit dessen Hilfe kann komfortabel eine neue Applikation erstellt werden.



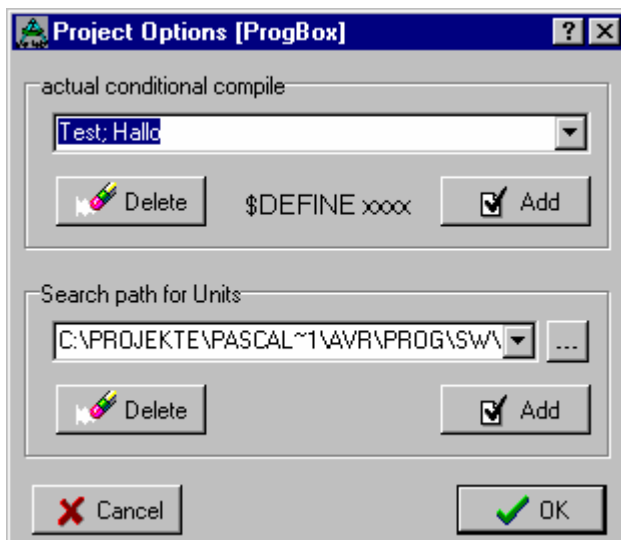
Beispiel für einen Applikations Experten

AppWizz für E-LAB
AVRco

3.11.1.6 Template

Existiert kein Wizard dafür aber eine Template-Datei, so wird sie unter dem MainFile Namen in das Arbeitsverzeichnis kopiert. Die Template Datei kann eine beliebige Textdatei mit beliebigem Inhalt sein.

3.11.1.7 Project Options



E-LAB Compiler der Serie **Pascal-scm** kennen das sogenannte Conditional Compile (siehe dazu Compiler Handbuch). Solche **\$DEFINE** Anweisungen können mit diesem Dialog erstellt, editiert und gültig gemacht werden. Der Dialog enthält alle für dieses Projekt erstellten Defines.

Ein neues Define wird im Editierfeld eingegeben. Mehrere zur gleichen Zeit gültige Werte müssen mit Strichpunkt getrennt werden. Ist die Zeile vollständig, muss sie mittels dem Add-Button hinzugefügt werden. Eine Liste aller Defines erhält man durch einen Click auf die Pfeiltaste. Durch einen weiteren Click auf eine Zeile des geöffneten Fensters wird eine schon erstellte Zeile ausgewählt. Mit der OK-Taste wird diese Definition allen zukünftigen Compile Läufen zugewiesen.

PED32 fügt diese Zeile in das xxx.ppro File ein, wo sie der Compiler abarbeitet. Der Compiler interpretiert die einzelnen Anweisungen dieser Zeile wie wenn sie im Sourcecode stehen würden. Aus der Zeile

Test; Sample

wird →

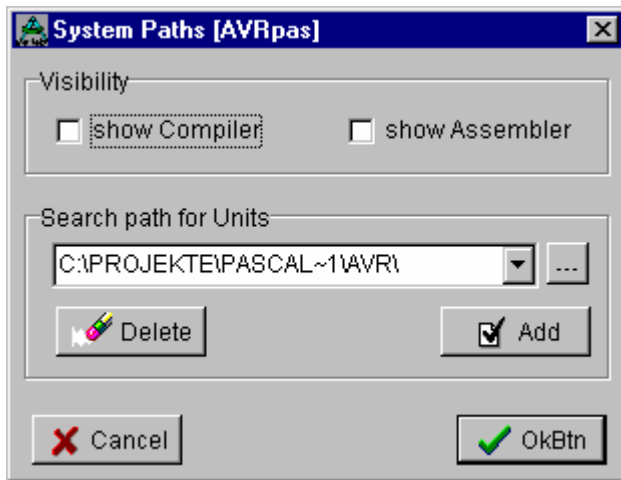
```
{ $DEFINE Test }
{ $DEFINE sample }
```

Die Suchpfade für projektbezogene Units werden in das Steuerfile „xxx.ppro“ abgelegt.



AVRco Tools

3.11.1.8 System Options



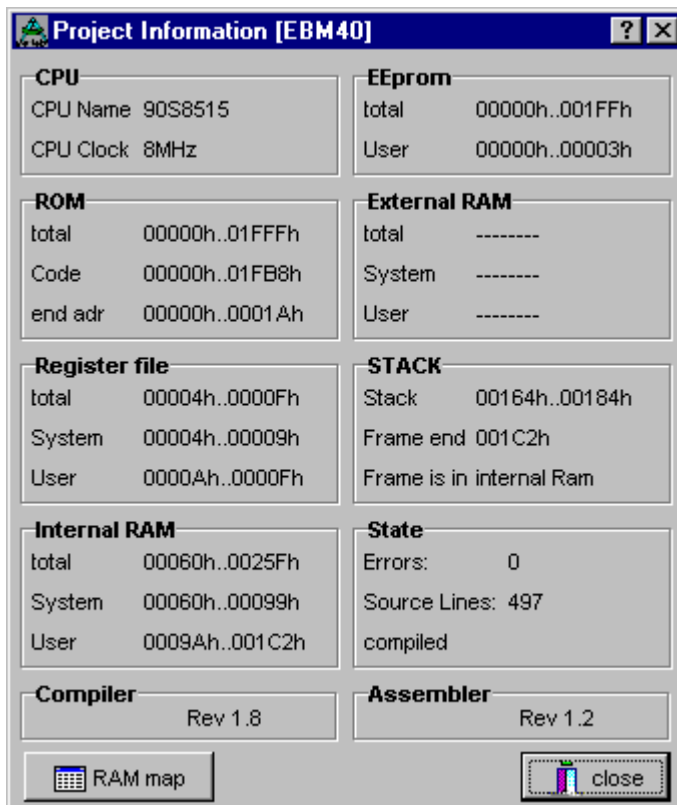
Visibility

Für alle Compiler- bzw. Assembler Läufe kann hier angegeben werden, ob das Tool sich mit einem Fenster meldet oder nicht. Ein unsichtbarer Lauf ist wesentlich schneller als ein sichtbarer.

Search path for Units

Grundsätzliche Unit-Suchpfade für alle Projekte können hier eingegeben werden. Diese systemweite Pfade werden wie auch die projektbezogenen Pfade (Dialog Project Options) in dem Projekt Steuerfile "xxx.ppr" abgelegt und stehen dem Compiler damit zur Verfügung. Das gilt für alle in den Liste befindlichen Pfade, und nicht nur der im Editfenster sichtbare Pfad.

3.11.1.9 Project Information



Alle relevanten Daten eines Projekts können mit diesem Dialog angezeigt und analysiert werden. Dadurch ist die etwas umständliche Analyse des Assembler Listings nur in Ausnahmefällen notwendig.

Mit dem Button RAM map erhält man zusätzliche Informationen über die genaue Speicherbelegung.

3.12 Controls

3.12.1 Was ist ein Control?

Wie weiter oben unter **Projekte** schon zu ersehen war, spielt die Steuer-Anweisung **Control** eine wesentliche Rolle beim Arbeiten mit einem Projekt.

Einmal mittels dem Menüpunkt **System/System Admin** und dem Dialog **System Admin** korrekt und komplett erstellt, braucht das Control in aller Regel nicht mehr zu verändert werden.

Fast jeder Punkt bzw. Zeile in diesem Dialog hat ein zugehörigen Knopf in der oberen Knopfleiste, den sogenannten SpeedButtons. Die Make-Zeile und der Simulator/Debugger können neben den jeweiligen SpeedButtons auch durch CTRL + F9 bzw. F9 aufgerufen werden.

Jede Zeile besteht aus Anweisungen, in der Regel EXE-Dateien, die von links nach rechts im Batch-Betrieb abgearbeitet werden.

Weiterhin können in einer Zeile auch Zahlen stehen, von Text und anderen Zahlen durch ein **Separatorzeichen** „|“ getrennt. Damit werden andere Zeilen innerhalb des Dialogs einbezogen. Auf Rekursionen achten!!

Ein **Beispiel**: Das Control **PICpas** ist die Steueranweisung für den E-LAB Pascal Compiler für die MicroChip PIC Prozessor Familie. Für alle zu erstellende Projekte mit PICs und dem Pascal Compiler muss nur das Control PICpas bei der Projekterstellung angegeben werden, und man kann sofort editieren, compilieren, debuggen usw.

Dieses Anweisung (Control) befindet sich im Heimatverzeichnis von PED32 und heisst PED32.ini, alle anderen schon definierte Controls sind ebenfalls in dieser Datei zu finden. Diese Datei ist wie ein Standard Windows-Ini-File aufgebaut und damit ein Textfile. Diese Datei sollte nur zu Reparaturzwecke verändert werden. PED32 pflegt diese Datei automatisch.

An dieser Stelle soll jedoch nicht verschwiegen werden, dass die Erstellung eines neuen Controls sehr sorgfältiger Überlegung bedarf. Ausserdem muss man sich mit den zugehörigen Dialogen intensiv beschäftigen. Der später daraus resultierende Automatismus bedarf etwas Vorarbeit.

Die einzelnen Punkte eines Controls sind in der Regel sog. Batch Anweisungen, die ausgeführt werden, wenn der jeweilige zugehörige SpeedButton geklickt wird. Wenn z.B. der SpeedButton *Link* geklickt wird, wird die Zeile Linker des Controls abgearbeitet. Dabei muss allerdings nicht unbedingt ein Linker aufgerufen werden, es können auch beliebige andere Operationen erfolgen, abhängig von diesem Eintrag in diesem Control.

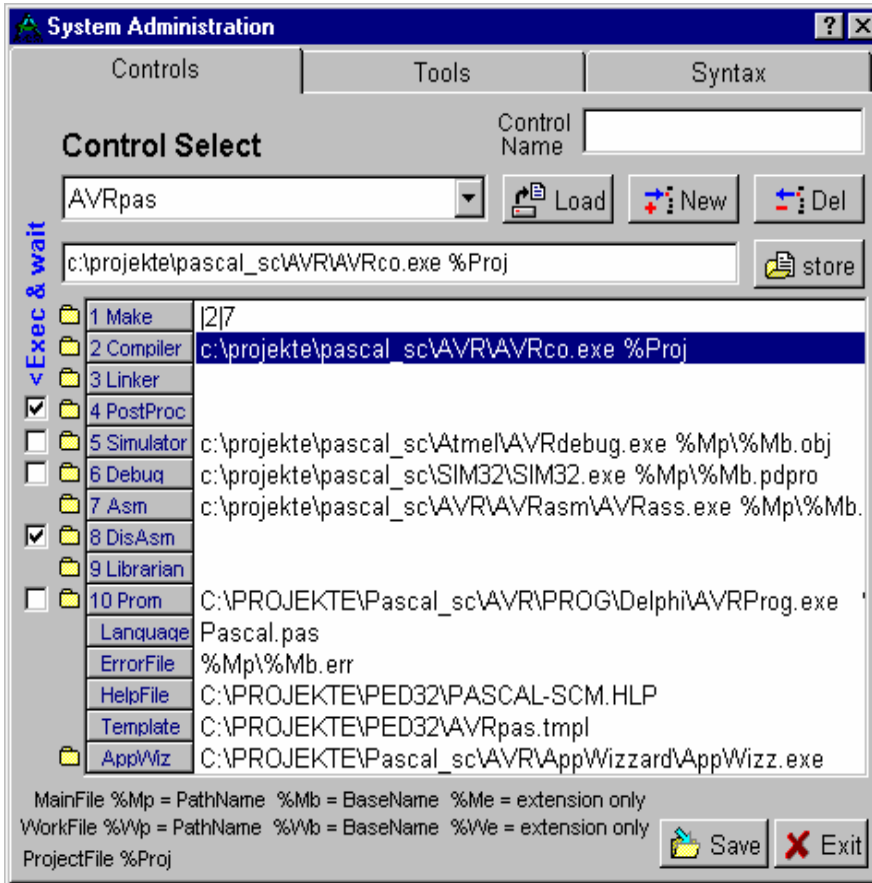
Ausser der festen Verbindung zwischen den einzelnen SpeedButtons und den Control-Einträgen besteht keine weitere Verknüpfung, d.h. jeder Control Eintrag ist eigentlich frei definierbar.

Ein Teil der Anweisungen (Make, Compiler, Linker, Assembler und Librarian) führt aber nach ihrer Ausführung einen Fehler-Check durch. Dazu wird im Arbeitsverzeichniss nach der FehlerDatei gesucht, deren Namen und Aufbau ebenfalls im jeweiligen Control abgelegt ist.

Da ein Control nur eine Fehler Datei und hier wiederum nur einen Aufbau kennt, müssen die aufgerufenen Tools in diesem Control alle die gleichen Fehlerdatei-Namen und Protokolle benutzen, ansonsten können Fehler nicht ausgewertet werden. Die E-LAB Tools wie z.B. PICco32 und PICasm32 erfüllen diese Anforderung.

3.12.2 Control Edit

Um ein vorhandenes Control zu verändern oder ein neues zu erstellen, muss der Dialog **System Admin** geöffnet werden. Das geht über den Menue-Punkt **System/System Admin**. Daraufhin wird der folgende Dialog eröffnet. Die Dialogseite **Controls** wird durch ein Click auf die jeweilige Klappe gewählt.



Soll ein vorhandenes Control **gelöscht** werden, so muss es zuerst mit der Load-Taste und anklicken im Auswahlfeld geladen werden. Das geladene Control kann jetzt mit der Delete Taste gelöscht werden.

Das Erstellen eines **neuen** Controls ist mit dem Editieren eines vorhandenen identisch, ausser dass zuerst ein Namen dafür angegeben werden muss.

Das geschieht durch ein Click auf die New-Taste. Es öffnet sich ein Eingabefeld über der **New** Taste. Hier muss der Name des neuen Controls eingegeben werden. Ist der Name komplett, wird dies durch einen Click auf die **Load** Taste bestätigt.

Das neue Control muss jetzt aus der Liste **Control Select** ausgewählt und mit der **Load** Taste geladen werden. Das weitere Vorgehen entspricht dem Editieren eines Controls und wird jetzt beschrieben.

Control Editieren.

Das zu editierende Control wird aus der Liste **Control Select** ausgewählt und mit der **Load** Taste geladen. Alle schon bearbeiteten Punkte dieses Controls sind in der Punkte Liste ersichtlich. Jeder einzelne Punkt (Batch Zeile) kann mittels einem Click auf die jeweilige Zeile angewählt werden.

Die gewählte Zeile erscheint dann im darüberliegenden **Editierfenster** zur weiteren Bearbeitung. Werden hier Veränderungen vorgenommen, so sind sie mittels der danebenliegenden **Store** Taste zu bestätigen und damit abzuspeichern. Die Veränderungen werden in der Liste sofort sichtbar.

Sollen die im Editierfeld gemachten Veränderungen verworfen werden, so darf die Store Taste nicht gedrückt werden, sondern es muss der nächste Punkt in der Liste gewählt werden. Damit sind die Änderungen hinfällig.

Die Punkte 1..10 (Make..RomSim) sind normale Batch-Zeilen, die mit dem Editierfeld verändert werden. Sie haben jeweils auch einen zugehörigen SpeedButton. Die Punkte Syntax, ErrorFile, HelpFile, Template und AppWizz eröffnen jeweils einen zusätzlichen Dialog. Diese Dialoge werden weiter unten beschrieben.

Aufbau der Batchzeile.

Eine Batch-Zeile (Punkte 1..10) kann wiederum andere Batch-Zeilen als Referenz benutzen. Der Eintrag **2|Hallo.exe|7** bedeutet, dass zuerst der Punkt 2 (Compile), dann das Programm *Hallo* und dann der Punkt 7 (Assemble) abgearbeitet wird.

Weiterhin kann das **MainFile**, das **ProjectFile**, das **Arbeitsverzeichnis**, sowie das aktuelle **Editorfenster** referenziert werden. Dazu sind die Platzhalter **%Mp**, **%Mb**, **%Me**, **%Proj**, **%Wp**, **%Wb** und **%We** bestimmt.

- %Mp** bezeichnet das Arbeitsverzeichnis
- %Mb** bezeichnet den Filenamen des MainFile ohne Extension
- %Me** bezeichnet das Extension des MainFiles (z.B. pas)
- %Wp** bezeichnet das Verzeichnis des aktuellen Editor Fensters
- %Proj** bezeichnet den Pfad und Dateinamen des Projekt Steuerfiles
- %Wb** bezeichnet den Filenamen des aktuellen Editor Fensters ohne Extension
- %We** bezeichnet das Extension des aktuellen Editor Fensters (z.B. pas)
- %Proj** bezeichnet den Projekt (File) Namen, nicht das Mainfile

Ein Beispiel zur Datei Referenzierung:

In der Annahme, dass das Arbeitsverzeichnis des Projekts "C:\Projekte\Tests" ist und das MainFile "MyProject.Pas" ist, wird folgender Eintrag erstellt:

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

Diese Batch Zeile ruft das Programm "PICco32.exe" auf und übergibt in der Kommandozeile "%Mp\%Mb.%Me" als folgenden String:

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

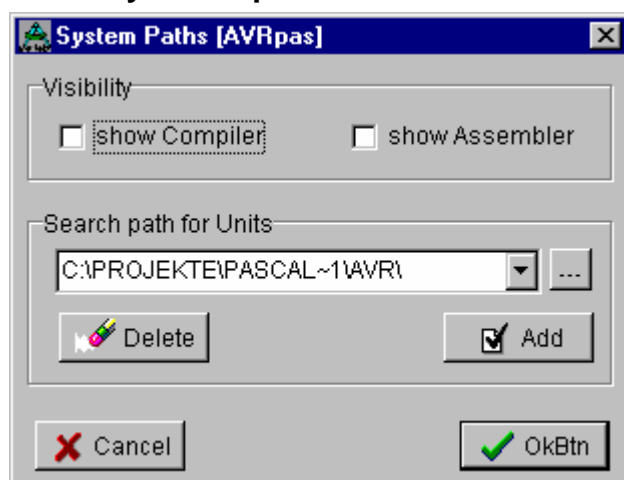
Für div. Zeilen gibt es am linken Rand eine **CheckBox**, die festlegt, ob nach dem jeweiligen Programm Aufruf die IDE warten muss, bis das aufgerufene Programm beendet wurde oder die Kontrolle gleich wieder übernehmen kann. (**Exec and Wait**) Für manche Simulatoren/Emulatoren ist ein kompletter Neustart nach jedem Compile-Vorgang sehr umständlich, weshalb die CheckBox ausgeschaltet wird und z.B. der Emulator nach dem ersten Start immer geladen bleibt.

Sind alle Punkte korrekt und möglichst komplett eingetragen, muss der Dialog mit der **Save** Taste abgespeichert werden. Alle Änderungen können an dieser Stelle noch einmal verworfen werden, indem nicht die Save Taste als Abschluss benützt wird, sondern die **Exit** Taste.

Achtung:

Keine Rekursion programmieren!! Eine Rekursion wäre, wenn in Punkt1 der Punkt2 aufgerufen würde und in Punkt2 wiederum der Punkt1. Oder auch in Punkt1 der Punkt1 selbst! Der Batch Prozessor würde bis zum Sankt-Nimmerleins-Tag oder Systemabsturz immer wechselweise Punkt1 und Punkt2 ausführen, bzw. Punkt1 ruft sich immer wieder selbst auf.

3.12.3 System Options

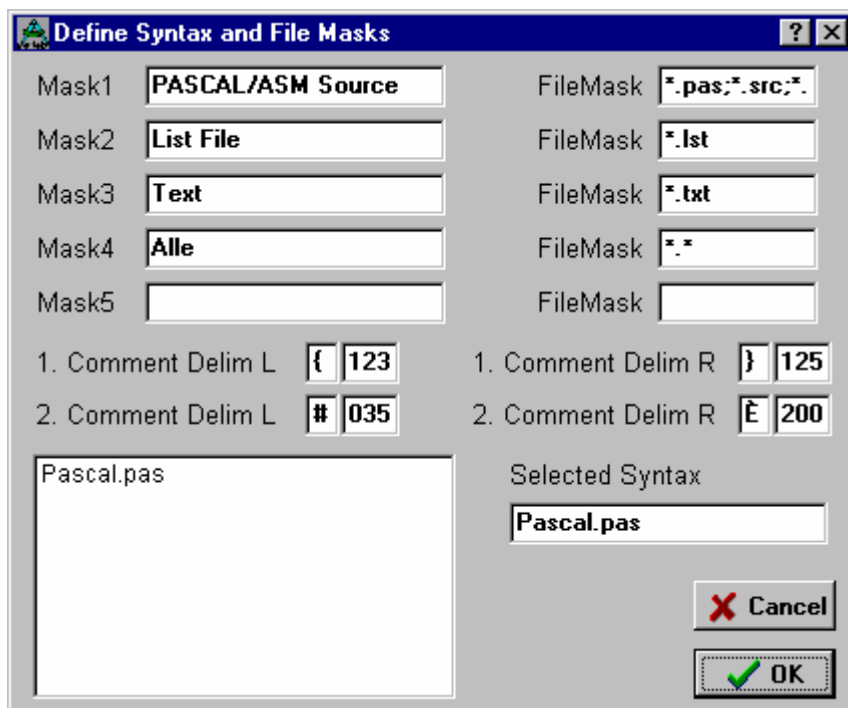


Für alle Compiler- bzw. Assembler Läufe kann hier angegeben werden, ob das Tool sich mit einem Fenster meldet oder nicht. Ein unsichtbarer Lauf ist wesentlich schneller als ein sichtbarer.

Grundsätzliche Unit-Suchpfade für alle Projekte können hier eingegeben werden. Diese systemweite Pfade werden wie auch die projektbezogenen Pfade (Dialog Project Options) in dem Projekt Steuerfile „xxx.pro“ abgelegt und stehen dem Compiler damit zur Verfügung. Das gilt für alle in den Liste befindlichen Pfade, und nicht nur der im Editfenster sichtbare Pfad.



AVRco Tools



Es können bis zu 5 **Dateimasken** definiert werden, die die IDE PED32 für die einzelnen Datei Dialoge heranzieht.

Jeder Eintrag besteht aus einem Kommentarfeld (links) und aus der Maske (rechts).

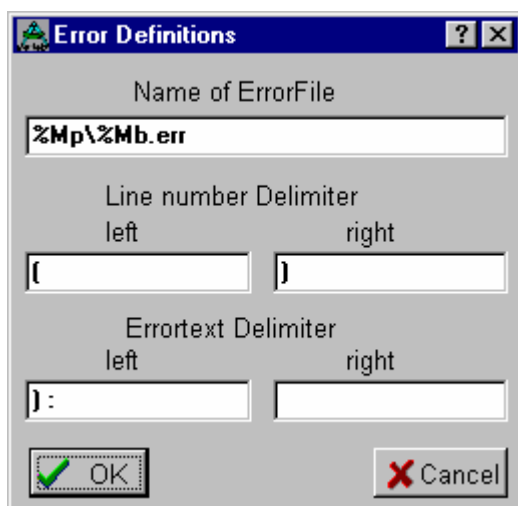
Die **Comment Delimiter** (Begrenzungen) links und rechts werden vom Editor für das Kommentar HighLight benutzt.

Unten Links wird eine Auswahl der zur Verfügung stehenden Hochsprachen (nicht Compiler) angezeigt. Durch anclicken des eines Punktes wird die "Sprache" in das Feld **Selected Lanugage** kopiert.

3.12.4 Error File define

Ein wesentlicher Punkt bei der Software Entwicklung mit einem Compiler oder Assembler ist die Fehlerauswertung. Ohne dieses Feature wird man in die Steinzeit der Software zurückgesetzt. Der wesentliche Aspekt ist aber, dass diese Auswertung nach einem Compile etc. die fehlerhafte Datei lädt, den Cursor an die fehlerhafte Stelle positioniert und, wenn möglich, auch noch den Fehlertext bzw. Beschreibung anzeigt.

Voraussetzung dafür ist allerdings, dass die Tools (Compiler etc.) ein auswertbares Fehler File erzeugen. Der Aufbau dieser Datei muss der IDE bekannt sein. Deshalb ist in untenstehendem Dialog (Unterdialog von Control Edit) die Datei und deren Aufbau zu spezifizieren.



Der Name und der Pfad des zu erwartenden Files ist im Editfeld **Name of ErrorFile** einzugeben. Hierbei sind auch die weiter oben unter Control Edit beschriebenen Platzhalter zulässig.

Der Dateiaufbau unterteilt sich pro Zeile in die Deklaration der Zeilennummer, begrenzt durch zwei Delimiters, und der Fehlerbeschreibung, wiederum begrenzt durch zwei Delimiters.

Da ein Control nur eine Fehler Datei und hier wiederum nur einen Aufbau kennt, müssen die aufgerufenen Tools in diesem Control alle die gleichen Fehlerdatei-Namen und Protokolle benutzen, ansonsten können Fehler nicht ausgewertet werden.

Die E-LAB Tools wie z.B. PICco32 und PICasm32 erfüllen diese Anforderung.

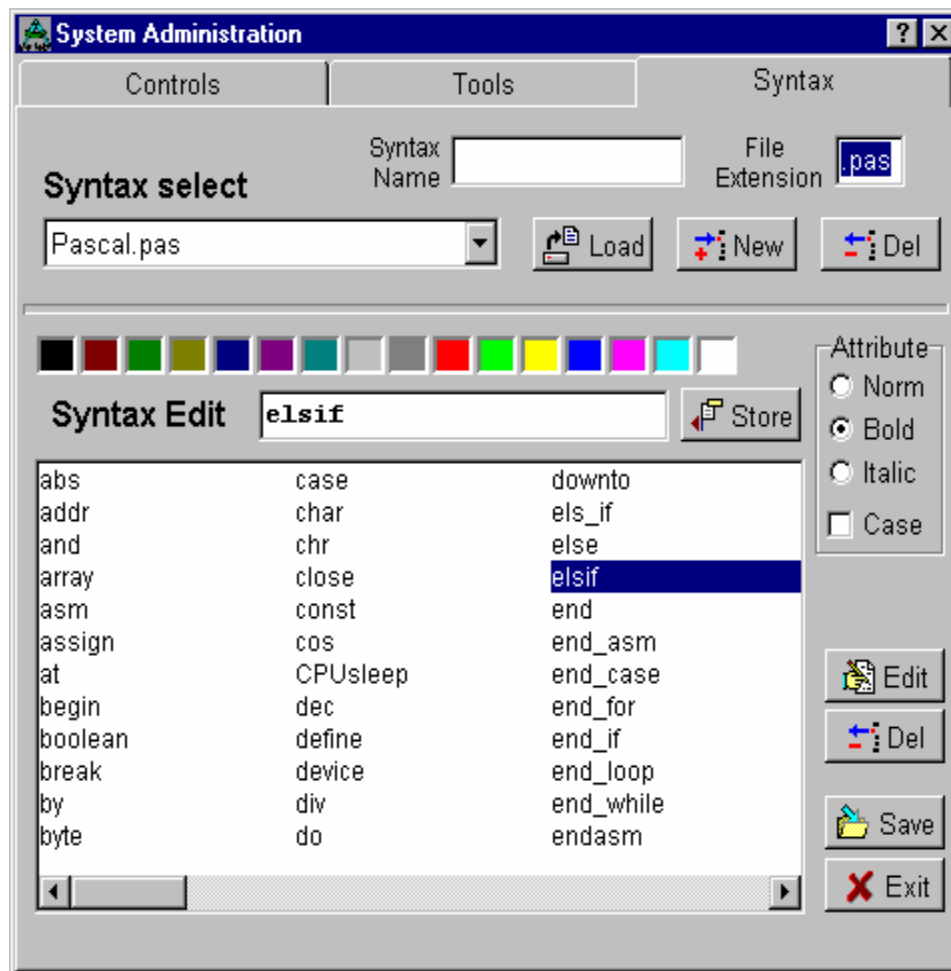
3.13 Syntax

Wessen Sprache

Die Spracheinstellung (Compilersprache wohlgemerkt) ist in der Regel auf ein bestimmtes Control bezogen und ist deshalb in einem Unterdialog von **System Admin|Control Edit** angesiedelt. Es ist aber unter Umständen, z.B. bei mehreren unterschiedlichen Pascal Compilern, sinnvoll auch mehrere Pascal „Sprachen“ zu haben. Diese müssen separat erstellt und in das jeweilige Control unter Syntax einbezogen werden.

Die Erstellung einer jeweiligen Sprach-Syntax Liste ist allerdings kein muss. Diese dient nur zur Hervorhebung von bestimmten Sprachelementen innerhalb des Editors.

3.13.1 Syntax edit



Um eine vorhandene Sprach-Syntax zu verändern oder eine neue zu erstellen, muss der Dialog **System Admin** geöffnet werden. Das geht über den Menüpunkt **System/System Admin**. Daraufhin wird der folgende Dialog eröffnet. Die Dialogseite **Syntax** wird durch ein Click auf die jeweilige Klappe gewählt.

Soll eine vorhandene Syntax **gelöscht** werden, so muss sie zuerst mit der Load-Taste und anklicken im Auswahlfeld geladen werden. Die geladene Sprache kann jetzt mit der Delete Taste gelöscht werden.

Das Erstellen einer **neuen** Sprach Syntax ist mit dem Editieren einer vorhandenen identisch, ausser dass zuerst ein Namen dafür angegeben werden muss.

Das geschieht durch ein Click auf die **New**-Taste. Es öffnet sich ein Eingabefeld über der New Taste. Hier muss der Name der neuen Sprache eingegeben werden. Ist der Name komplett, wird dies durch einen Click auf die **Load** Taste bestätigt.



AVRco Tools

Die neue Sprach Syntax muss jetzt aus der Liste **Syntax Select** ausgewählt und mit der **Load** Taste geladen werden. Das weitere Vorgehen entspricht dem Editieren einer Sprache und wird nachfolgend beschrieben.

Sprach Syntax Editieren.

Die zu editierende Sprache wird aus der Liste **Syntax Select** ausgewählt und mit der **Load** Taste geladen. Alle schon vorhandenen Syntax Wörter sind in der Liste ersichtlich. Jedes einzelne Wort kann mittels einem DoppelClick auf den jeweiligen Eintrag oder Click auf die **Edit** Taste angewählt werden.

Der gewählte Begriff erscheint dann im darüberliegenden **Editierfenster** zur weiteren Bearbeitung. Hier kann der Begriff geändert werden. Eine **Änderung** am Text erzeugt aber nicht eine Änderung in der Liste, sondern einen **neuen Begriff!** Eine Textänderung kann nur durch **Laden, Löschen und Neueingabe** des Begriffs erzielt werden. Bei einem Click auf die **Store** Taste wird abgeprüft ob der neue bzw. geänderte Begriff sich schon (identisch) in der Liste befindet. Ist das der Fall, wird der alte Begriff bestehen bleiben und nur die Textattribute übernommen. Ist der Begriff nicht in der Liste wird der komplette Text mit Attributen übernommen.

Damit ist sichergestellt, dass neue oder geänderte Begriffe, auch wenn sie sich nur in einem Buchstaben unterscheiden, auch einen neuen Begriff in der Liste erzeugen.

Attribute Änderungen über die RadioButtons **Norm, Bold, Italic** als auch über die CheckBox **Case** erzeugen genausowenig einen neuen Begriff wie Änderung der Schriftfarbe, immer vorausgesetzt, es wurden am Text selbst keine Änderungen vorgenommen.

Die Schrift Attribute Norm (normal), Bold (fett) und Italic (schräg) schliessen einander aus. Es ist immer nur ein Attribut pro Begriff möglich.

Die CheckBox **Case** stellt ein, ob der Editor auf Gross/Kleinschreibung für diesen Begriff achten soll. Ist Case aktiv, so benutzt der Editor die eingestellten Schrift Attribute und die Farbe nur, wenn das im Sourcetext gefundene Wort absolut identisch (Gross/Kleinschreibung) ist mit dem Eintrag.

Jedem Begriff kann auch eine Textfarbe zugeordnet werden. Dies geschieht durch klicken der jeweiligen Farbfläche.

Alle Attribut Änderungen werden sofort in dem Editierfeld sichtbar. Das Listenfeld dagegen enthält die Attribute zwar auch, es ist aber nicht möglich sie sichtbar zu machen.

Wurden Veränderungen vorgenommen, so sind sie mittels der danebenliegenden **Store** Taste zu bestätigen und damit abzuspeichern. Die textlichen Veränderungen werden in der Liste sofort sichtbar.

Sollen die im Editierfeld gemachten Veränderungen verworfen werden, so darf die Store Taste nicht gedlickt werden, sondern es muss der nächste Punkt in der Liste gewählt werden. Damit sind die Änderungen hinfällig.

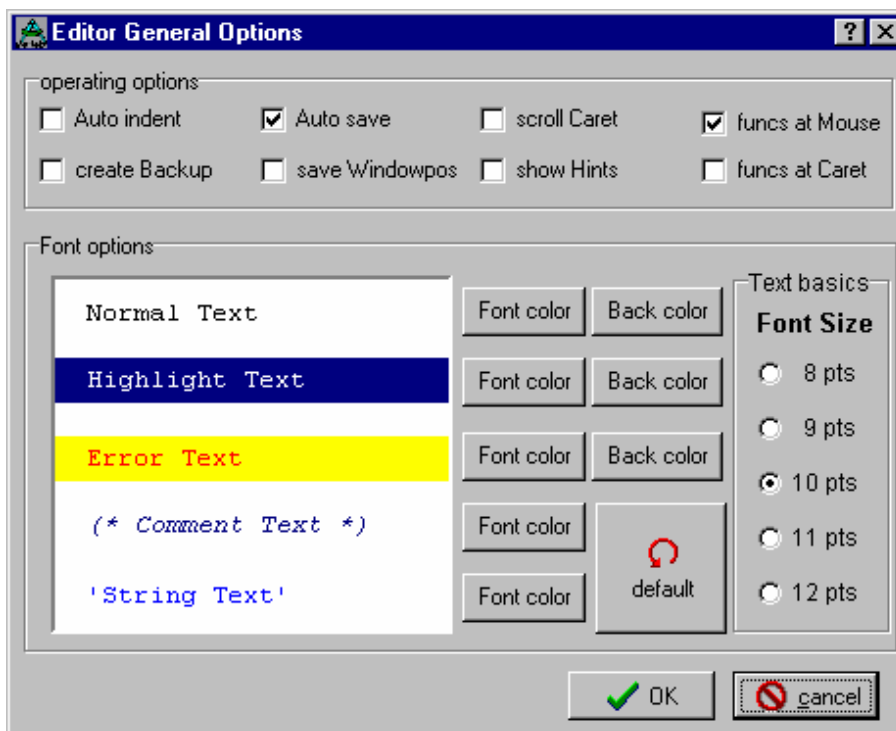
Um alle Veränderungen abzuspeichern (Datei **NewSyntax.Ing**) muss der Vorgang am Ende bzw. Verlassen des Dialogs mit **Save** abgespeichert werden.

3.14 Editor Setup

3.14.1 Fonts, Farben, Backup und Schnellhilfe

Mit diesem Dialog, aufrufbar mit dem Menüpunkt **IDE/General Options**, wird die farbliche Darstellung und die Zeichengröße des Editors eingestellt. Weiterhin wird mit **Operating Options** das Verhalten des Editors und der IDE wie z.B. Auto-Backup eingestellt.

Auto Indent	legt das automatische Einrücken fest.
Create Backup	zwingt den Editor bei einem Speichervorgang das vorhandene File auf „xxx.BAK“ umzubenennen
Auto save	erzwingt eine Sicherung aller geöffneten Dateien, wenn ein Tool aufgerufen wird.
Save Windowpos	speichert beim schliessen des Projekts die Positionen und Grösse aller Editorfenster. Bei einer Wiedereröffnung werden diese Positionen und Grössen wieder hergestellt.
Scroll Caret	setzt den Eingabe Cursor (Caret) beim Scrollen so, dass er immer sichtbar bleibt.
Show Hints	bringt alle Hilfensterchen der Buttons, Eingabefelder etc. zur Anzeige.
Show Funcs at Mouse	verbindet die Syntax-Schnillhilfe mit der Maus-Position
Show Funcs at Caret	verbindet die Syntax Schnellhilfe mit dem Caret (Eingabe Cursor).



Für die 5 mögliche Textarten ist jeweils eine eigene Textfarbe wählbar.

Für 3 Text Typen ist auch noch die jeweilige Hintergrundfarbe wählbar.

Die Textgröße wird für alle gemeinsam über Fontsize eingestellt.

Der Button **default** setzt alle Font options auf den Default bzw. Standardwert zurück.

Mit **ok** werden die Font options abgespeichert.



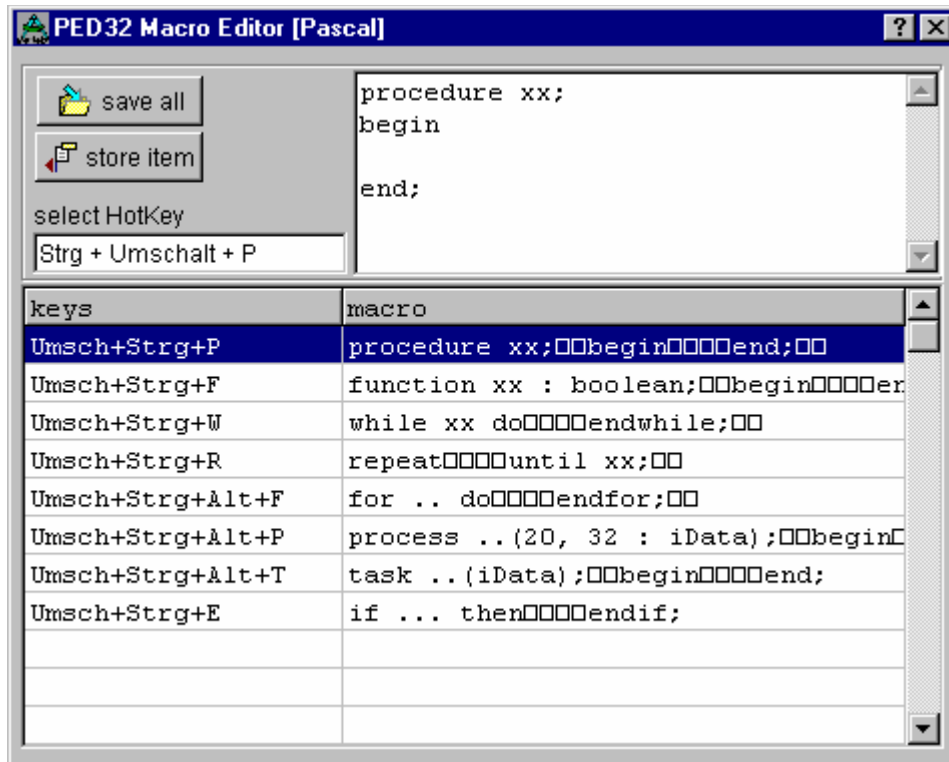
AVRco Tools

3.14.2 Zeichentabelle

Diese Tabelle dient zum Nachschlagen der Ordinal- oder Hexwerte einzelner Zeichen. Weiterhin kann man sich eine Textzeile zusammenstellen, die ins ClipBoard oder direkt in den Editor kopiert werden kann.

The screenshot shows a window titled "PED32 Character table [Courier new]". The main area is a grid of characters. The first row contains symbols like !, ", #, \$, %, &, ' () * +, -, . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?. The second row contains uppercase letters A-Z and [\] ^ _ . The third row contains lowercase letters a-z and { | } ~ □. The fourth row contains various special characters including €, □, , f „ … † ‡ ^ % Š < Œ □ □ □ □ \ / " " • - - ~ ¢ š > œ □ □ Ÿ. The fifth row contains characters like ; ¢ £ * ¥ | § ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿. The sixth row contains accented uppercase letters 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. The seventh row contains accented lowercase letters 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. Below the grid, there is a section with a large character 'E' in a box. To its right are labels 'decimal' and 'hex' above input fields containing '140' and '\$8C'. There is an 'append >>' button. To the right of these are buttons for 'Paste to editor', 'Copy to clipboard', 'clear', and 'close'. A text input field contains the character 'E'.

3.14.3 Keyboard Macros



Keyboard Macros sind Texte, die mit sog. Hot-Keys (Steuertasten) in den laufenden Text eingefügt werden können.

Dazu müssen die Macros mit diesem Dialog erstellt werden. Der Aufruf erfolgt über **IDE/Edit Keyboard Macros**.

Zum Erstellen eines neuen Macros wird ein freies Feld mit der Maus-Taste blau geklickt. Im oberen Eingabefeld kann jetzt der Macro-Text eingegeben werden. Mit einem Click auf das Feld **select HotKey** wird der Cursor/Caret auf dieses Feld platziert.

Mit der gewünschten Tasten-kombination wird der HotKey zugewiesen, z.B. „Ctrl+Shift+P“ für Procedure. Mit **Store Item** wird das Editierfeld und der HotKey in die Tabelle gespeichert. Die komplette Tabelle wird mit **save all** gespeichert.

Das Editieren eines vorhandenen Eintrags geschieht in gleicher Weise.

Beachten Sie, dass die ALT-Taste in den Fällen nicht funktioniert, wo das Hauptmenue einen unterstrichenen Buchstaben besitzt.

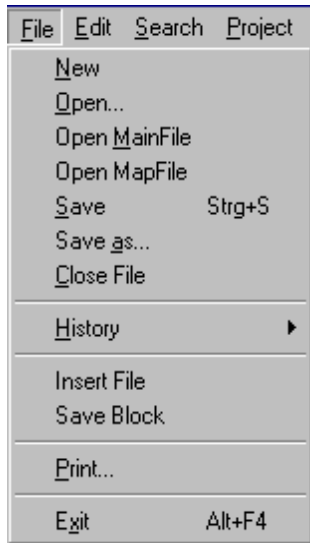
Beim Editieren kann jetzt mit der jeweiligen eingestellten Tastenfunktion (HotKey) der zugehörige Text an die Stelle des Eingabe-Cursors (Caret) eingefügt werden.



AVRco Tools

3.15 Menues

3.15.1 File Menue



- New** Eine neue Datei eröffnen.
- Open** Ruft den Datei-Open Dialog auf.
- Open Mainfile** Öffnet das MainFile des Projekts
- Open Mapfile** Öffnet das MapFile des Projekts
- Save** **Ctrl + S** Sichert das aktuelle Editorfenster in die entspr. Datei
- Save As** Speichert das aktuelle Editorfenster unter einem neuen Namen
- Close File** Schliesst das aktuelle Editorfenster mit evtl. Dateisicherung

- History** Zeigt die 10 zuletzt bearbeiteten Projekte in zeitlicher Reihenfolge.
Durch Anklicken wird das gewählte Projekt geladen.
- Insert File** Eine Datei an die aktuelle Cursor Position einlesen.
- Save Block** Den markierten Block als Datei wegschreiben.

- Print** Die aktuelle Datei (Fenster) oder Block ausdrucken.
- Exit** **Alt + F4** PED32 beenden.

3.15.1.1 History

Das Unter-Menue **History** zeigt die 10 zuletzt bearbeiteten Projekte in zeitlicher Reihenfolge auf. Dabei ist immer das zuletzt geöffnete Projekt an oberster Stelle. Hiermit ist ein Überblick über die aktuellen Projekte möglich. Durch ein Click auf denjeweiligen Eintrag wird dieser geöffnet bzw. geladen.

Der weitere Vorgang ist dann identisch wie weiter oben unter **Load Project** beschrieben.

3.15.1.2 Insert File

Mit dem Menüpunkt Insert File wird ein Dialog zur Datei-Wahl eröffnet. Wird eine Datei ausgewählt, so wird diese anschliessend an die Cursorposition (Caret) des aktuellen Editorfensters kopiert.

3.15.1.3 Save Block

Mit dem Menüpunkt Save Block wird ein Dialog zur Datei-Namen Eingabe eröffnet. Wurde ein Dateinamen eingegeben, so wird der im aktuellen Editorfenster markierte Block in diese Datei geschrieben. Der evtl. schon vorhandene Datei Inhalt geht dabei verloren.

3.15.2 Edit Menu

Edit	Search	Project	System
U <u>ndo</u>		Strg+Z	
R <u>edo</u>		Umsch+Strg+Z	
Cu <u>t</u>		Strg+X	
C <u>o</u> py		Strg+C	
P <u>a</u> ste		Strg+V	
D <u>e</u> lete		Entf	
S <u>e</u> lect All			

Undo Letzte Änderung widerrufen (Undo)

Redo Letztes Undo widerrufen (Redo)

Cut Markierter Block ausschneiden und in die Ablage kopieren

Copy Markierter Block in die Windows-Ablage kopieren

Paste Windows-Ablage an die Cursor Position kopieren

Delete Zeichen rechts vom Cursor oder ganzer Block löschen

Select All Block markieren von Datei Anfang bis Ende

3.15.3 Search Menu

Search	Project	System	IDE	Window
F <u>in</u> d...			Strg+F	
R <u>e</u> place...			Strg+R	
F <u>in</u> d <u>n</u> ext			F3	
Replace TABs by softtabs				
Replace TABs by spaces				
G <u>o</u> to line...			F4	
clear all Markers				

Find **Ctrl + F** Wort (am Cursor) suchen

Replace **Ctrl + R** Wort (am Cursor) suchen und ersetzen

Find Next **F3** Letztes Suchen/Ersetzen wiederholen

Replace Tabs by SoftTabs Harte Tabs mit Leerzeichen auffüllen

Replace Tabs by spaces Harte Tabs durch ein Leerzeichen ersetzen

Goto Line **F4** Cursor (Caret) auf Zeilennummern setzen

clear all Markers Alle Marker Buttons löschen

3.15.4 Project Menu

Project	System	IDE	Window
L <u>o</u> ad Project			
E <u>d</u> it Project			
Project I <u>n</u> formations			
Project <u>o</u> ptions {DEFINE}			

Load Project Öffnet den Projekt Dialog zum Laden eines Projekts

Edit Project Öffnet den Projekt Dialog zum Editieren eines Projekts

Project Information Öffnet den Info Dialog mit den Daten des Projekts

Project options Öffnet den Projekt Options Dialog zur Eingabe von Conditional Compile Schaltern und Suchpfade

3.15.5 System Menu

System	IDE	Window
S <u>y</u> stem <u>O</u> ptions		
S <u>y</u> stem <u>A</u> dmin		

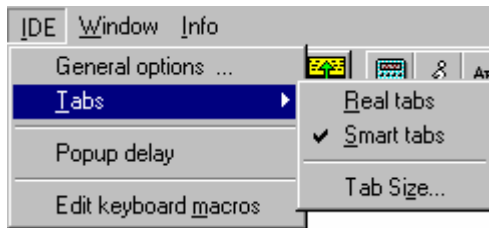
System Options Öffnet System Options Dialog (Show Compile und System Suchpfade für Units)

System Admin Öffnet den System/Control Dialog



AVRco Tools

3.15.6 IDE Menue



**General Options
Tabs**

Öffnet den Editor Options Dialog
Einstellen der TABS

PopUp Delay Verzögerungszeit für die Syntax Schnellhilfe
Edit Keyboard Macros Öffnet den Keyboard Macro Dialog

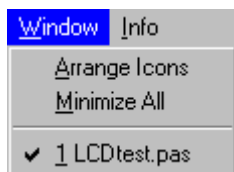
3.15.6.1 Tabs

Bei den Tabs wird grundsätzlich zwischen echten Tabs (**real Tabs**) und Smart Tabs unterschieden. Bei echten Tabs wird grundsätzlich mit der TAB-Taste ein TAB-Zeichen in den Text eingefügt. Dabei ist der Menue-Punkt Auto Indentation (automatisches Einrücken) ohne Funktion. Diese Tabs werden allerdings vom Editor mit der im Dialog Tab Size angegebenen Länge ausgeführt (Pseudo Leerzeichen).

Die **Smart Tabs** führen zu einem differenziertem Verhalten des Editors. Bei der Eingabe der Enter Taste am Zeilenende erfolgt das Einfügen einer neuen Zeile. Der Eingabe Cursor (Caret) springt dabei unter den ersten sichtbaren Buchstaben der darüberliegenden Zeile, wenn **Auto Indentation** eingeschaltet ist. Falls nicht, bleibt der Cursor am Zeilenanfang stehen. Die TAB-Taste hat bei Auto Indent eine Doppelfunktion. Wird sie gedrückt, so springt das Caret unter den nächsten Wortbeginn der darüberliegenden Zeile. Ist da kein Wort mehr vorhanden, werden soviel Leerzeichen aufgefüllt wie unter Tab Size angegeben.

Ohne **Auto Indent** werden immer die Anzahl "TAB Size" Leerzeichen aufgefüllt.

3.15.7 Window Menue



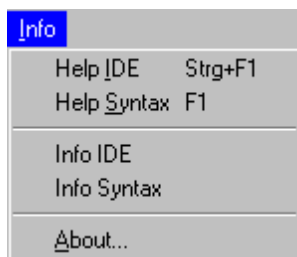
**Arrange Icons
Minimize all**

Auf Icons verkleinerte Editorfenster neu anordnen
Alle Editor Fenster auf Icongröße verkleinern

Window ...

Ein bestimmtes Editorfenster in den Vordergrund bringen

3.15.8 Info Menue



**Help IDE
Help Syntax**

Die IDE bzw. Editor Hilfe im Kontext aufrufen
Die Compiler bzw. Syntax Hilfe im Kontext aufrufen

**Info IDE
Info Syntax**

Die Übersicht der IDE bzw. Editor Hilfe aufrufen
Die Übersicht der Compiler bzw. Syntax Hilfe aufrufen

About

Anzeige der aktuellen Version etc. von PED32

Die IDE PED32 enthält zwei Helpsysteme. Das eine bezieht sich auf die IDE bzw. den Editor und das andere auf den jeweiligen Compiler bzw. Assembler.

Die Hilfe kann entweder über das obige Menue eingeschaltet werden oder über die **F1** Taste. Eine Hilfe für den Compiler kann über F1 context-bezogen erreicht werden, während die IDE-Hilfe über Ctrl + F1 erreicht wird. In Dialogen ist die IDE-Hilfe auch über F1 oder den Hilfe Button **?** zu erhalten.

3.15.8.1 Help IDE

Zu allen Dialogen und Menue Punkten ist eine Hilfe zu erhalten. Im Fall von geöffneten Dialogen ist dies mit der Taste **F1** oder **Ctrl+F1** oder dem Hilfe Button **?** zu erreichen. Die Hilfe entspricht in den wesentlichen Punkten diesem Handbuch.

3.15.8.2 Help Syntax

Soll zu einem Begriff in der Source eines Programms (z.B. **repeat**) eine Online Hilfe aufgerufen werden, so ist im Editorfenster das Caret (Einfüge Cursor) auf das Wort **repeat** zu plazieren und anschliessend die **F1** Taste zu drücken. Die Hilfe wird an der Stelle der Erklärung von **repeat** eröffnet.

3.15.8.3 Info IDE

Eine Übersicht über die Hilfethemen der IDE ist über das obige Menue durch "Info IDE" zu erhalten. Danach kann in der Hilfe wie üblich weiter verfahren werden, z.B. suchen, kopieren etc.

3.15.8.4 Info Syntax

Eine Übersicht über die Hilfethemen des Pascal Compilers ist über das obige Menue durch "Info Syntax" zu erhalten. Danach kann in der Hilfe wie üblich weiter verfahren werden, z.B. suchen, kopieren etc.

3.15.8.5 About... und Compiler Registrierung

Hier wird die aktuelle Version von der IDE/PED32 angezeigt. Zwei Registrier Felder dienen zum Freischalten des Compilers oder der Compiler von E-LAB Computers. Die Vorgehensweise liegt als Info dem entsprechenden Compiler bei.



AVRco Tools

4 Simulator / Debugger

von Gunter Baab

4.1 Allgemeines

Der Simulator ist ein unverzichtbares Werkzeug um logische Fehler in Ihren Programmen zu finden. Jedes Programm sollte zunächst mit dem Simulator getestet werden.

Der Simulator wird aus der IDE mit F9 oder einen der SpeedButtons



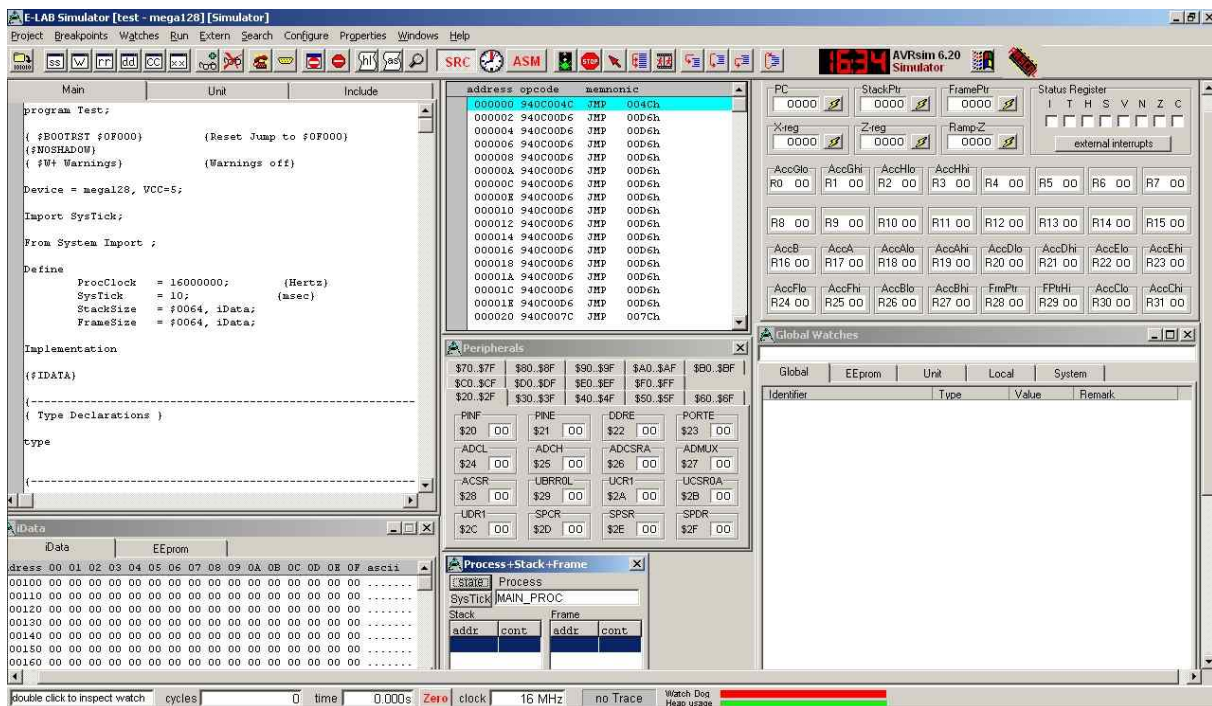
gestartet.

4.2 Übersicht - die Oberfläche

Wie üblich besteht die Oberfläche aus mehreren Bereichen:

Kopfzeile, Menuezeile, Toolbar, Arbeitsbereich und Statuszeile.

Das Erscheinungsbild des Arbeitsbereichs variiert je nach den von Ihnen ausgewählten Fenstern (Beispiele folgen).



4.2.1 die Kopf Zeile

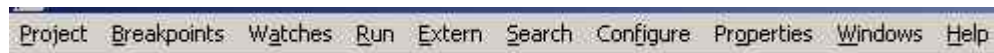
Die Kopfzeile zeigt den Projektnamen und den verwendeten Controller:



4.2.2 die Menu Zeile

Die Menuezeile enthält die in Windows üblichen PopUp Menus.

Eine ausführliche Beschreibung der einzelnen Menus finden Sie im Kapitel "die Bedienung des Simulators".



4.2.3 die Toolbar

Die Toolbar enthält viele Funktionen als "SpeedButtons". Sie kann im Menu "Windows" de-/aktiviert werden. Die SpeedButtons stellen –neben der Menuezeile- eine weitere Möglichkeit dar, die einzelnen Funktionen aufzurufen.



Sobald der Mauszeiger über einem SpeedButton steht wird die Bedeutung des Buttons in der Status Zeile angezeigt (siehe Kapitel "die Status Zeile").

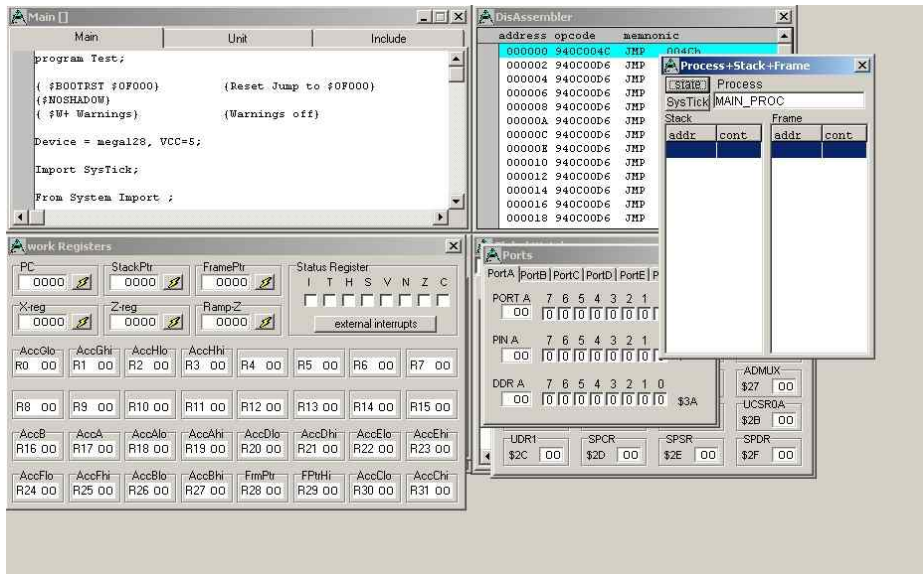
4.2.4 der Arbeitsbereich

Wie schon erwähnt, variiert das Erscheinungsbild des Arbeitsbereichs je nach den von Ihnen ausgewählten Fenster. Der Status (Größe/Position) der Fenster wird für jedes Projekt gespeichert und bei einem erneuten Simulator Start wieder hergestellt.

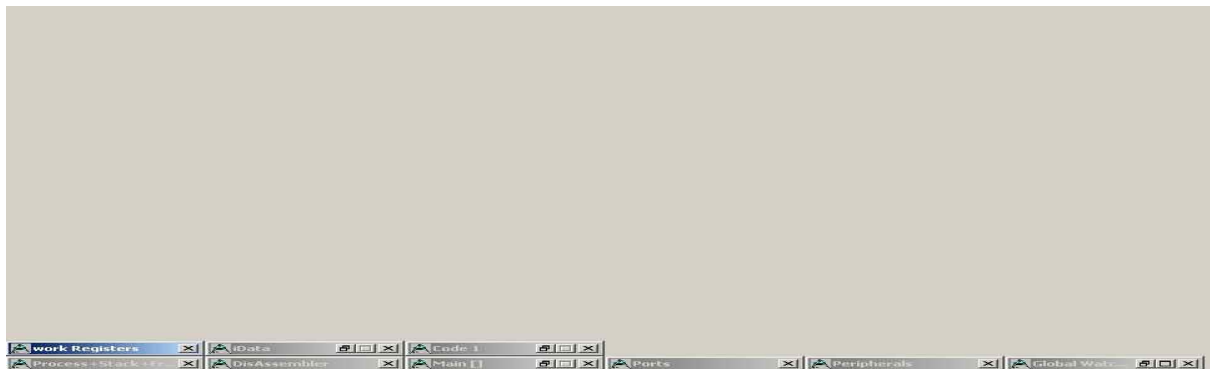
Der Simulator unterscheidet zwischen obligatorischen und optionalen Fenstern.

Die obligatorischen Fenster sind immer auf der Arbeitsoberfläche, können jedoch minimiert werden.

Im folgenden Beispiel sind alle obligatorischen Fenster geöffnet, überdecken sich jedoch teilweise:

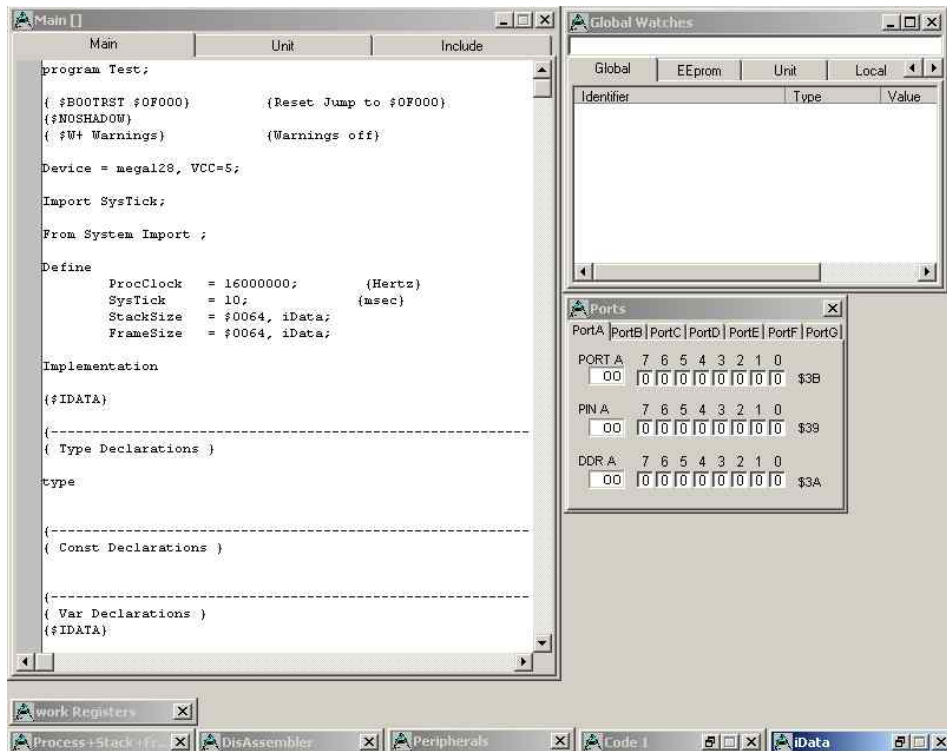


Werden alle obligatorischen Fenster minimiert und keines der optionalen Fenster geöffnet, ergibt sich folgende Darstellung:



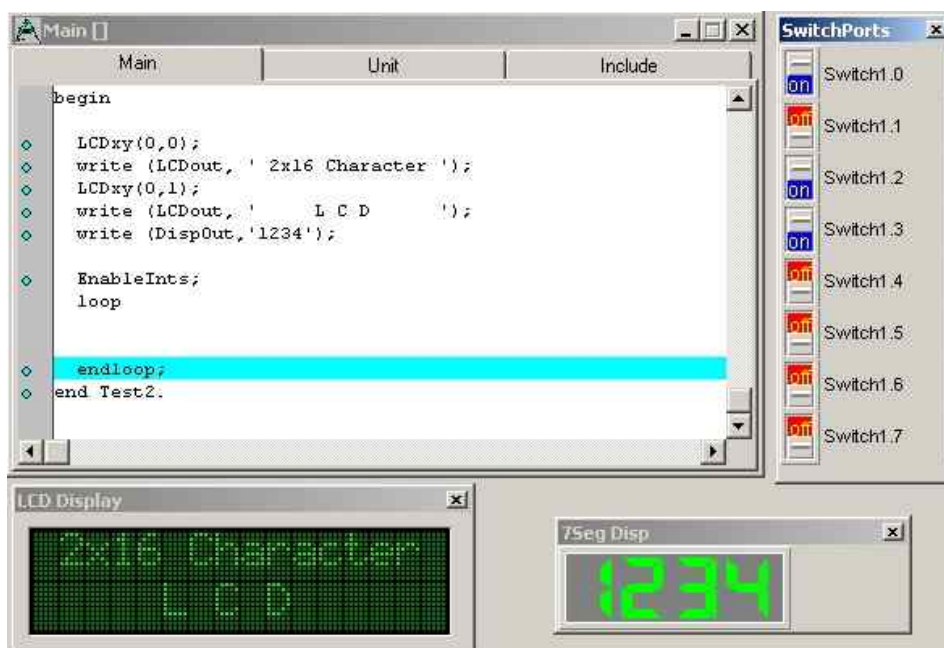
Das Minimierung der obligatorischen Fenster erfolgt über die Schaltflächen Maximiert werden die Fenster über Doppelklick auf die Titelleiste (den Namen).

Eine typische Einstellung könnte beispielsweise wie folgt aussehen:



Die optional zu öffnenden Fenster werden von der eingesetzten Hardware und den importierten Treibern des AVRco bestimmt.

Im folgenden Beispiel wurde der SwitchPort-, der LCD- und der 7-Segment Treiber importiert und die entsprechenden Fenster (zusätzlich zum Source Fenster) geöffnet.



4.2.5 die Status Zeile

Die Status Zeile befindet sich am unteren Bildschirmrand



Im Feld "Status" erhalten Sie Hinweise zu den möglichen Aktionen an der augenblicklichen Position des Mauszeigers. Beispielsweise die Bedeutung des SpeedButtons unter dem Mauszeiger.

"cycles" zeigt die Anzahl der ausgeführten Prozessor Zyklen an. Im obigen Beispiel wurde 1 Zyklus ausgeführt.

"time" zeigt die (beim realen System) benötigte Zeit. Oben 62,5 nsek., was einer Taktfrequenz von 16Mhz entspricht.

Mit "zero" können Sie "cycles" und "time" für eine neue Messung auf 0 stellen

"clock" ist die im Projekt eingestellte Taktfrequenz.

Im "Trace" Feld wird angezeigt, ob ein (Assembler- oder Pascal-) Trace mitgeschrieben wird. Dies ist im Kapitel "Menu Run" näher beschrieben.

An den Fortschrittsbalken für "Watchdog" und "Heap" können Sie erkennen, wie viel Zeit noch bleibt, den Watchdog zu triggern, bzw. wie viel Speicher auf dem Heap belegt / frei ist. Diese Funktionen wurden im obigen Beispiel nicht benutzt.

4.3 die Bedienung des Simulators

Im folgenden werden die einzelnen Menus der Menu Leiste und deren Optionen beschrieben.

4.3.1 Menu Projekt

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

Diese Optionen arbeiten analog dem "Datei" Menu im Windows / Office Standard.

4.3.1.2 Reload / Reload EEPROM

Da die (simulierten) Speicherbereiche IDATA und EEPROM zu Debug Zwecken interaktiv im Simulator geändert werden können, ist es manchmal hilfreich, die Initialbelegung wieder zu laden.

Statt den Simulator zu beenden und neu zu starten, können Sie dies einfacher hier erreichen.

"Reload EEPROM" initialisiert nur den EEPROM Bereich neu, "Reload" lädt das ganze Projekt neu.

JTAG Mode: siehe Kapitel "JTAG / OWD Debugging – UpLoad/DownLoad".



AVRco Tools

4.3.2 Menu Breakpoints

Breakpoints werden im "Source"- oder "Disassembler"- Fenster gesetzt (siehe "Menu Windows-Source / Disassembler").

Hinweise:

der Simulator "merkt" sich die Breakpoints anhand absoluter Code Adressen und diese bleiben auch bei einem Neustart erhalten. Hat sich jedoch der Source File geändert, können die Breakpoints dann an anderer Stelle stehen.

Für "Toggle Breakpoint" (oder Funktionstaste F5) und "Show List" gibt es auch SpeedButtons.

Im JTAG Mode sind max. 3 Breakpoints möglich: siehe Kapitel "JTAG / OWD Debugging – Hardware Breakpoints".

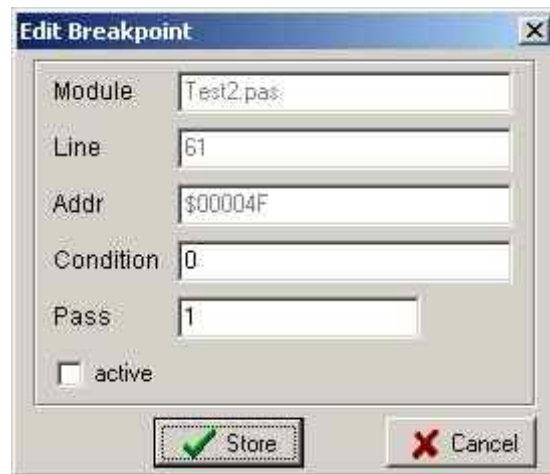
4.3.2.1 Show list

Zeigt eine Übersicht über alle derzeit gesetzten Code-Breakpoints an:

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" gibt die Source Zeile an, "Address" die Adresse im Flash.

Über "Status" können Sie gesetzte Breakpoints deaktivieren. Dies geschieht über die Schaltfläche "Edit" oder Doppelklick auf die entsprechende Zeile:



Deaktivierte Breakpoints werden im Source Fenster weiterhin angezeigt, halten den Simulator jedoch nicht mehr an.

Der Wert in "Pass" gibt an, wie oft der Breakpoint erreicht werden muß, bis der Simulator anhält. Diese Einstellung ist nicht dauerhaft und muß nach jedem "Break" neu gesetzt werden

Die weiteren Optionen sind für zukünftige Erweiterungen gedacht.

4.3.2.2 Reset all Breakpoints

Löscht alle Breakpoints. Danach sind diese in der Source nicht mehr vorhanden. Bitte nicht mit dem Status "inactive" verwechseln, wo die Breakpoints erhalten bleiben.

4.3.2.3 Stop after ..

Hier ist die Abbruchbedingung die Anzahl der ausgeführten Prozessor Zyklen. Bei Anwahl dieser Option öffnet sich ein PopUp Fenster, wo die Zyklen eingegeben werden.

Diese Option ist im JTAG Mode nicht verfügbar.

4.3.2.4 Stop on Schedule, Stop on TASK kill

Diese Optionen werden Ein- oder Ausgeschaltet und sind im Zusammenhang mit MultiTasking hilfreich.

"Stop on Schedule" hält die Simulation an, sobald der "Scheduler" einen Task- / Prozesswechsel vornimmt.

"Stop on TASK kill" stoppt die Simulation wenn der Scheduler einen Task abbrechen mußte. Dies deutet in der Regel auf ein fehlerhaftes Design der MultiTasking Systems hin.

Diese Optionen sind im JTAG Mode nicht verfügbar



AVRco Tools

4.3.2.5 Memory Write Breakpoints

Diese Option hält den Simulator an sobald eine Speicher Adresse im RAM / EEPROM (Variable) geändert wird.

Hinweis: der Simulator "merkt" sich auch diese Breakpoints anhand der Speicheradresse. Diese bleiben auch bei einem Neustart erhalten. Hat sich jedoch der Source File geändert, können die Breakpoints dann an anderer Stelle stehen.

Achtung: Diese Breakpoints werden nicht in "Show List" angezeigt !!! Dort werden nur die Code Breakpoints aufgeführt..

Die Breakpoints können mit Doppelklick editiert werden. Derzeit ist nur die Adresse relevant.

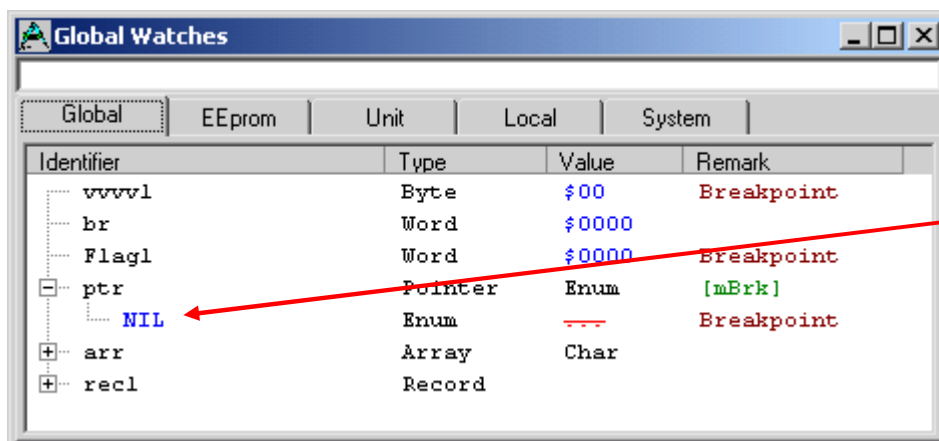
"Pass" / "Condition" ist für zukünftige Erweiterungen gedacht.

Im Menu "Watches-Add Watch" (siehe unten) besteht eine einfache Möglichkeit einen solchen Breakpoint auf die Adresse einer Variablen zu setzen.

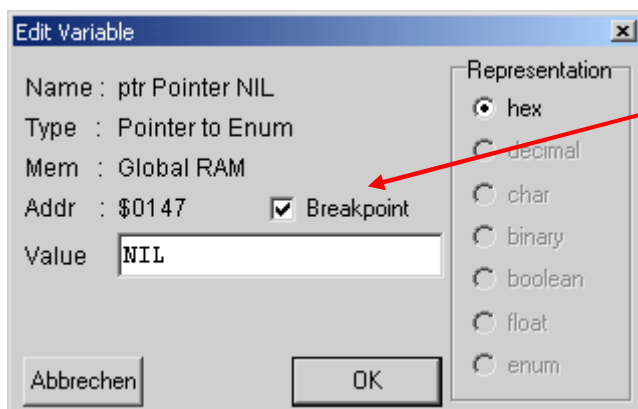
Wenn viel mit Pointern gearbeitet wird, aber auch ganz allgemein, kann es vorkommen, dass irgendwelche Variablen überschrieben werden, ohne dass man den Grund dafür erkennt. Deshalb wurde der Simulator so erweitert, dass man Breakpoints auf Variable setzen kann.

Jedesmal wenn nun eine solche Variable vom Programm verändert wird, wird das Programm angehalten und eine Meldung erscheint. Damit ist es sehr einfach möglich festzustellen, wer oder was diese Variable beschreibt bzw. verändert.

Ein Memory Breakpoint kann nur auf eine Variable gelegt werden, wenn diese schon im Watch-Fenster ist.

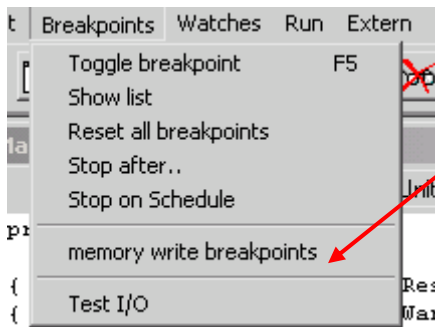


Ein Doppelklick auf das blaue Editfeld einer Watch Variablen öffnet den Watch Edit Dialog.

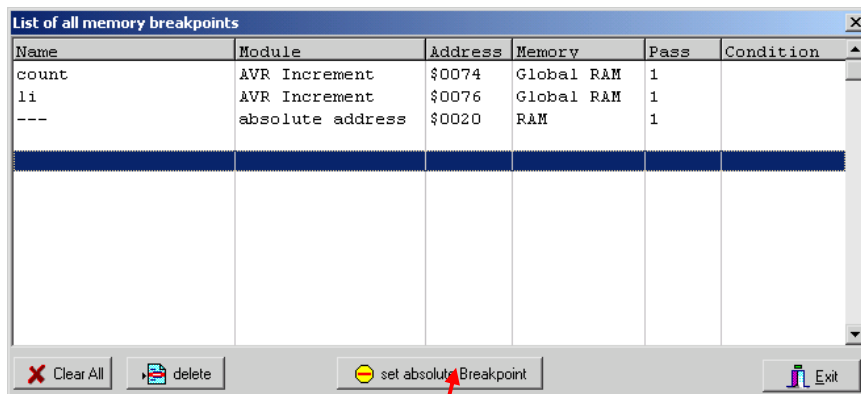


Im Watch Edit Dialog kann nun der Memory Breakpoint ein oder ausgeschaltet werden.

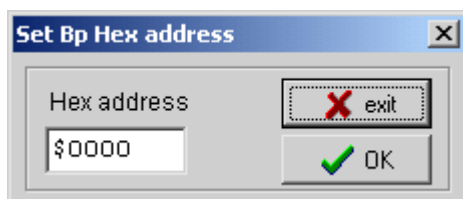
Breakpoints auf Records, Arrays und Pipes sind nicht möglich. Man kann hier jedoch mit einer Overlay Variablen arbeiten. Damit können indirekt auch diese Typen mit einbezogen werden.



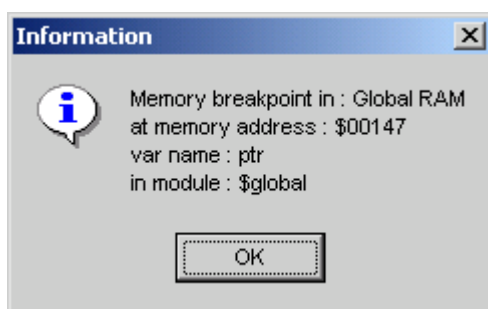
Im Breakpoints Menu kann man sich eine Übersicht aller Memory Breakpoints anzeigen lassen.



In diesem Dialog können auch Memory Breakpoints auf absolute Adressen gelegt werden.



Dann muss die gewünschte Adresse hier erfasst werden.



Wurde in einem Debug Lauf ein Schreibvorgang auf eine solche Variable erfasst, wird das Program angehalten und nebenstendes Info erscheint.



4.3.2.6 Test I/O

ist für zukünftige Erweiterungen gedacht.

4.3.3 Menu Watches

Hier können Sie Variablen auswählen, deren Wert sie im Simulator beobachten wollen und deren Darstellungsform zu definieren.

Im JTAG Mode: unbedingt Kapitel "JTAG / OWD Debugging – UpLoad/DownLoad" beachten !

4.3.3.1 Add Watch

Der Simulator listet hier alle globalen Variablen auf. Die Variablen finden Sie bei der Klappe "Globals". Unter "System" finden Sie die Systemvariablen, unter "unit" die in Units definierten (globalen) Variablen. Lokale Variablen von Prozeduren und Funktionen können *nicht* ausgewählt werden, da diese erst zur Laufzeit erzeugt werden wenn die entsprechende Prozedur oder Funktion aktiv ist.

Die Schaltfläche "Memory Break" erlaubt das einfache Setzen eines "memory write breakpoints".
"Add to WatchList" und "Remove Watch" fügt ausgewählte Variable der "WatchList" zu bzw. entfernt diese.

Hinweis:

die "WatchList" ist ein obligatorisches Fenster und im Kapitel "Windows-Watches" näher beschrieben. Mittels der Schaltfläche "Edit" können Sie den aktuellen Wert der Variable ändern, deren Darstellungsform (z.B. dezimal / hex / binär) auswählen, sowie auch einen "memory write breakpoint" definieren.

Eine weitere Möglichkeit Watches hinzuzufügen ist ein Doppelklick auf eine Variable in der Liste. Doppelklick auf eine Variable die sich bereits in der "WatchList" befindet öffnet hingegen das "Edit" Menu.

Hinweis:

Analog dazu können Sie auch direkt im "Source" Fenster (siehe Kapitel "Windows-Source") auf Variablen klicken um diese in/aus die/der "WatchList" zu nehmen oder zu editieren.

4.3.3.2 Delete all Watches

Ermöglicht das schnelle Leeren der "WatchList".

4.3.3.3 Popup Raw Display

Dies ist ein Auswahlpunkt. Falls aktiviert wird bei komplexen Variablen (Strukturen, Arrays) zusätzlich ein hexadezimaler Speicherauszug in der "WatchList" angezeigt (durch Doppelklick auf die entsprechende Variable).

Im JTAG Mode wird dadurch auch die entsprechende Variable aktualisiert.

4.3.3.4 default Watch representation

In einer Matrix können für sämtliche Variablen Typen (byte, word, integer, ...) die sinnvoll möglichen Default Darstellungsformen (hex, dezimal, binär, ...) eingestellt werden. Die individuelle Auswahl der Darstellungsform geschieht im "Edit" Menu (siehe Kapitel "Watches-Add Watch" und " Windows-Watches").

4.3.4 Menu Run

Hier wird die Simulation (in der gewünschten Art und Weise) gestartet. Weiterhin können diverse Einstellungen für die Simulationsarten getroffen werden und es wird das Aufzeichnen von Traces gesteuert. Traces sind im JTAG Mode *nicht* möglich.

Für den Großteil dieser wichtigen Funktionen gibt es auch einen entsprechenden SpeedButton.

Im Einzelschritt Betrieb bestimmen die mittleren SpeedButtons



ob Pascal (SRC – Source Step) oder Assembler (ASM – Assembler Step) gemeint ist.

Im JTAG / OWD Mode ist unbedingt das Kapitel "JTAG / OWD Debugging" zu beachten !

4.3.4.1 Reset processor Ctrl+F2

setzt den Programmzähler auf 0, sodaß die Simulation von vorne beginnt. Entspricht einem Hardware-Reset im realen System. Sind für Prozessor interne Register Initialwerte vorgegeben, werden diese geladen.

4.3.4.2 Go F9

Startet die Simulation mit maximaler Geschwindigkeit.



AVRco Tools

4.3.4.3 Goto cursor pos F4

Startet die Simulation mit maximaler Geschwindigkeit bis die Cursor Position im Source Fenster erreicht wird bzw. vorher eine Break Bedingung erreicht wird.

4.3.4.4 Stop simulation F2

Hält die Simulation an

4.3.4.5 Step into F7

Führt einen Einzelschritt aus. Auch Funktionen / Prozeduren / Schleifen werden im Einzelschritt ausgeführt.

4.3.4.6 Step over F8

Führt einen Einzelschritt aus. Funktionen / Prozeduren / Schleifen werden *nicht* in Einzelschritten durchlaufen (d.h. in einem Schritt ausgeführt).

4.3.4.7 Step out F6

Führt die aktuelle Funktion / Prozedur / Schleife mit maximaler Geschwindigkeit aus und stoppt danach (sofern vorher keine Break Bedingung auftritt).

4.3.4.8 Multiple Steps Shift+F9

Führt eine vorgegebene Anzahl von Schritten in maximaler Geschwindigkeit aus. Siehe unten: "Multiple step value".

4.3.4.9 Animate Ctrl+F9

Führt die Simulation in Einzelschritten und verlangsamer Geschwindigkeit aus. Siehe unten: "Animation speed".

4.3.4.10 Multiple step value

Hier wird die Anzahl der Schritte für "Multiple Steps" eingestellt.

4.3.4.11 Animation speed

Hier wird die Geschwindigkeit für "Animate" eingestellt.

4.3.4.12 Enable Trace ASM / Enable Trace HLL

Es kann *entweder* ein Trace auf Assembler-Ebene *oder* ein Trace auf Pascal Ebene mitgeschrieben werden.

Hinweise:

den Trace können Sie sich im Menu "Windows-view Trace" ansehen.

Im Assembler Trace werden zusätzlich zu den Assembler Zeilen auch die Pascal Statements angezeigt. Sie können während einer Trace Session beliebig zw. ASM, HLL und "disable" umschalten.

Dies Funktionen sind im JTAG Mode nicht verfügbar.

4.3.4.13 Clear Trace buffer

Löscht den Trace Speicher, der ansonsten immer fortgeschrieben wird.

Diese Funktion ist im JTAG Mode nicht verfügbar.

4.3.4.14 Call Stack Ctrl+F3

Zeigt die augenblickliche Verschachtelung (die Rücksprung - Prozedur / Funktions Namen) auf dem Stack an.

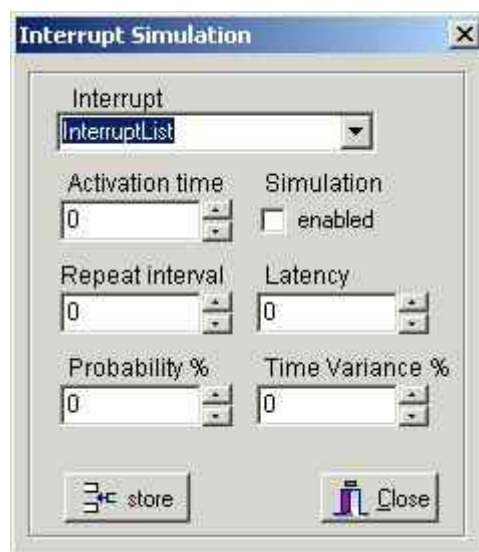
Diese Funktion ist im JTAG Mode nicht verfügbar.

4.3.5 Menu Extern

Hier sind alle Prozessor externen Geräte (im Gegensatz zu z.B. den Ports) zu finden, die simuliert werden können. Außer dem Unterpunkt "Interrupts" sind diese auch im Menu "Windows" zu finden und werden dort beschrieben.

4.3.5.1 Interrupts

Hier können die externen Interrupts (an den Prozessor Anschlüssen INT0, INT1, ...) simuliert werden.



Diese Funktion ist derzeit noch nicht verfügbar.



AVRco Tools

4.3.6 Menu Search

Wird zusammen mit den Fenstern für "Source", "iData" (dem RAM Speicher), "EEProm" oder "Code" (dem Flash Speicher) verwendet und maximiert diese wenn nötig. Es kann nach bestimmten Stellen (Adressen), Hex-Mustern oder Text gesucht werden.

4.3.6.1 Show Code at..

Anzeigen des Flash-Inhalts bei einer vorgegebenen Adresse. Maximiert ggf. das "Code" Fenster.

4.3.6.2 Show Data at..

Anzeigen des RAM- oder EEPROM Inhalts bei einer vorgegebenen Adresse. Maximiert ggf. das "iData / EEPROM" Fenster.

4.3.6.3 Search Code hex pattern..

Sucht nach einem Hex-Muster im Code Bereich. Maximiert ggf. das "Code" Fenster.

4.3.6.4 Search Data hex pattern..

Sucht nach einem Hex-Muster im Daten Bereich. Maximiert ggf. das "iData / EEPROM" Fenster

4.3.6.5 Search in Source F3

Erlaubt die Volltextsuche im Quellfile. Dabei stehen die üblichen Optionen (vorwärts, rückwärts, ganzes Wort etc. zur Verfügung).

4.3.7 Menu Configure

Hier werden einige Einstellungen für die Oberfläche und die Simulation getroffen

4.3.7.1 Show Hints

Schaltet die PopUp Hilfe unter dem Mauszeiger ein oder aus. Diese Hilfetexte sind auch im Status Feld links unten zu sehen.

4.3.7.2 Save as default

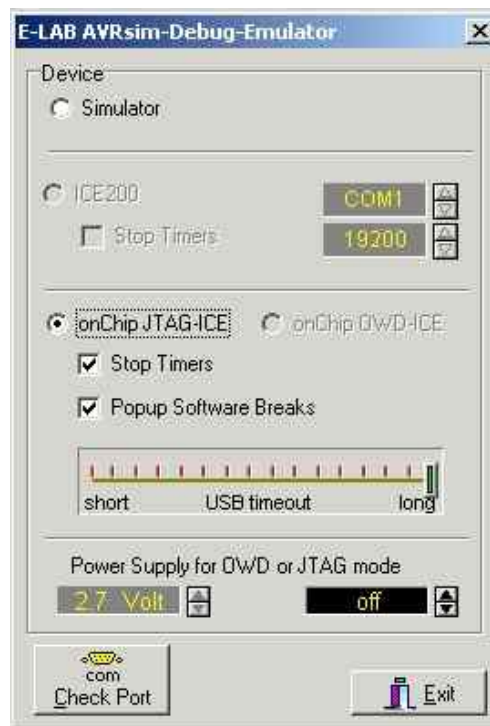
Betrifft die Position und Einstellung (maximiert / minimiert) der Fenster.

4.3.7.3 Config with default

Stellt die Position und Einstellung (maximiert / minimiert) der Fenster wieder her.

4.3.7.4 COMport [ICE..Monitor]

Hier wird die ggf. angeschlossene Debug Hardware konfiguriert.



"Simulator": es ist keine Hardware angeschlossen, die ein Debuggen unterstützt.

"ICE200": es ist ein Atmel ICE200 Emulator angeschlossen. "Stop Timers" hält die Timer des Controllers an, sobald die Simulation gestoppt wird – z.B. wenn ein Breakpoint erreicht wird. Der ICE200 kommuniziert über die serielle Schnittstelle mit dem PC. Diese kann hier ebenfalls konfiguriert werden.

"onChip JTAG-ICE", "onChip OWD-ICE": es werden die Controller internen Debug Hilfen benutzt. JTAG ist das bereits implementierte standardisierte JTAG Debug Interface (JTAG = "Joint Test Action Group"). "OWD" ist für das zukünftige "One Wire Debug" Interface vorgesehen. Stop Timers hält die Timer des Controllers an, sobald die Simulation gestoppt wird – z.B. wenn ein Breakpoint erreicht wird. "Popup Software Breaks" zeigt ein Info-Fenster an wenn der Debugger einen Software Breakpoint erreicht. "Check Port" muß bei Hardware unterstütztem Debuggen zwingend ausgeführt werden, da dadurch auch der Debugger initialisiert wird.

Die JTAG fähigen Programmer kommunizieren über die USB Schnittstelle mit dem PC. Mit dem USB Timeout kann die Geschwindigkeit (vorsichtig!) etwas optimiert werden. "Power Supply for OWD or JTAG mode" bestimmt, ob der Programmer das Zielsystem mit Spannung versorgen soll.

Der Button "Check Port" durchsucht die verfügbaren Schnittstellen nach einem Programmer.

Da der Programmer dadurch auch initialisiert wird ist diese Schaltfläche zwingend anzuklicken.



AVRco Tools

4.3.8 Menu Properties

Da der Simulator nicht die Geschwindigkeit eines realen Systems erreichen kann, wirken sich einige Funktionen störend aus. Die folgenden Schalter optimieren das Zeitverhalten.

4.3.8.1 Short mDelay

"mDelays" sind (im realen System) Warteschleifen im Millisekunden Bereich. Da diese im Simulator ggf. sehr lange dauern, können sie hier für die Simulation verkürzt werden.

4.3.8.2 Fast RTC

Damit wird der Takt der Echtzeituhr beschleunigt um die Funktionen des RTC Treibers realistisch simulieren zu können.

4.3.8.3 Short Beep

Passt die Dauer der Töne des Sound Treibers an die langsamere Geschwindigkeit des Simulators an, sodaß diese realistisch klingen.

4.3.9 Menu Windows

Hier können die optionalen Fenster eingeblendet, sowie Informationen zum MultiTasking System abgerufen werden. Optionale Menüpunkte können nur ausgewählt werden, wenn der entsprechende Treiber eingebunden ist. Ansonsten sind sie "ausgegraut".

4.3.9.1 Toobar

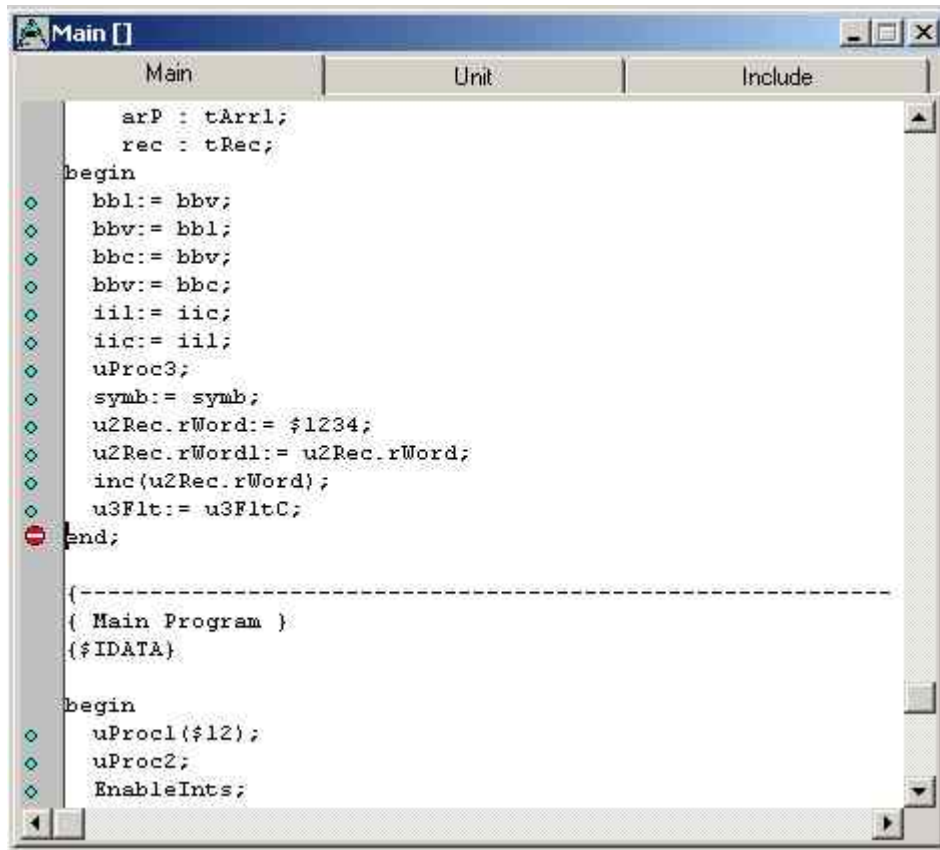
Schaltet die Toolbar mit den Speed Buttons ein oder aus.

4.3.9.2 Arrange icons

Ordnet die Icons der geschlossenen obligatorischen Fenster am unteren Bildschirmrand. Die optionalen Fenster werden *nicht* zu Icons verkleinert, sondern ggf. komplett geschlossen.

4.3.9.3 Source

Ist ein obligatorisches Fenster.



```

Main [
Main      Unit      Include
-----
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" zeigt immer Ihre Pascal Source. Die Fenster "Unit" und "Include" sind zunächst leer. Durch Rechtsklick innerhalb einem dieser Fenster kann die anzuzeigende Unit bzw. das anzuzeigende Include File ausgewählt werden. Voraussetzung ist, daß die Units als Source File (nicht nur vorkompiliert als .PCU) vorliegt. Wird während eines Debug Laufs in eine Unit oder Include File verzweigt, wird dieses File automatisch in das entsprechende Fenster geladen und angezeigt.

Weiterhin bietet ein Rechtsklick die Optionen "Find Text" um zu suchen.

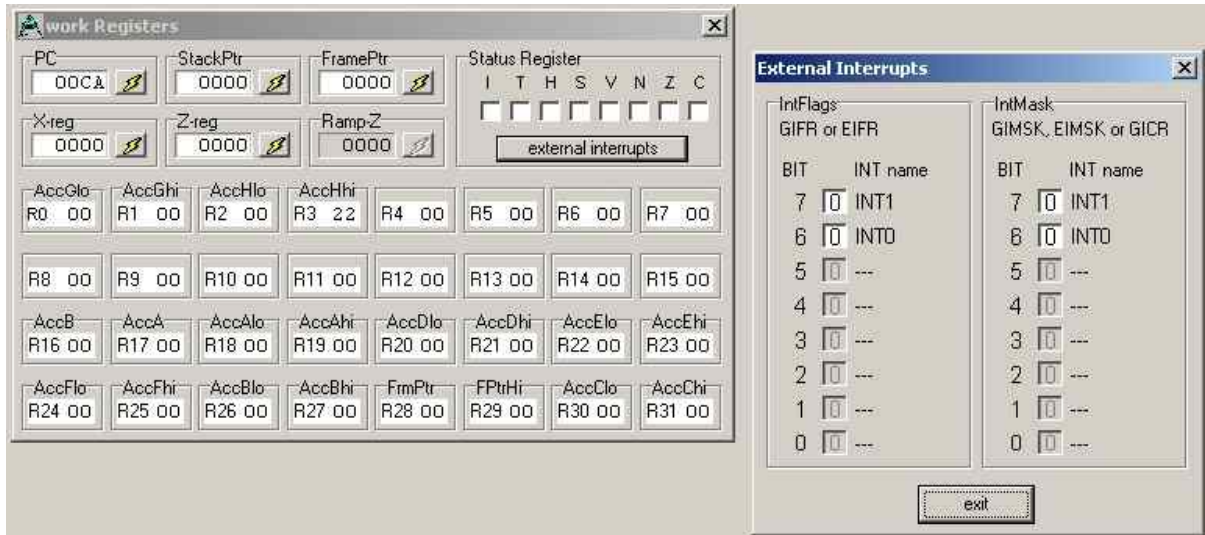
Steht der Textcursor auf einem ausführbaren Statement, sind weiterhin die Optionen "Set PC to line pos" (damit führt der nächste Single Step diesen Befehl aus) und "Goto line pos" (damit wird das Programm bis zur aktuellen Cursorposition ausgeführt) verfügbar.

Klick auf die blauen Kreise (in der Spalte ganz links) setzt / löscht dort einen Breakpoint.

Doppelklick auf eine Variable übernimmt diese ins "Watch Fenster".

4.3.9.4 Work Registers

Ist ein obligatorisches Fenster.

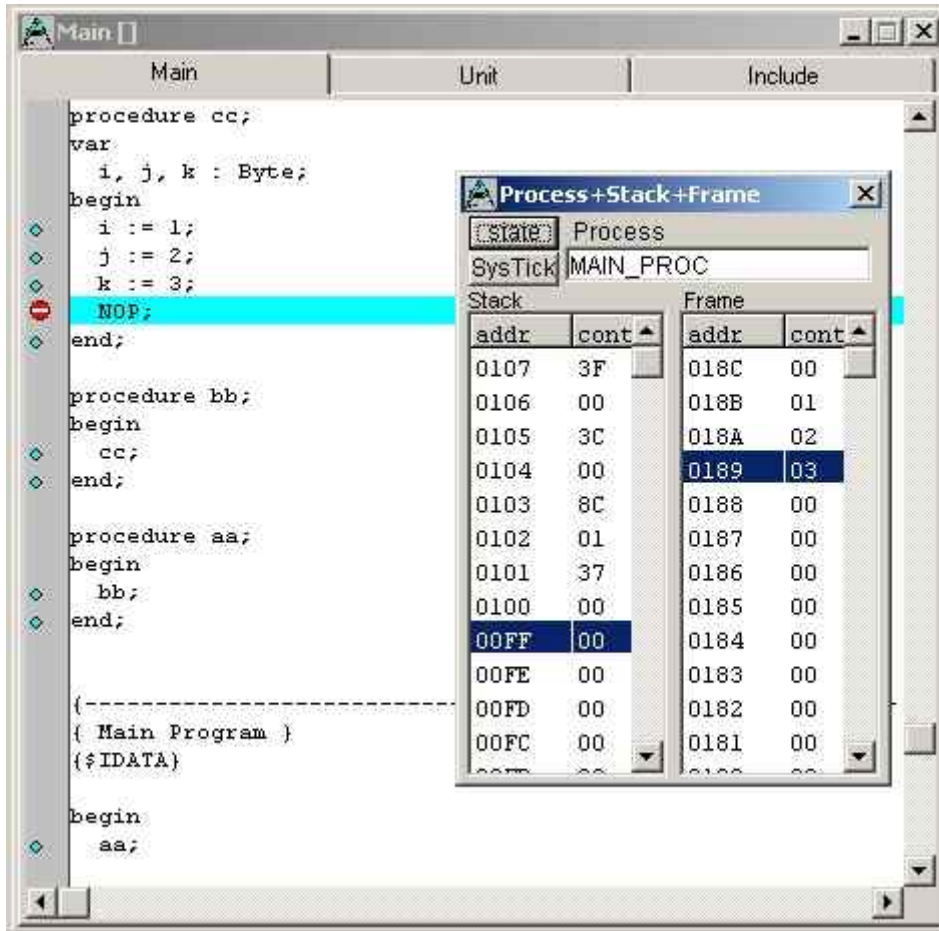


Dieses Fenster ist für das Debuggen von Assembler Routinen gedacht. Es zeigt die Prozessor internen Register, die symbolischen Namen die der AVRco dafür benutzt, sowie die externen Interrupts. Der Register Inhalt kann hier auch geändert werden.

Beachten Sie hierzu auch unbedingt das Kapitel "JTAG / OWD Debugging – UpLoad / DownLoad".

4.3.9.5 Processes

Ist ein obligatorisches Fenster.



Im obigen Beispiel sind drei Prozeduren verschachtelt. Die tiefste deklariert drei lokale Byte Variablen. Sie sehen Sie die Stack und Frame Belegung bei Erreichen des Breakpoints. Die Position des Stack / Frame Pointers ist jeweils blau hinterlegt.

Die Schaltflächen "state" und "SysTick" sind für das MultiTasking interessant und in den Kapiteln "Windows – view Process states" und "Windows – SysTick / Scheduler Timings" erläutert.

4.3.9.6 Disassembler

Ist ein obligatorisches Fenster.

Disassembler zeigt den Assembler Code. Die aktuelle Position des Programm Counters ist hellblau unterlegt, die Position des Text Cursors in der Pascal Source dunkelblau. Durch klicken einer (Code generierenden) Source Zeile zeigt das Disassembler Fenster den entsprechenden Code. Analog zeigt ein Klick im Assembler Code die entsprechende Source Zeile im Source Fenster.

Durch Rechtsklick können Sie den Programm Counter auf die Adresse des Befehls unter dem Maus Zeiger setzen.



AVRco Tools

4.3.9.7 Code memory

Ist ein obligatorisches Fenster. Zeigt den Hexdump des Assembler Codes. Mit der rechten Maustaste können verschiedene Darstellungsweisen gewählt, sowie gesucht werden.

4.3.9.8 Data memory

Ist ein obligatorisches Fenster. Zeigt Hexdumps des Ram- und des EEPROM Bereichs. Die Daten-bereiche können hier auch geändert werden.

Mit der rechten Maustaste sind Suchfunktionen erreichbar. Im JTAG / OWD Mode sind weitere Optionen zum Upload / Download der Datenbereiche verfügbar.

4.3.9.9 Watches

Ist ein obligatorisches Fenster. Hier kann der aktuelle Wert von Variablen beobachtet werden.

Im JTAG / OWD Mode ist unbedingt das Kapitel "JTAG / OWD Debugging" zu beachten !

Durch Doppelklick auf den Wert einer Variable kann dieser geändert, sowie die Darstellungsform (dezimal / hex ...) gewählt werden.

Zur besseren Übersicht sind die zu beobachtenden Variablen nach verschiedenen Kriterien unter den Klappen "Global", "EEProm", "Unit", "Local" und "System" zusammengefaßt.

Bei "Local" kann durch Rechtsklick im entsprechenden Fenster eingestellt werden, ob bei einem Stop im JTAG Mode in einer Prozedur / Funktion die lokalen Variablen angezeigt werden sollen.

Mittels "check Frame and Stack" werden bei einem Fenster Update im JTAG Mode zusätzlich der Stack- und Frame-Pointer hochgeladen und auf Bereichsüberschreitungen geprüft.

Ein Rechtsklick in den anderen Fenstern öffnet ein Menu um Watches hinzuzufügen oder zu entfernen.

Weiterhin können hier auch Memory Breakpoints gesetzt werden. Die Option "Refresh all Watches ist im JTAG / OWD Mode sehr wichtig.

4.3.9.10 Ports

Ist ein obligatorisches Fenster. Zeigt die beim Controller verfügbaren Port Register (PortX, PinX, DDRx) mit ihren symbolischen Namen an und erlaubt diese zu modifizieren.

4.3.9.11 Peripherals

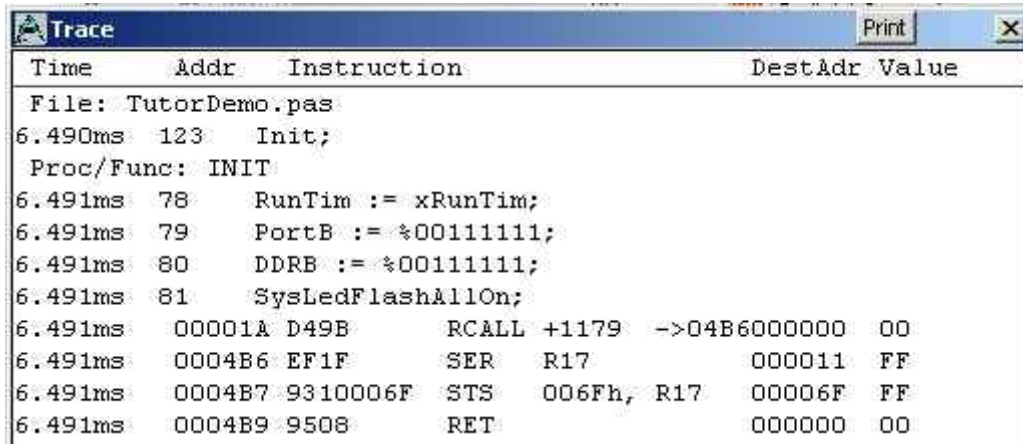
Ist ein obligatorisches Fenster. Zeigt alle Controller internen Register (geordnet nach Adressen) an und ermöglicht es diese zu ändern, soweit sie nicht nur gelesen werden können. Dieses Fenster ist für das Debuggen von Assembler Routinen nützlich.

4.3.9.12 View Trace

Zeigt die im Menu "Run" zu startenden Traces (SRC/HLL) an.

Im folgenden Beispiel wurden die Zeilen 78, 79, 80 als HLL Trace und die Zeile 81 als ASM Trace mitgeschrieben.

Im JTAG Mode sind keine Traces möglich.




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

Dient zur Beurteilung des MultiTasking Verhaltens.

Insbesondere können Sie hier die Prioritäten der Tasks / Prozesse und deren maximalen Bedarf an Stack und Frame Speicher ("StkPeak" / "FrmPeak") sehen. Dazu sollte die Applikation einige Zeit simuliert werden und man sollte möglichst alle Ereignisse (I/Os, externe Interrupts etc.) simulieren.

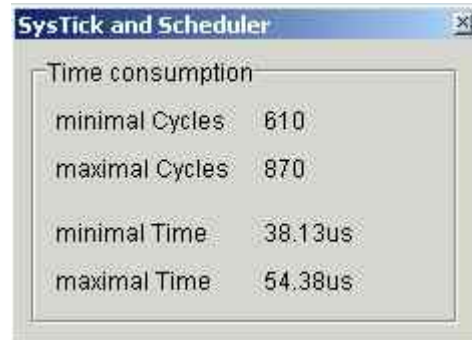
Diese Funktion ist im JTAG Mode *nicht* verfügbar.



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

Zur Beurteilung des Multitasking Verhaltens kann hier der Ressourcen Verbrauch des Schedulers angezeigt werden. Diese Funktion ist im JTAG Mode *nicht* verfügbar.



4.3.9.15 Terminal I/O

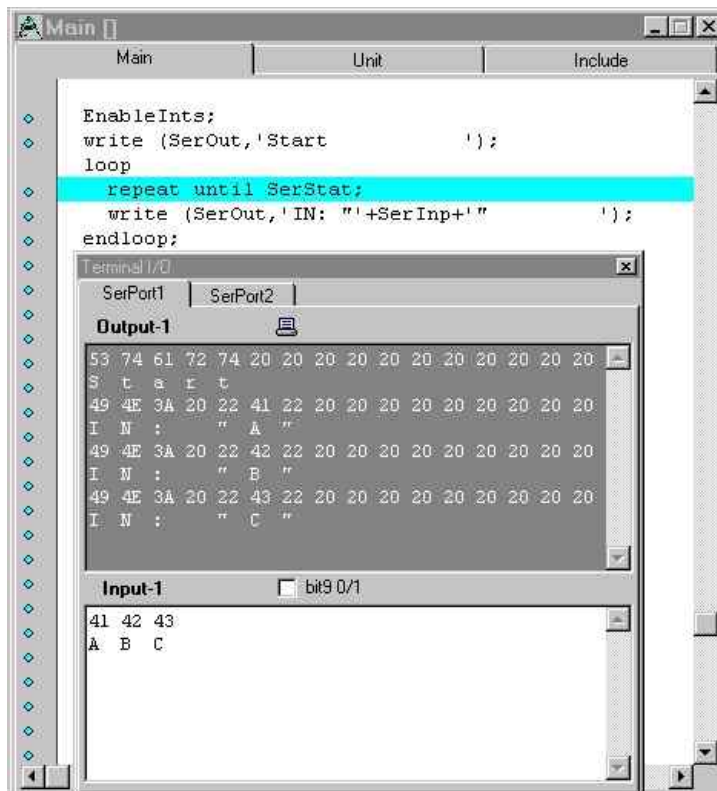
Import: SerPort / SerPort2

Ist ein optionales Fenster zum Debuggen der Datenübertragung über die seriellen Schnittstelle(n).

Die moderneren Controller besitzen USARTs, die auch 9 Datenbits (z.B. für Multi Prozessor Kommunikation) zur Verfügung stellen. Der Wert des 9. Bit ist dann im Input Bereich einstellbar.

Hexadezimale Inputs sind mit Ctrl + 0...9, A...F möglich.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.



Im oberen (grauen) Fenster sehen Sie die Ausgaben von "SerOut", im unteren Fenster die Benutzer Eingaben, die mittels "SerInp" eingelesen werden können und die im realen System über die serielle Schnittstelle eintreffen.

4.3.9.16 ADC

Import: ADCPort

Ist ein optionales Fenster zur Einstellung der Analogwerte an den ADC Eingängen, sowie den Pegeln an den Pins AIN0 und AIN1 für den Analog Komparator.

Mittels "GetADC(i)" kann das Programm den gewandelten Wert des Kanal "i" einlesen. Ist nur ein Kanal definiert, muß das "(i)" entfallen, also nur "GetADC" verwendet werden.

Die automatischen Funktionen (Sinus, Dreieck, ...) sind für zukünftige Erweiterungen vorgesehen.

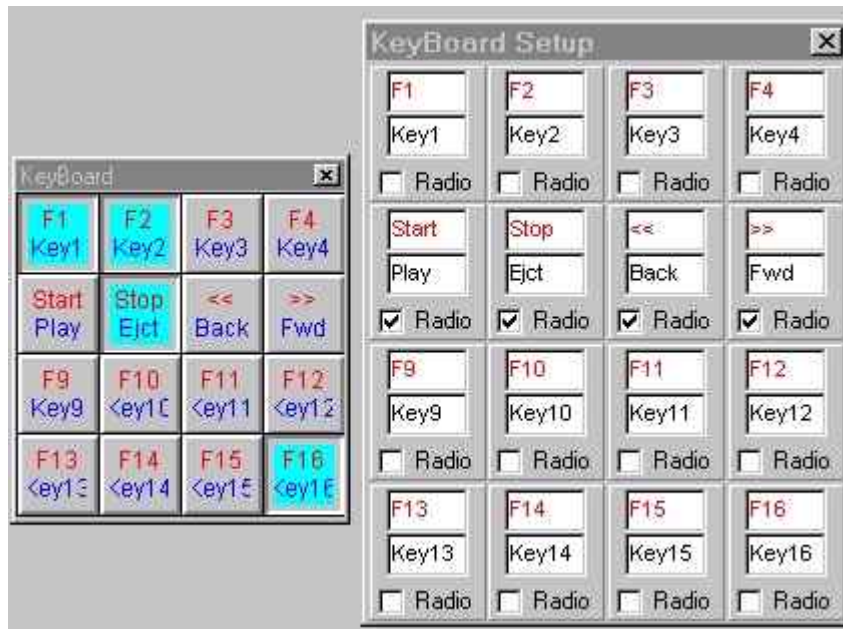


Bei den Controllern mit 10 Bit ADC steht das Ergebnis in zwei Bytes. Default ist eine rechtsbündige Anordnung (das high Byte enthält die beiden höchstwertigen Bit) und nur diese wird vom Simulator unterstützt. Die optionale linksbündige Anordnung (das high Byte enthält die 8 höchstwertigen Bit) wird nicht unterstützt. Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.17 KeyBoard 4x4

Import: MatrixPort

Ist ein optionales Fenster zum Simulieren eines Matrix Keyboard mit 16 Tasten.



Mit Rechtsklick auf eine beliebige Taste kann das Menu "KeyBoard Setup" aufgerufen werden. Hier können individuelle Beschriftungen angegeben, sowie *ein* Satz von sich gegenseitig auslösende Tasten ("Radio Buttons") definiert werden.

Zum Auslesen des Status stehen eine Vielzahl von Funktionen zur Verfügung, z.B. "ReadKeyBoard".

Der KeyBoard Treiber bietet weiterhin eine Memory Funktion, sodaß man sich ein kontinuierliches Pollen des Keyboards sparen kann

Zusätzlich wird vom System eine Semaphore "KeyBoard" exportiert, wodurch Multitasking optimal unterstützt wird (siehe Prozedur "WaitSema").

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.18 KeyBoard 8x8

Import: KeyPort8

Ist ein optionales Fenster analog zu "KeyBoard 4x4". Dieses Keyboard besteht immer aus genau 8 Zeilen und bis zu 8 Spalten. Sich gegenseitig auslösende Tasten ("Radio Buttons") stehen hier *nicht* zur Verfügung.

Zum Auslesen des Status stehen eine Vielzahl von Funktionen zur Verfügung, z.B. "ReadKeyBoard8".

Der KeyBoard Treiber bietet weiterhin eine Memory Funktion, sodaß man sich ein kontinuierliches Pollen des Keyboards sparen kann

Zusätzlich wird vom System eine Semaphore "KeyBoard8" exportiert, wodurch Multitasking optimal unterstützt wird (siehe Prozedur "WaitSema").

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.19 LCD display

Import: LCDport

Ist ein optionales Fenster zur Simulation eines/zwei an 7/ 8 Port Pins angeschlossenen LCDs mit 1,2, oder 4 Zeilen, bis zu 40 Zeichen/Zeile und Display Controller HD4470, HD66712, KS0070 oder KS0073.

Mittels Rechtsklick können die Farben für den Hintergrund und die Pixel (ein- /aus) eingestellt werden. Die Zeichenausgabe erfolgt über die Prozedur "LCDout". Es werden auch Benutzer definierte Zeichen unterstützt. Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.



4.3.9.20 LCD_M display

Import: LCDmultiPort

Ist ein optionales Fenster zur Simulation von bis zu 8 LCDs mit 1,2, oder 4 Zeilen, bis zu 40 Zeichen/ Zeile und Display Controller HD4470, HD66712, KS0070 oder KS0073. Die Displays werden über I²C I/O-Expander angeschlossen sind. Mittels Rechtsklick können die Farben für den Hintergrund und die Pixel (ein- /aus) eingestellt werden. Die Zeichenausgabe erfolgt hier über die Prozedur "LCDout_M". Es werden auch Benutzer definierte Zeichen unterstützt.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.21 7seg display

Import: Disp7sPort

Ist ein optionales Fenster zur Simulation von 1...8 stelligen 7-Segment Anzeigen. Durch Rechtsklick können die Farben für den Hintergrund und die Segmente eingestellt werden.

Die Displays können sowohl gemultiplext als auch über Schieberegister mit Latches angeschlossen werden. Die Zeichenausgabe (auch stilisierte alphanumerische Zeichen) erfolgt mittels "DispOut".

Weiterhin stehen eine Vielzahl von Funktionen wie Blinken etc. zur Verfügung. Ebenso kann ein eigener Zeichensatz definiert werden.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.22 I2C 7seg display

Import: I2C_Disp7

Ist ein optionales Fenster zur Simulation von bis zu 16 7-Segment Anzeigen, die über I²C I/O-Expander angeschlossen sind. Die Anzeigen können dabei auf 4 Displays verteilt sein, wobei jedes Display jedoch maximal 8 Stellen besitzen kann. Durch Rechtsklick können die Farben für den Hintergrund und die Segmente eingestellt werden. Die Zeichenausgabe (auch stilisierte alpha-numerische Zeichen) erfolgt hier mittels "I2C_Disp7Out".

Weiterhin stehen eine Vielzahl von Funktionen wie Blinken etc. zur Verfügung. Ebenso können eigene Zeichen definiert werden.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.



AVRco Tools

4.3.9.23 14seg display

Import: Disp14sPort

Ist ein optionales Fenster zur Simulation von 1...8 stelligen 14-Segment Anzeigen, die im Multiplex Mode betrieben werden. Durch Rechtsklick können die Farben für den Hintergrund und die Segmente eingestellt werden. Die Zeichenausgabe (alphanumerisch) erfolgt über die Prozedur "Disp14out".

Weiterhin stehen eine Vielzahl von Funktionen wie Blinken etc. zur Verfügung. Ebenso können eigene Zeichen definiert werden.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

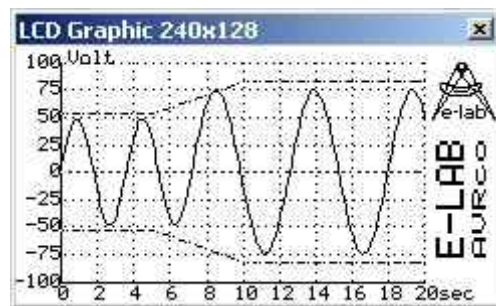
4.3.9.24 LCD Graphic

Import: LCDgraphic

Ist ein optionales Fenster zur Simulation von graphischen Displays.

Durch Rechtsklick können die Farben für den Hintergrund und die Pixel eingestellt werden.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.



Das System stellt eine Vielzahl von high Level Funktionen (Text und Graphik) bereit. Der Anwender muß lediglich die grundsätzlichen Funktionen (z.B. "write Byte" oder "set Pixel") in dem User Device "GraphIOS" definieren. Für eine Reihe von Display Controllern (z.B. T6963, SED1531, HD61202...) sind diese bereits in Beispiel Programmen enthalten.

4.3.9.25 File System

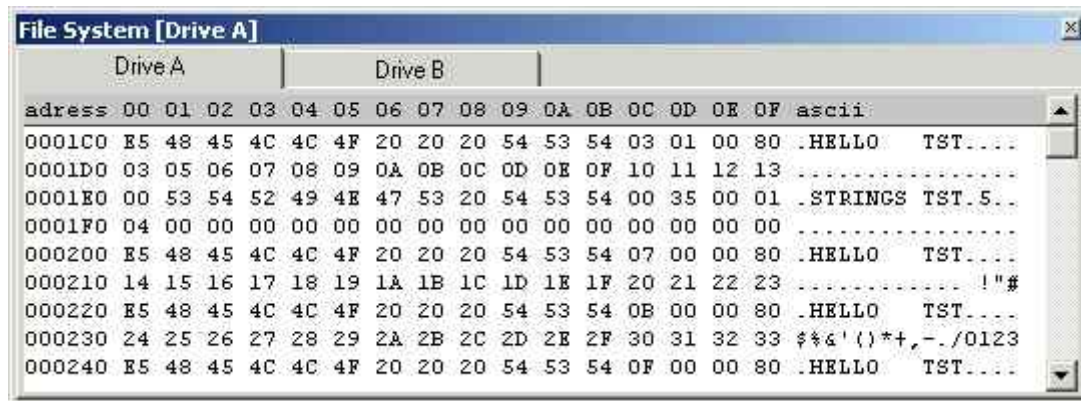
Import: FileSystem

Ist ein optionales Fenster.

Die Laufwerke werden im Arbeitsspeicher des PC nachgebildet und diese Fenster entspricht einem Disk Editor für die (maximal 4: Drive A bis D) definierten logischen Laufwerke.

Der Inhalt kann (im hexadezimalen oder im ASCII Bereich) auch geändert werden.

Ein Rechtsklick erlaubt es die Laufwerke zu formatieren.



"FileSystem" ist ein sehr einfaches Filesystem, das schon mit etwa 6 kByte Flash und 500 Byte Ram auskommt. Als Disk Hardware kommen externes SRAM, EEPROM, Flash, Floppy, Harddisk etc. infrage. Das System stellt eine Vielzahl von high Level Funktionen (z.B. Erstellen / Löschen von Dateien, Lesen / Schreiben, Datei Attribute setzten etc.) bereit. Der Anwender muß lediglich die grundsätzlichen Funktionen (z.B. Sektor lesen / schreiben) in dem User Device "FileIOS" definieren. Für das Atmel Flash AT45DB161 ist dieses bereits in einem Beispiel Programm enthalten.

Da das "FileSystem" eine proprietäre Struktur benutzt, können die Datenträger nicht unter DOS / Windows benutzt werden.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.26 FAT16

Import: FAT16

Ist ein optionales Fenster.

Das Laufwerk wird im Arbeitsspeicher des PC nachgebildet und diese Fenster entspricht einem Disk Editor für das logischen Laufwerk.

Der Inhalt kann (im hexadezimalen oder im ASCII Bereich) auch geändert werden.

Ein Rechtsklick erlaubt es das Laufwerk zu formatieren.

Im Gegensatz zum "FileSystem" Treiber ist dieses Filesystem voll PC kompatibel. Ein Minimalsystem benötigt etwa 12k Flash und 1k Ram. Direkt unterstützt werden MMC und CF Flash Karten, sowie Standard IDE Laufwerke. Für davon abweichende Hardware ist auch ein allgemeiner Treiber vorhanden. Auch hier stehen eine Vielzahl von high Level Funktionen (Datei erstellen / löschen / schreiben / lesen etc.) zur Verfügung.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.27 Incr Counter

Import: IncrPort

Ist ein optionales Fenster zur Simulation von einem Inkremental Encoder am Eingang des Analog Comparators.



Mit dem grünen/roten Pfeil wird der aktuelle Wert eingestellt ("der Encoder verdreht"). Da der Treiber im Interrupt Betrieb läuft, ist eine Änderung nur möglich wenn sich der Simulator im "Run" Mode befindet. Im "auto" Mode durchläuft der Drehgeber automatisch den gesamten Bereich. Der maximale Bereich ist von der gewählten Auflösung (16 oder 32 bit) abhängig. Mittels "pos trip" / "neg trip" kann dieser Bereich eingegrenzt werden. Mit "speed" wird die "Drehgeschwindigkeit" festgelegt.

Der Treiber arbeitet mit einem internen Zähler von wahlweise 16 oder 32 Bit. Dieser Zähler kann vom Programm gelesen / gelöscht und gesetzt werden (Get / Clear / SetIncrementalVal).

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.28 Freq Counter

Import: FreqCount

Ist ein optionales Fenster zur Simulation des Frequenzzähler Treibers. Die Simulation ist dabei auf den ersten Kanal beschränkt (der ggf. vorhandene zweite Kanal "FreqCount2" kann derzeit *nicht* simuliert werden).



Mit dem rechten Schiebeschalter wählen Sie einen Frequenzbereich 0...0,1Hz, 0...1Hz, etc. bis 0...1MHz aus. Mit dem Drehregler können Sie dann die gewünschte Frequenz einstellen. Mittels der kleinen Schaltfläche in der linken unteren Ecke wird der Frequenzzähler dann freigegeben / gestoppt. Der Frequenz Zähler kann auch zur Impulsbreiten Messung eingesetzt werden. Dementsprechend kann der aktuelle Stand über "GetFreqCounter" bzw. "GetTimeCounter" eingelesen werden. Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.29 I2C PortExpand

Import: I2Cexpand

Ist ein optionales Fenster zur Simulation von bis zu 8 über I²C I/O-Expander angeschlossenen Ports. Die eingesetzten Expander vom Typ PCA9554A verfügen über die gleiche interne Struktur wie die Prozessor Ports (PORT / PIN / DDR). Somit erfolgt die Anzeige analog zu dem "Ports" Fenster mit den symbolischen Namen "PORT 0...7" / "PIN 0...7" und "DDR 0 ...7". Die einzelnen Port Bits können hier auch geändert werden. Weitere Einstellungen sind nicht möglich.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.30 System Blinker

Import: SysLEDBlink

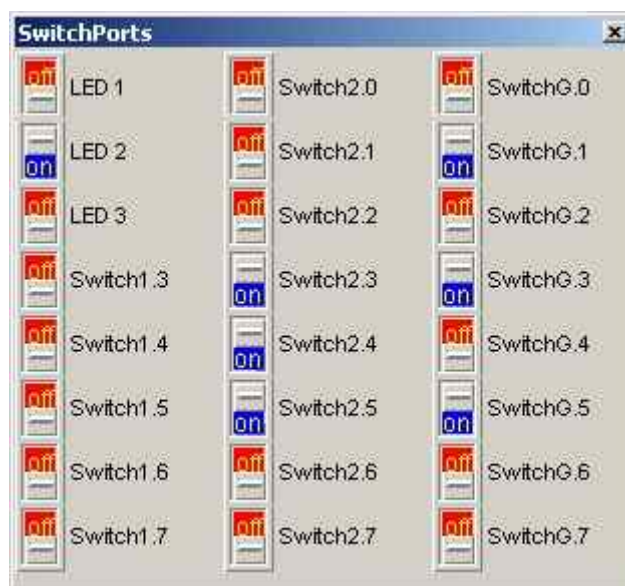
Ist ein optionales Fenster zur Simulation des System Led Treibers. Weitere Einstellungen können hier nicht vorgenommen werden. Gesteuert wird der Treiber über diverse Funktionen wie "SysLEDon", "SysLEDflashOn" etc.. Weiterhin kann mittels "SysLedFlashMsg" ein Blinkmuster ausgegeben werden (z.B. für Error Codes).

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.31 SwitchPort

Import: SwitchPort1 / SwitchPort2 / SwitchPort_G

Ist ein optionales Fenster zur Simulation des SwitchPort Treibers. Es werden nur die importierten SwitchPorts angezeigt (im folgenden Beispiel wurden alle SwitchPorts importiert).





AVRco Tools

Mittels Rechtsklick können die Bezeichnungen der Schalter, sowie deren Art (Taster = "auto release" ansonsten Schalter) geändert werden.

Zum Einlesen der entprellten Schalter gibt es die Funktionen "INP_STABLEx" und "INP_RAISEx".

Unter "PORT_STABLEx" kann der komplette entprellte Port als Byte gelesen werden.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.32 RC5receiver

Import: RC5Rxpport

Ist ein optionales Fenster zur Simulation des Infrarot Empfangs. Dazu wird ein Infrarot Empfänger simuliert. In einem kleinen Fenster können Sie das zu sendende Telegramm (das empfangen werden soll) definieren.

Die Werte werden hexadezimal angegeben. Ein angekreuztes "toggle" setzt dieses Bit = 1.

Die Schaltfläche "send" startet / stoppt die Übertragung.

Mittels der Funktion "RecvRC5" können diese Daten dann eingelesen werden.

Eine analoge Simulation für den RC5 Transmitter besteht derzeit nicht.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.33 Servos

Import: ServoPort

Ist ein optionales Fenster zur Darstellung von maximal 8 digitalen Servos. Zeigt die Servo Stellungen der Kanäle 0 bis max. 7 prozentual zum (positive / negativen) Vollausschlag. Einstellungen können hier nicht vorgenommen werden.

Die Einstellung des Servos wird vom Programm mittels der Prozedur "SetServoChan" vorgenommen.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.9.34 Heap

Import: Heap

Ist ein optionales Fenster um die aktuelle Belegung des Heap Speichers anzuzeigen.

descr @	data @	size
\$001F	\$00B3	\$0001
\$00B4	\$00B8	\$000B
\$00C3	-free-	\$0018
\$00DF	\$00E3	\$0003

Mittels der Funktion "GetMem" wird ein Speicherbereich angefordert. Diese Funktion liefert einen Pointer, da auf den Heap ausschließlich über Pointer zugegriffen wird. Mittels der Funktion "FreeMem" kann der Speicher wieder freigegeben werden,

"descr @" zeigt die Adresse der für jeden Speicherblock notwendigen Verwaltungsinformationen (4 Byte) an, "data @" zeigt die Startadresse des Daten Bereiches an, "size" die Größe des Daten Bereichs. Im obigen Beispiel ist z.B. folgendes erkennbar:

- der Bereich des Heap beginnt bei der Adresse \$00AF
- der erste benutzte Bereich belegt für ein einziges Byte Nutzdaten den Adressbereich von \$00AF bis \$00B3 (jeweils einschließlich) = 5 Byte
- der zweite benutzte Bereich belegt für 11 Byte Nutzdaten 15 Byte
- durch eine Speicher Freigabe ist der Heap fragmentiert. Es besteht ein freier Bereich zw. \$00C3 und \$00DE (jeweils einschließlich) entsprechend 28 Byte oder 24 Byte Nutzdaten.

4.3.9.35 Stepper

Import: StepPort

Ist ein optionales Fenster.



Zeigt die Geschwindigkeit des Schrittmotors in Schritten/Sekunde als Balken und als Dezimalwert an. Die Balkenanzeige ist zwischen 0 und der maximalen Schrittgeschwindigkeit, die mit der Definition von "StepMaxFreq" angegeben wird, skaliert.

Weiterhin wird die Position in Schritten und der aktuelle Modus angezeigt:

Mode "StepUp":	Beschleunigungsrampe fahren
Mode "StepDown":	Abbremsrampe fahren
Mode "StepStop":	Angehalten
Mode "StepRun":	Drehen mit maximaler Geschwindigkeit.

Zum Ansteuerung der Schrittmotor Endstufen stehen eine Reihe von Funktionen zur Verfügung:

"StepperOneCW":	ein Schritt im Uhrzeigersinn drehen
"StepperOneCCW":	ein Schritt gegen den Uhrzeigersinn drehen
"StepRampCW":	Beschleunigungsrampe im Uhrzeigersinn fahren

etc.

Der StepPort ist zur Ansteuerung von H-Brückenschaltungen vorgesehen. Um auch intelligente Treiber Bausteine zu unterstützen gibt es weiterhin einen UserPort Modus.

Da im JTAG Mode die Simulation / das Debugging mit der realen Hardware stattfindet ist die Funktion da nicht sinnvoll / verfügbar.

4.3.10 Menu Help

In diesem Menu ist derzeit nur die "Info"- Funktion implementiert, um die Version des Simulators anzuzeigen.

4.4 SysTick und Scheduler Timings

Der SysTick ist das Arbeitspferd im AVRco System. Er erledigt die meisten der Hintergrund Jobs, z.B. Entprellen von Ports, Lesen des ADC, Software RTC, Bearbeiten diverser Timeouts, Beeper, Blinker, Refresh der LED-Displays, Software Timer, Keyboard Scan etc. Weiterhin ist beim MultiTasking der Scheduler ein Teil des SysTicks. In extrem ausgebauten Systemen werden hier also auch extrem viele Arbeiten erledigt.

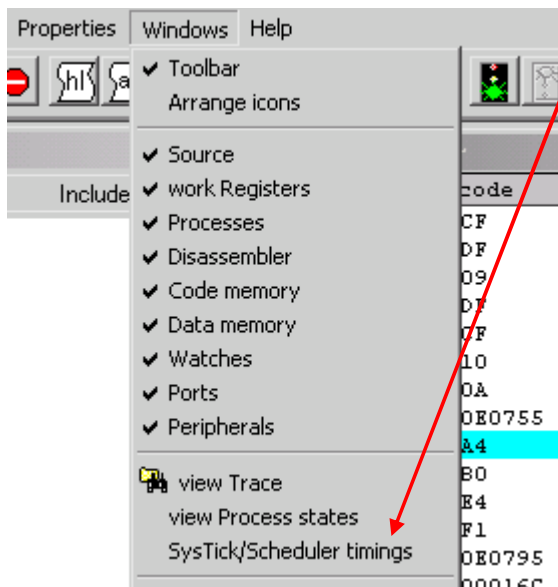
Da der SysTick immer ein Hardware Timer Interrupt ist (Timer0 oder Timer2), ist während der ganzen Verarbeitung, die der Tick leistet, auch der globale Interrupt gesperrt. In den meisten Fällen ist diese Sperrung unkritisch, da die evtl. anstehenden anderen Interrupts nur zeitlich verschoben werden und dann trotzdem zum Zug kommen. Bei Interrupt Burst z.B. von einem sehr schnell von extern geschalteten Int0 etc kann es aber dann trotzdem vorkommen, dass ein Interrupt nicht verarbeitet wird weil schon der nächste ansteht. Das System wird deshalb nicht instabil, aber es gehen u.U. Informationen verloren.

Gravierend kann dies z.B. bei der seriellen Schnittstelle sein (Rx mit Buffer). Ist der Interrupt länger als die Transferdauer zweier Zeichen gesperrt, wird das erste Zeichen im UART überschrieben. Moderne AVR's haben deshalb im Empfangszweig ein doppelt gepufferter UART, was das Problem erheblich entspannt. Aber auch hier kann es zu Zeichenverlust kommen bei extrem hoher Baudrate, grossem SysTick Job und niedrigem Prozessor Clock. Es ist daher immer ratsam mit möglichst hohem Prozessortakt zu arbeiten, vor allen Dingen bei sehr grossem Treiber Import. Die Art der Applikation ist hier nicht ausschlaggebend, sondern die Verarbeitung im SysTick.

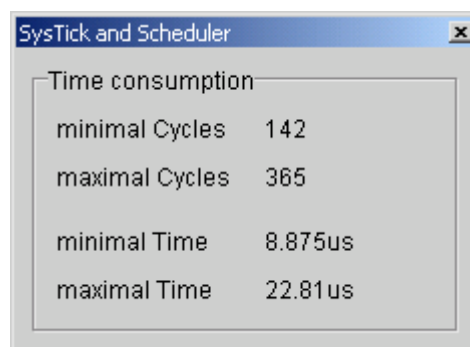
Es ist nicht möglich irgendeine Berechnungsformel anzugeben, wie lange der Interrupt während eines SysTick Interrupt gesperrt bleibt. Hier sind extrem viele und unterschiedliche Faktoren massgebend.

Um trotzdem eine grobe Zahl (clock zyklen oder usec) zu bekommen, bietet der AVRco Simulator hier Unterstützung an. Während eine Debug Laufs wird die Anzahl der CPU Zyklen innerhalb der Tick-Verarbeitung gezählt. Der kleinste und der grösste Wert wird gespeichert. Das geht natürlich nur im Simulations Mode, mit ICEs und ROM Monitor ist das nicht möglich. Der Debuglauf sollte natürlich möglichst alle Stadien des Programms einmal durchlaufen haben um präzise Ergebnisse zu erhalten.

Nach einem ausreichend langen Simulator Lauf kann man sich die erfassten Werte anzeigen lassen. Dazu wird im Simulator das Menu „Windows“ geöffnet und das Item **SysTick/Scheduler timings** angeklickt.



Dadurch wird ein Fenster geöffnet, welches die erfassten minimalen und maximalen Interrupt Sperrzeiten während des SysTicks anzeigt.



Wie aus dem Bild zu ersehen ist, ist die maximale Sperrzeit hier ca. 23usec. Bei Multiprocessing und niedrigem Prozessor Clock und vielen Importen kann diese Zeit 100usec und mehr betragen. Es macht deshalb auch bei offensichtlich unkritischen Programmen durchaus einen Sinn mit dem max. Processor Clock zu arbeiten.

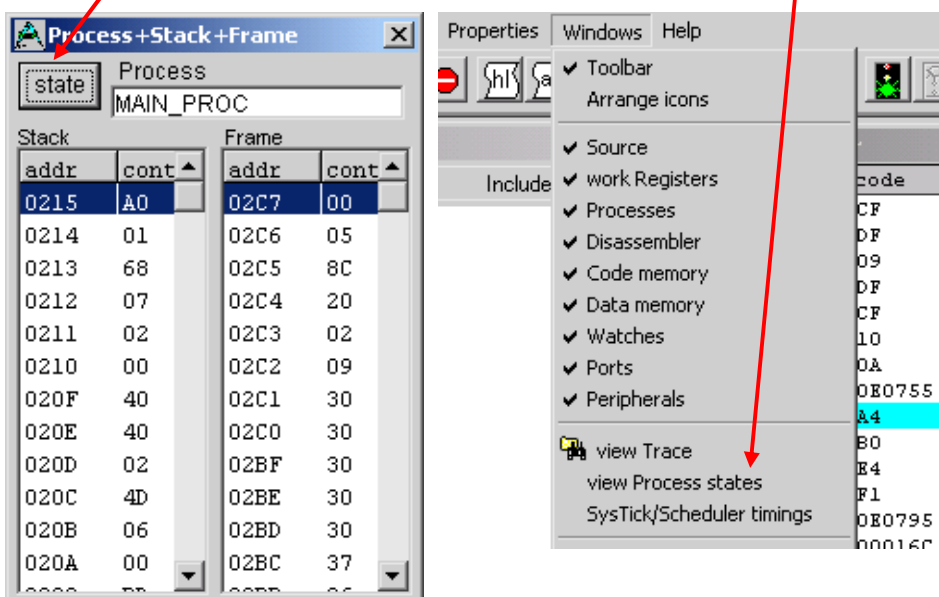
4.5 Frame und Stack Verbrauch im Simulator ermitteln

Ein absolut kritischer Punkt bei einem System Design ist die Definition der Stack und Frame Grösse. Erfahrene Programmierer können dies oft in ungefähr abschätzen. Werden die Ressourcen jedoch knapp (RAM) ist hier ein Tuning notwendig. Wird zu wenig Speicher für Stack oder Frame requiriert, kann dies u.U. zu bösen Überraschungen führen, manchmal erst nach Wochen oder Monaten, wenn mehrere Ereignisse zusammentreffen.

Ein Stack oder Frame Überlauf zeigt sich meistens in einem irregulären Verhalten des Programms, z.B. werden Berechnungen falsch ausgeführt. Ganz offensichtlich wird es bei Text Anzeigen, wenn Texte verstümmelt werden. Das kommt daher dass die Stringverarbeitung und Konvertierung extrem Stack und Frame intensiv ist.

Da auch hier keine allgemein gültige Angaben gemacht werden können und auch keine irgendwie geartete Formel angegeben werden kann, muss die tatsächlich benötigte Stack und Frame Grösse wiederum im Simulator ermittelt werden. Hier stellt sich aber das Problem der fehlenden Umwelt (Realität) in einem noch grösseren Masse. Bestimmte Prozeduren, Funktionen, Prozesse und Tasks werden oft nur ausgeführt, wenn externe Ereignisse anliegen. Das kann der Simulator selbst natürlich nicht wissen. Deshalb ist es Sache des Programmierers möglichst alle dieser Ereignisse in einer geeigneten und sinnvollen Weise herbeizuführen, z.B. durch Manipulation der Ports etc.

Der AVRco Simulator prüft nicht nur alle Stack und Frame Operationen während eines Simulations Laufs, er zeichnet auch die maximal Werte auf, und zwar getrennt für alle vorhandenen Stacks und Frames. Nach einem ausgiebigen Simulations Lauf kann man sich diese Diversen Informationen anzeigen lassen und zwar einmal mit dem **state** Button des Stack/Frame Fensters oder durch das **view Process states** Item des Windows Menus.



Dadurch wird unten stehender Dialog geöffnet, der eine gute Übersicht über die einzelnen Stacks und Frames bietet.

War der Simulations Lauf lange und umfassend genug, so kann ein Peak-Wert als Anhalts Punkt herangezogen werden.

Aus Sicherheits-gründen sollten u.U. noch einmal 50% hinzu gerechnet werden.

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



AVRco Tools

4.6 Frame und Stack Check zur Laufzeit

Grosse Prozessoren aus dem 16 und 32bit Bereich bieten einen internen Hardware Check um Stack und Frame Überläufe zur Laufzeit feststellen und melden zu können. Leider hat der AVR eine solche Einrichtung nicht. Es ist zwar möglich dies in Software nachzubilden, erste Versuche haben aber gezeigt, dass der Code sich um bis zu 50% vergrössert und das Programm um den gleichen Faktor langsamer wird. Der Grund dafür ist, dass **jeder** PUSH und **jeder** POP eine Stackpointer Prüfung nach sich ziehen muss, ansonsten wäre der Check ja nutzlos. Das gleiche gilt auch für den FramePointer (R28/R29).

Mit ein paar kleinen Tricks kann man sich aber trotzdem behelfen. Dies ist zwar nicht so sicher wie ein kontinuierlicher Hardware oder Software Check aber trotz allem sehr hilfreich.

Die Checks können in zwei Stufen implementiert werden. Die erste simplere ist eine System Funktion, die genau weiss, wo der jeweilige Stack oder Frame anfängt und wo er aufhört. Beim Aufruf dieser Funktion prüft diese den ganzen Stack/Frame Bereich, beginnend von der niedrigsten Adresse (Top-of). Ein noch nicht benutztes Byte sollte noch von der Speicher Initialisierung her „00“ enthalten. Wurde diese Speicherstelle schon einmal benutzt, so hat sie normalerweise einen Wert <> „00“. Aber ganz sicher kann man da nicht sein, da ja auch „00“ gepusht worden sein kann. Die Funktion prüft jetzt also alle Speicherstellen des Stacks/Frames bis sie auf einen Wert <> „00“ stösst. Die Anzahl der gezählten Nullen ist offensichtlich der noch nicht benutzte Bereich. Die Funktion gibt diesen Zählwert als Ergebnis zurück.

Prüfung des Stacks und Frames des Hauptprogramms bei Multitasking **und** non-multitasking Programmen:

```
Function GetStackFree : word;  
Function GetFrameFree : word;
```

Prüfung des Stacks und Frames von Prozessen und Tasks bei Multitasking Programmen:

```
Function GetStackFree(prcs : Process) : word;  
Function GetFrameFree(tsk : Task) : word;
```

4.6.1 Erweiterte Stack/Frame Checks

Eine erweiterte Version besteht darin, dass das Ende eines Stacks oder Frames mit einem Word mit einem bestimmten Inhalt belegt wird. Wird dieses Word ganz oder in Teilen verändert, so steht fest, dass ein Stack oder Frame Überlauf vorhanden war. In diesem Fall kehrt die Funktion mit einer „-1“ zurück, ansonsten mit der Zahl der unbenutzten Bytes. Hierzu muss ein kleiner Treiber importiert werden.

```
From System Import LongInt, StackChecks, ..;
```

Prüfung des Stacks und Frames des Hauptprogramms bei Multitasking **und** non-multitasking Programmen:

```
Function CheckStackValid : integer;  
Function CheckFrameValid : integer;
```

Prüfung des Stacks und Frames von Prozessen und Tasks bei Multitasking Programmen:

```
Function CheckStackValid (prcs : Process) : integer;  
Function CheckFrameValid (tsk : Task) : integer;
```

Es ist klar, dass damit nicht festgestellt werden kann, wer oder was die Stack oder Frame Verletzung verursacht hat. Zumindest das Vorhandensein eines Überlaufs kann aber erfasst werden.

Ein generelles Problem mit solchen Tests ist immer „was tun“ wenn ein Fehler auftritt? Eine Bildschirm Ausgabe wie beim PC ist hier nicht möglich, da dieser nicht vorhanden ist und evtl. auch nicht funktionieren kann, da mit grosser Wahrscheinlichkeit durch den Stack oder Frame Überlauf das System instabil geworden ist.

Mit einem ICE macht dies allerdings trotzdem wiederum Sinn, da hier immer noch z.B. der Speicher ausgelesen werden kann. Und zumindest das Wissen, dass ein Stack oder Frame Problem aufgetreten ist, hilft sehr viel weiter.

4.7 JTAG / OWD Debugging

implementiert mit grossartigem Support von Victor Chekygin

Auch im Programmer muß unter "Options – Programmer options" der JTAG mode eingestellt werden.

4.7.1 UpLoad / DownLoad

Bedingt durch sehr primitive Debug Logik in den Controllern dauert ein Up-/Download der Daten bis zu einer Minute.

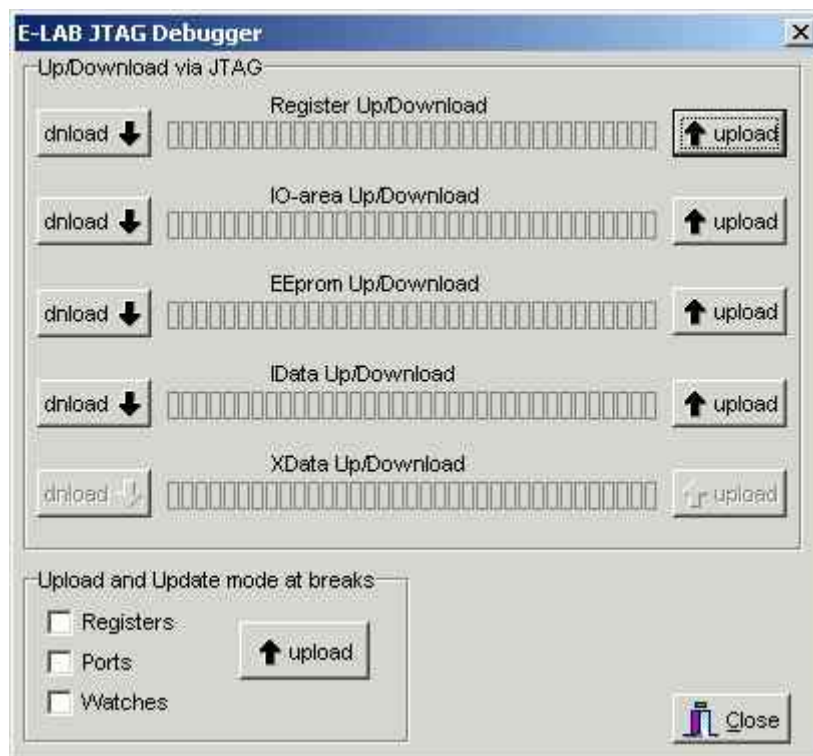
Aus diesem Grund werden beim Erreichen eines Breakpoints die Daten *nicht automatisch* aktualisiert.

Werden die aktuellen Daten benötigt, muß dies der Benutzer dem Simulator durch Klick auf den grünen Pfeil in der Toolbar mitteilen.

Hinweis: dieser Button steht nur bei aktiviertem JTAG Debugging zur Verfügung.



Dadurch öffnet sich ein Fenster



Hier können die verschiedenen Bereiche ausgewählt werden, die vom Controller zum Simulator ("Upload") oder vom Simulator zum Controller ("Download" von veränderten Bereichen) geladen werden sollen.

Weiterhin kann mittels der "Check-Boxen" unter "Upload and Update mode at breaks" ein automatischer Upload der Register/Ports/Watches bei jedem Breakpoint eingestellt werden.

Dadurch wird aber *nicht das komplette RAM (\$IDATA)* automatisch aktualisiert.

Einzelne Arbeitsregister des Controllers können im Fenster "work Registers" durch Doppelklick auf den entsprechenden Register Namen oder Register Inhalt aktualisiert und/oder geändert werden.

Im Fenster "watches" können durch Rechtsklick sämtliche Watches aktualisiert werden.

Komplexe Variablen (Arrays / Strukturen) können durch Doppelklick auf die Struktur aktualisiert werden.

4.7.2 Hardware Breakpoints

Bei aktiviertem JTAG Debugging steht in der Toolbar ein weiterer Button zur Verfügung:



Dieser öffnet ein Fenster zur Einstellung der maximal 3 Hardware Breakpoints im JTAG Mode. Die Restriktion auf 3 ist durch die Implementierung des JTAG Debugging in den Controllern vorgegeben.



"Standard" Breakpoints können beliebig im Programm Code gesetzt werden.

"Extended" Breakpoints halten das Programm beim Zugriff auf den Speicher an. Die verschiedenen Modi können dann bei "Breakpoint modes" eingestellt werden.

"masked" Breakpoints betreffen ebenfalls den Datenspeicher. Hier kann jedoch ein ganzer Bereich angegeben werden. Eine binäre 0 in der Maske bezeichnet ein "don't care" Bit.

Im obigen Beispiel (falls "1standard, 1 masked" ausgewählt ist):

Adr = 04D1, Maske = FFFF alle Bits relevant: eine Breakbedingung bei 04D1

Adr = 04D1, Maske = FFFE Bit 0 nicht relevant: Break bei 04D0 oder 04D1.

4.7.3 Tips und Bemerkungen zu JTAG Debugging

achten Sie darauf, die Firmware Ihres ISP aktuell zu halten!
Viele Verbesserungen bedingen ein Firmware Upgrade.
(Firmware Download/Update in das ISP-USB ICE nur mit extern 5Volt am Target Stecker!!)

Die Wartezeit nach einem Break oder Step hängt von der Zahl und der Komplexität der Watches ab.
Viele Watches, viel Zeitverbrauch.
Die automatische Anzeige der lokalen Watches in Prozeduren/Funktionen kann abgeschaltet werden (rechte Maustaste im Watch Fenster).

Der JTAG Debugger kann optional auch Stack und Frame Überlauf bei Funktions Aufrufen feststellen, allerdings nur unmittelbar nach einem Single Step, einem Programm Stop oder Breakpoint.
Da auch diese mit einem grossen Datenverkehr zwischen PC und AVR verbunden ist, ist diese Prüfung normalerweise abgeschaltet. Freigabe im "Local Watch" Fenster mit rechtem Maus Click.

Unterstützen Sie den Debugger dadurch, dass Sie möglichst immer nur 1 Pascal Statement pro Source Zeile schreiben. Er wird es Ihnen durch schnelleres single-step danken.

Alle Dateien des Systems müssen für ein ordentliches JTAG Debugging immer synchron sein, eine Änderung im Source Code ohne recompile und Flash Programmieren kann zu extrem seltsamen Verhalten des Debuggers führen.

Im Aufruf Fenster des JTAG-ICE ist ein Schieberegler für das USB-Port zu finden. Damit können gewisse, vom PC abhängige Timeouts eingestellt werden.
Dieser Regler sollte am Anfang of long stehen und kann bei sicherem Betrieb schrittweise Richtung short verändert werden.
Die Geschwindigkeits Zuwächse sind allerdings recht bescheiden, also nicht übertreiben.

4.8 Compiler Schalter und deren Auswirkungen

4.8.1 `{$D+}` `{$D-}`

schaltet Debug Informationen ein bzw. aus. Wenn aus, werden die folgenden Statements nicht im Single Step Modus abgearbeitet.

4.8.2 `{$E+}` `{$E-}`

Die nachfolgenden Statements werden durch den Simulator nicht ausgeführt. Hilfreich bei sehr langen mDelays während des Debuglaufs



AVRco Tools

5 LookUp und Interpolation

5.1 Nichtlineare Funktion von Sensoren

Viele Sensoren, aber auch andere Funktionen, arbeiten extrem nicht-linear. Beispiel: PT100, PTC, NTC, Foto-Detektoren, aber auch z.B. Dioden. Die Reihe lässt sich fast beliebig fortsetzen. Da dies alles analoge Werte sind, werden diese in der Regel mit einem AD-Wandler erfasst. Normalerweise steht das Messergebnis in einem bestimmten Verhältnis zu dem äusseren Ereignis, z.B. Temperatur. Aber eben dieses Verhältnis ist sehr oft nicht linear. Ein PT100 z.B. hat bei 0grad einen Widerstand von 100 Ohm bei 50grad ca. 124 Ohm, bei 100grad ca. 143 Ohm. Der Zusammenhang zwischen Temperatur und Widerstand ist nicht-linear.

Um jetzt die aktuelle Temperatur zu erhalten, kann man zwei Wege beschreiten:

1. Mit einer passenden Formel, die meistens sehr komplex ist, kann die dem Widerstandswert zugeordnete Temperatur ermittelt werden.
2. Man erstellt sich eine sogenannte LookUp-Table in der in Stufen bestimmte Widerstandswerte und die zugehörige Temperatur abgelegt sind. Mit dem Widerstandswert (Messwert) als Index wird jetzt auf die Tabelle zugegriffen. Kann das Messergebnis von 100 bis 200 laufen, werden 100 Werte (Ergebnisse) in der Tabelle benötigt. Etwas schwieriger wird es, wenn der Analogteil und AD-Wandler so beschaltet ist, dass das Messergebnis von 0 bis 1023 variiert. Dann braucht man 1024 Einträge in der Tabelle.

5.2 Linearisierung

Die vorliegende Implementation basiert auf einer Tabelle, in der sowohl die Mess-Ergebnisse als auch die zugehörigen Resultate immer paarweise abgelegt sind. Der LookUp Algorithmus sucht mit dem Messergebnis als Argument in der Tabelle, bis er auf einen gleichen Wert stösst, oder mit dem Argument zwischen 2 Werten liegt. Der Suchvorgang wird aus Geschwindigkeitsgründen mit einem binären Verfahren ausgeführt.

Ist ein passender Wert gefunden, wird das Ergebnis zurückgegeben. Liegt das Argument zwischen 2 Werten, wird linear interpoliert. Dieses Verfahren ermöglicht je nach Genauigkeits Forderungen relativ kurze Tabellen. Wenn die Koordinaten Zahl in der Tabelle relativ hoch ist, ergibt die lineare Interpolation eine ausreichende Genauigkeit, da man im allg. davon ausgehen kann, dass kurze Abschnitte auch näherungsweise linear sind.

5.2.1 System Funktionen:

```
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;
```

Der Pointer muss in die Tabelle im ROM zeigen. Das erste Argument ist der Suchwert. Er bestimmt auch die Arbeitsweise (Integer, LongInt oder Float) der Funktion. Das Ergebnis wird im zweiten Parameter abgelegt, wenn die Suche erfolgreich war.

5.3 LookUp Table Definition und Import

Der Parameter **const LookUp : pointer** der den Funktionen übergeben werden muss, muss ein Pointer sein, der auf die im ROM liegende Tabelle zeigt. Diese Tabelle muss so konstruiert sein:

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
```

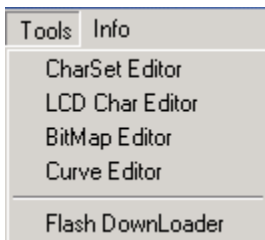
Die Array Definition ist eigentlich nur proforma als Platzhalter für das zu ladende binär File gedacht. Der Parameter **Size** bestimmt die Anzahl der x/y-Datenpaare und der dahinter stehende Multiplikationsfaktor (4 bzw. 8) ist die Anzahl der Bytes für einen Datenpaar (sizeof(integer), sizeof(LongInt) oder sizeof(float)). Die drei Bytes Info enthalten den Datentyp und die Datenpaar Zahl.


5.4 Erzeugung der LookUp Table

Zur Erstellung der Tabelle enthält das System den Table Generator **CurveGen**, mit dem man graphisch und interaktiv eine Kurve erstellen und als binär File zum Import in die Applikation ablegen kann.

Ein Beispiel Programm ist unter den Demos als "AVR Interpol" zu finden. Hierin wird ein optischer Entfernungsmesser (Sharp) linearisiert und die Entfernung in cm ausgegeben. Ein Datenblatt dieses Sensor ist ebenfalls in der Applikations Directory zu finden.

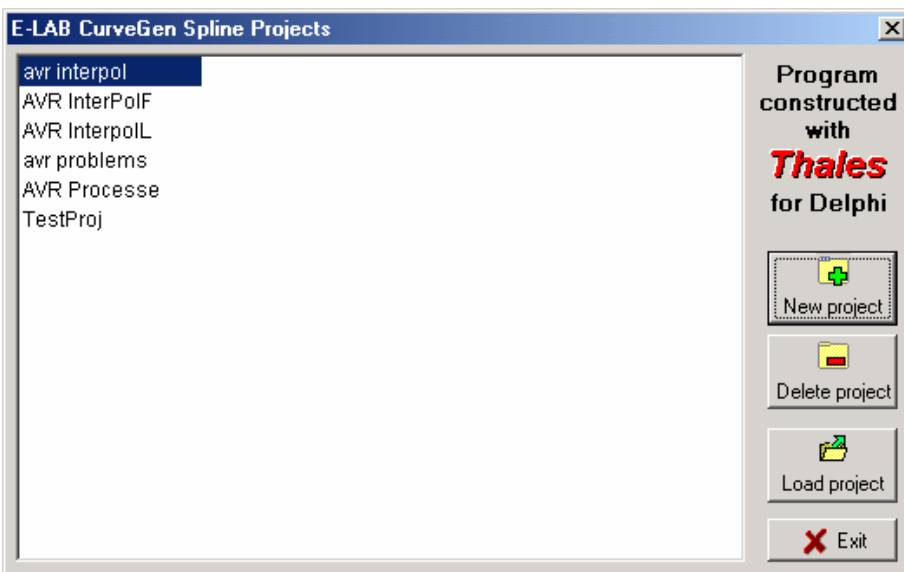
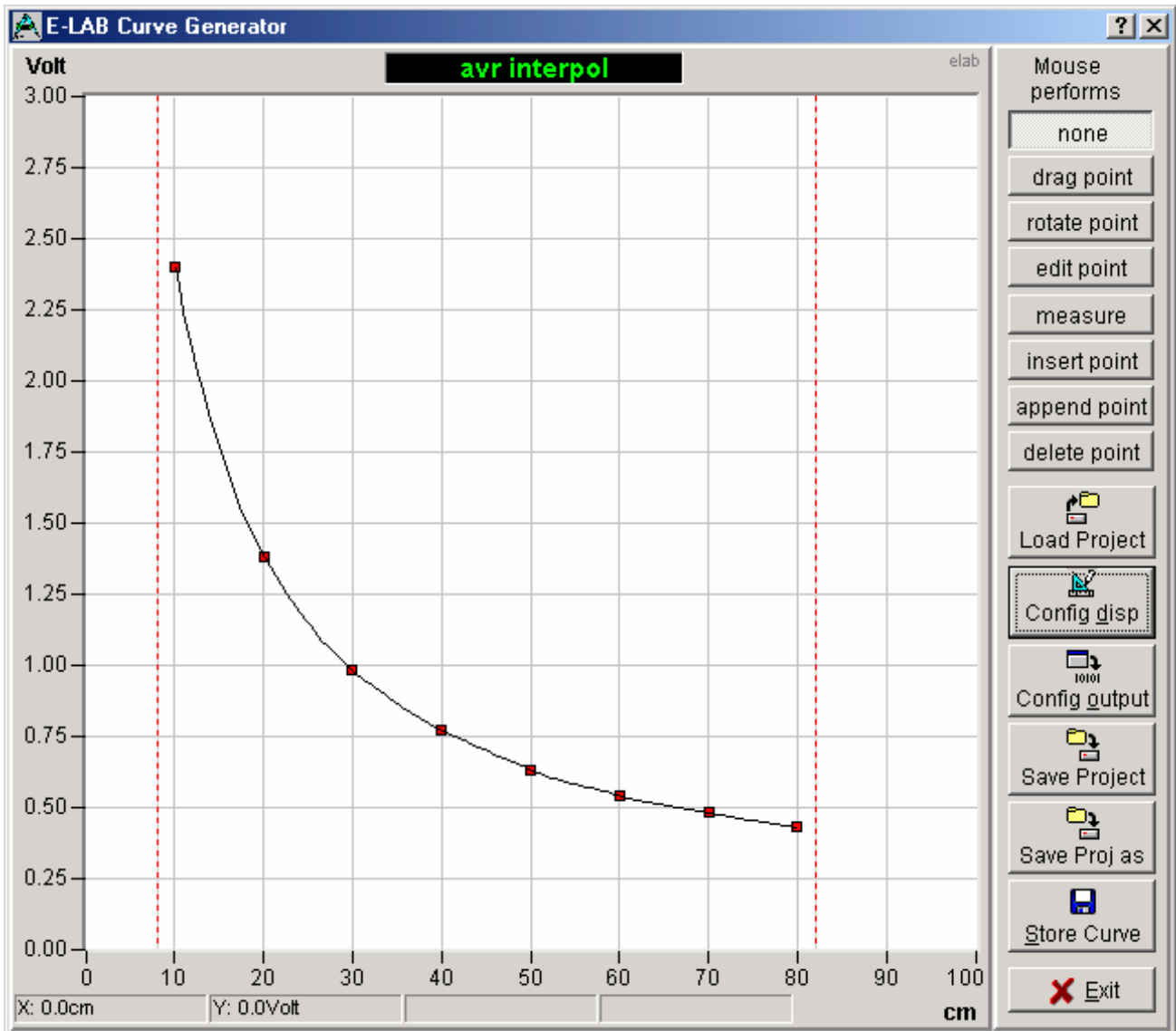
5.4.1 Programm Aufruf



Das Programm *CurveGen.exe* wird aus der IDE PED32 heraus mit dem Button  oder mit nebenstehendem Menu aufgerufen. Es generiert nach Erfassung der Parameter und Stützpunkte das für die Interpolation benötigte binär File.

CurveGen ist vollkommen interaktiv und graphisch ausgelegt. Dadurch erhält man immer eine optische Aussage für jede Manipulation der Parameter. Die Teilkurven zwischen den vorgegebenen Stützpunkten werden nach einem speziellen, Spline ähnlichen Verfahren errechnet. Damit ist es auch möglich treppenförmige Kurven zu erzeugen, was mit einem reinen Spline nicht möglich ist.

Das unten stehende Bild zeigt einen Screenshot von *CurveGen* mit dem geladenen Projekt **AVR Interpol**. Die beiden senkrechten roten Linien bezeichnen dabei die Anfangs und Endpunkte für die Table Generierung. Die roten Quadrate sind die vorgegebenen Stützpunkte.



Der Projekt Auswahl Dialog bietet neben dem Laden schon vorhandener Projekte auch Projekte zu löschen als auch neue Projekte anzulegen.

Mit einem Doppelclick auf einen Eintrag wird dieses Projekt geladen.

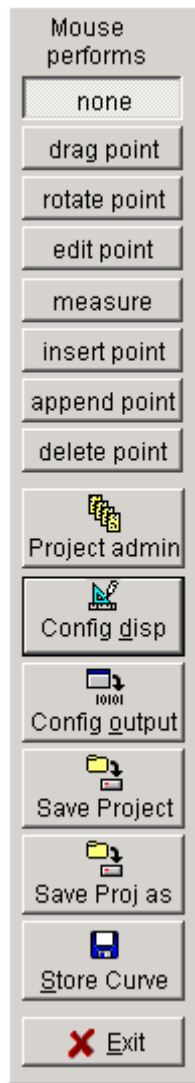
Mit einem Click auf den **New project** button erscheint ein Dialog zur Eingabe des neuen Projekt Namen und und des Arbeits Verzeichnisses.

Ein neues Projekt enthält schon eine Default Einstellung und eine kleine Zahl von passenden

Stützpunkte. Sämtliche Einstellungen müssen jetzt den Vorgaben (Sensor etc) entsprechend angepasst werden.



AVRco Tools



Die obere Hälfte des nebenstehenden Bildes zeigt die Einstellmöglichkeiten für Maus Clicks und Bewegungen.

None bedeutet, wie der Name sagt, dass die Maus keine spezielle Funktion im Display hat.

Drag point verschiebt einen angeklickten roten Stützpunkt, solange die linke Maustaste gedrückt bleibt.

Rotate point erlaubt durch anlicken und ziehen mit der Maus einem Punkt eine bestimmte Richtung (Vektor) und Gewichtung zu geben.

Mit **edit point** wird durch einen Click auf einen Stützpunkt ein Dialog eröffnet, mit dem man die exakte Position (x/y) des Punktes eingeben kann.

Measure erlaubt durch rechten bzw. linken Mausclick zwei Fadenkreuze zu verschieben, die als Messpunkte dienen. Die Resultate werden am unteren Rand eingeblendet.

Mit **Insert point** können durch Mausclicks zusätzliche Stützpunkte zwischen zwei vorhandenen eingesetzt werden.

Append point hängt mit jedem Mausclick einen Stützpunkt an den letzten rechten an. Mit **delete point** kann ein vorhandener Punkt mit einem Mausclick gelöscht werden.

Project admin öffnet den weiter unten beschriebenen Projekt Dialog.

Config display öffnet den Dialog zum Einstellen der Display Parameter wie Skalierung, Teilung etc. Beschreibung siehe weiter unten.

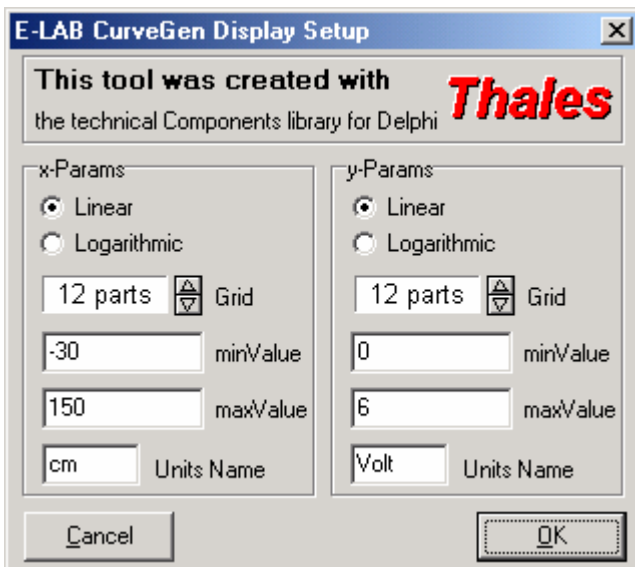
Config output öffnet den Dialog zum Editieren der Ausgabe Parameter, wie z.B. Datenformat (integer, longint etc). Beschreibung siehe weiter unten.

Mit **Save Project** werden die aktuellen Projekt Daten und Einstellungen in der Projekt Datei *ProjectName.inicg* abgelegt.

Mit **Save Proj as** kann das aktuelle Projekt unter einem anderen Namen abgelegt werden.

Store Curve öffnet den weiter unten stehenden Export Dialog. In einem Fenster sind dabei im Klartext die generierten Tabellen Daten zu sehen, mit einem Überblick zu den Einstellungs Vorgaben.

Exit beendet das Programm.

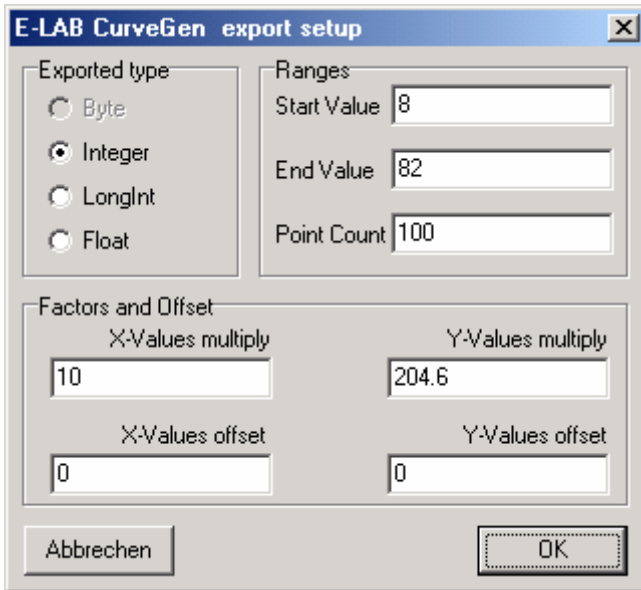


Mit dem **Display** Setup Dialog wird die Skalierung und Teilung des Displays eingestellt.

Die Einstellung erfolgt für die x und y Achse getrennt. Es kann zwischen linearer und logarithmischer Darstellung gewählt werden.

Die Einstellung der Skalenteilung muss so gewählt werden, dass bei der Skalenbeschriftung keine Rundungsfehler auftreten.

Die Minimal Werte der Skalierung beziehen sich auf die untere linke Ecke, die Maximal Werte auf die obere rechte Ecke des Displays.



E-LAB CurveGen export setup

Exported type

Byte

Integer

LongInt

Float

Ranges

Start Value

End Value

Point Count

Factors and Offset

X-Values multiply

Y-Values multiply

X-Values offset

Y-Values offset

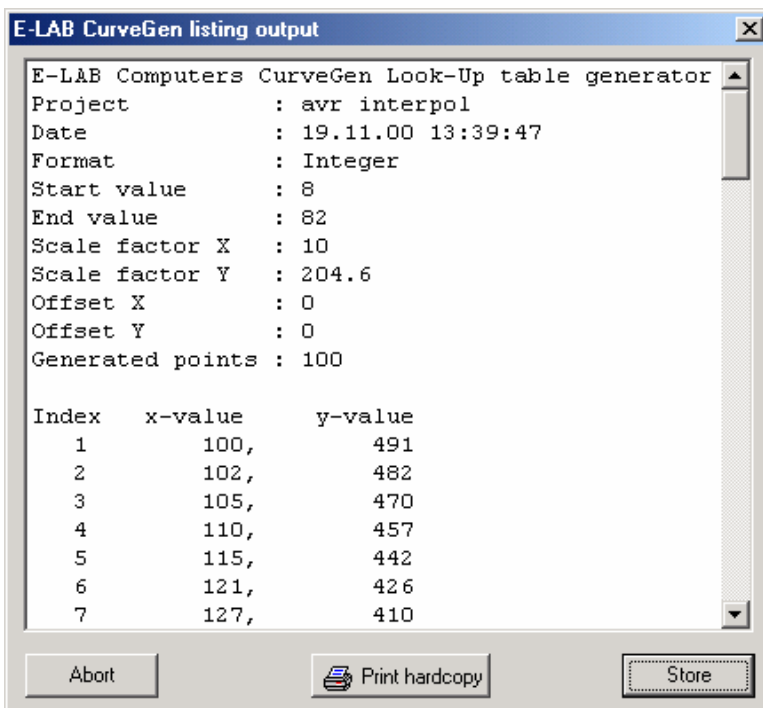
Abbrechen

Mit nebenstehendem **Export** Dialog werden die Vorgaben für die Erzeugung des binär Files erfasst. Der Werte Typ wird unter **Exported type** eingestellt. Mit **Ranges** wird ein Ausschnitt aus der Kurve ausgewählt, der im Display mit den roten senkrechten Linien dargestellt wird. Start und End Value beziehen sich dabei auf die x-Achse. Es werden dabei nur die Punkte exportiert, die innerhalb dieser X-Grenzen liegen.

Der Wert **point count** bestimmt wieviele Werte innerhalb des gewählten Bereiches im binär File abgelegt werden sollen.

Für die x und y Werte können **Skalierungen** und **Offsets** für den Export angegeben werden. Die Berechnung sieht dabei so aus:

$$\text{val} := (\text{val} - \text{offset}) * \text{factor}$$



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

Abort

Der **Output** Dialog listet alle relevanten Vorgaben und die generierten Wertepaare auf. Diese Liste kann auch ausgedruckt werden.

Mit dem **Store** Button wird das Binär File *ProjectName.crvg* erzeugt, das direkt in die Source nach oben beschriebenen Schema importiert werden kann.

5.5 Eigenschaften von CurveGen

Genauigkeit:

Das Programm basiert auf einer speziellen Spline Funktion. Spline hat die Eigenschaft, dass die generierte Kurve immer durch ihre Stützpunkte geführt wird. Das bedeutet in der Konsequenz, dass jede Ungenauigkeit bei der Erfassung der **Stützpunkte** zumindest bei diesem Punkt voll und in seiner näheren Umgebung zum Teil durchschlägt. Damit wird die Präzision der interpolierten Werte zum ganz überwiegenden Teil von den Stützpunkten abhängig.

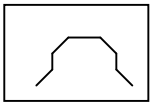
Nicht ganz so gravierend für die Genauigkeit ist die **Anzahl** der zur Verfügung gestellten Stützpunkte. Bei „normalen“ quadratischen, kubischen oder logarithmischen Kurven kommt man mit relativ wenigen Punkten aus (5..20). Je komplexer die Kurvenform ist, desto mehr Stützpunkte braucht man. Der Extremfall wäre eine logarithmische Kurve mit linearen Teilen und Treppen.

Kurvenform:

Durch die Zahl und Lage der Stützpunkte ist die Kurvenform im Normalfall schon sehr gut vorgegeben. Eine unbearbeitete neue Kurve hat noch eine strenge Konzentration der Richtungen und Gewichte auf den einzelnen Stützpunkten. Das ist leicht daran zu erkennen, dass die Teilstücke zwischen den Punkten linear verlaufen.

Dieser Effekt ist aber normalerweise unerwünscht. Eine Kurve sollte auch eine Kurvenform haben. Deshalb ist es unumgänglich die Attribute der Stützpunkte richtig einzustellen. Dies geschieht mit der Maus und der Einstellung **rotate point**. Mit gedrückter linker Maustaste wird der Punkt gezogen.

Es bewegt sich jetzt allerdings nicht der Punkt, sondern visuell seine beiden Attribute in Form eines weissen Kreises. Durch verschieben des weissen Kreises wird die Gewichtung und die Richtung (Vektor) des Punktes verändert. Je grösser die Distanz des Kreises vom Punkt ist, desto mehr wird die Kurve in diese Richtung verzogen. Diese Einstellung muss ebenfalls sehr sorgfältig vorgenommen werden, damit die erzeugte Kurve möglichst nahe an die Ideal Kurve herankommt. Das menschliche Auge ist in dieser Richtung relativ unbestechlich.



Eindeutigkeit: Kurven müssen aus der Sicht des LookUp Parameters eindeutig sein. Im nebenstehenden Bild würden sich bei einer Vorgabe des y-Wertes zwei x-Werte ergeben. Welcher jetzt von der Funktion zurückgegeben wird ist unbestimmt. Ähnliches gilt auch umgekehrt für den x-Vorgabe Wert. Sättere Implementation der AVRco LookUp Funktionen werden eine Bereichsvorgabe für die LookUp Funktionen erlauben.

Generierung:

Ist die Kurve fertig konfiguriert, so kann das binär File erzeugt werden. Hierzu sollten noch die Skalierungs Faktoren, Offset, Anzahl der Punkte und der Kurven Ausschnitt vorgegeben werden. (**Export** Dialog)

Der **Ausschnitt** aus der Kurve kann fast beliebig gewählt werden. Es ist allerdings nicht sinnvoll, die Grenzen ausserhalb der x-Anfangs und Endwerte zu legen. In diesem Fall werden nur Punkte innerhalb der x-Werte der Kurve exportiert. Ein zu kleiner Ausschnitt bringt später bei der Interpolation u.U. relativ ungenaue Werte.

Die Offsets und Skalierungs Faktoren müssen so gewählt werden, dass im Programm brauchbare Werte herauskommen. Wird der y-Wert zum Beispiel durch einen 10bit AD-Wandler gebildet, sollte sich das Resultat aus $((\text{Kurven-y-Wert} - \text{offset}) * \text{factor})$ auch im Bereich von 0..1023 bewegen. Ähnliches gilt auch für den x-Wert. Die Werte von **offset** und **factor** sind erheblich abhängig von der Gesamt Verstärkung des System, das heisst von der Kette vom Sensor über OP-Amps bis zum AD-Wandler und dessen Auflösung.

Die Anzahl der generierten Punkte wird durch die Vorgabe im Dialog Feld **point count** bestimmt. Hier muss ein Kompromiss eingegangen werden. Wünschenswert wäre eine möglichst grosse Zahl von Punkten, so dass der durch das lineare Interpolieren des Systems erzeugte Fehler möglichst Null wird. Dies bedeutet allerdings sehr grosse Tabellen mit entsprechend langer Suchzeit. Wieviel Punkte wirklich gebraucht werden, kann nur ein Praxis Test erweisen. Eine Obergrenze ist die Auflösung des Erfassungs Systems, z.B. AD-Wandler. Mit 10bits sind nun einmal nicht mehr als 1024 mögliche Eingangswerte möglich.

Grundsätzlich ist bei Genauigkeits Überlegungen zu fragen: welches Teil im System ist das ungenaueste und lässt sich auch nicht verbessern. Meistens fällt da der Sensor ins Auge. Wenn dieser grosse Toleranzen hat, macht es wenig Sinn innerhalb der Software auf 0.1% Genauigkeit hinzuarbeiten.

6 Source-Code-Control-System - SCCS

Ein immer wieder kehrendes Problem bei der Software Entwicklung ist die Pflege der Sourcen oder besser gesagt die Wiederherstellung eines Entwicklungsstands vom Datum dd.dd. Das kann notwendig werden, wenn eine Änderung in einer längst zu den Akten gelegten Version notwendig wird, oder wenn ein Projekt sich in eine total falsche Richtung entwickelt hat und man wieder zu einem früheren (alten) Stand zurückkehren muss.

Um diese Aufgaben bewältigen zu können, werden sogenannte Version Control Systeme für sehr viel Geld angeboten. Diese Systeme sind sehr aufwendig und komplex, arbeiten mit Datenbanken etc, was ihren Preis erklärt. Wenn man das Problem auf das wesentliche reduziert, auf Datenbanken und komplexe Verfahren verzichtet, kann man die Aufgabe mit relativ einfachen Mitteln trotzdem zufriedenstellend lösen.

6.1 Einführung Source-Code-Control-System

Die in die IDE PED32 von E-LAB integrierte Lösung lässt sich zwar nicht unbedingt mit einem professionellen System dieser Art vergleichen, erfüllt aber ihren Zweck.

Die Eigenschaften des E-LAB SCCS sind;

- die aktuell geladene Version eines Projekts inkl. aller gewünschten Unit und Include Files, die Projekt Einstellungen etc, können mit einem Tastendruck abgespeichert (eingefroren) werden.
- Alle Abspeicherungen erfolgen in eine spezielle Directory mit Datums und Uhrzeit Information.
- Die Dateien werden weder verschlüsselt noch komprimiert, so dass diese auch jederzeit einsehbar sind.
- Jede einzelne abgespeicherte Version wird in einer Liste aufgeführt und kann auf Wunsch wieder hergestellt werden, entweder in die Original Directory oder in eine beliebige andere Directory. Das Projekt ist sofort wieder verwendbar.
- Alle Operationen erfolgen aus der IDE PED32 heraus. Es werden keine externe Programme benötigt oder aufgerufen.
- Versionen werden auf Anforderung des Benutzers abgespeichert. Es gibt keine automatische Protokollierung von Datei Änderungen.

6.2 Arbeitsweise des E-LAB SCCS

Alle Sicherungen (Versionen) befinden sich in der Subdirectory **_SCCS_** innerhalb der Projekt Directory. Darin hat jede Version ihre eigene Directory, deren Namen aus **sccs_ +Datum +Uhrzeit** besteht. Beispiel:

```
sccs_020628_1729
```

Hierbei ist **020628** das Datum 2002.06.28 und **1729** die Uhrzeit 17:29. Damit ist es theoretisch möglich jede Minute eine neue Version abzuspeichern. In dieser Directory befinden sich alle die Dateien des Projects, die für die Versions-Sicherung ausgewählt wurden.

Dateien, die in einer Version gesichert werden sollen, müssen in einem Dialog erfasst werden. Eine automatische Einbeziehung aller verfügbaren Dateien erfolgt nicht. Allerdings können diese Dateien auch in einer von dem Projekt unterschiedlichen Directory liegen. In diesen Fall verfährt das System bei der Wiederherstellung unterschiedlich.

Wird bei der Wiederherstellung als Ziel-Directory die Original-Directory des Projekts angegeben, werden alle gespeicherten Dateien an ihre original Positionen kopiert und evtl. dort schon/noch existierende mit gleichem Namen überschrieben.


Wird bei der Wiederherstellung als Ziel-Directory eine andere als die Original-Directory des Projekts angegeben, werden alle gespeicherten Dateien in diese Ziel Directory kopiert, ohne Rücksicht darauf, ob die ursprünglichen Dateien auf unterschiedliche Directories verteilt waren.

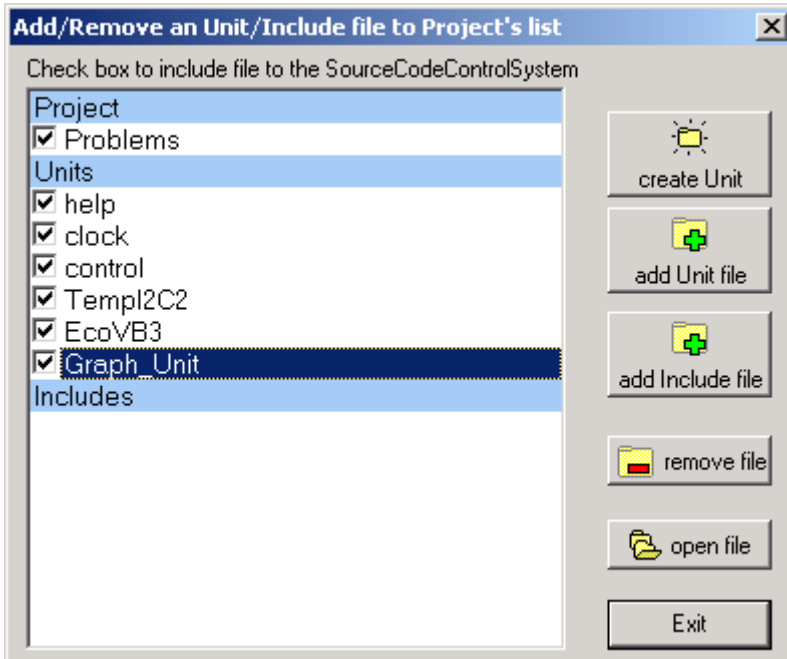
In beiden Fällen werden, falls notwendig, auch die Datei-Pfade in den Projekt Steuerdateien ‚*dddd.ppr*‘ entsprechend verändert.

Und so funktioniert es ...

Alle Operationen des SourceCodeControlSystems (SCCS) beziehen sich immer auf das geladene Projekt.

6.2.1 Version abspeichern

Soll der aktuelle Stand eines Projekts als Version abgespeichert werden, muss zumindest einmal eine Auswahl der zu sichernden Dateien getroffen werden. Das geschieht mit dem  Button, worauf der unten stehende Dialog geöffnet wird.

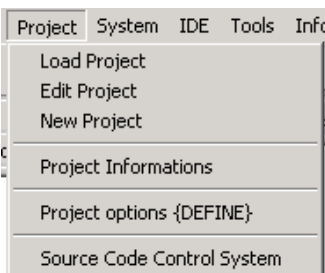


Mit **add Unit file** oder **add Include file** wird eine Datei dem Projekt hinzugefügt, wobei diese Datei nicht unbedingt in der Projekt Directory liegen muss. Unit Dateien müssen die Endung **.pas** haben und Includes die Endung **.inc**

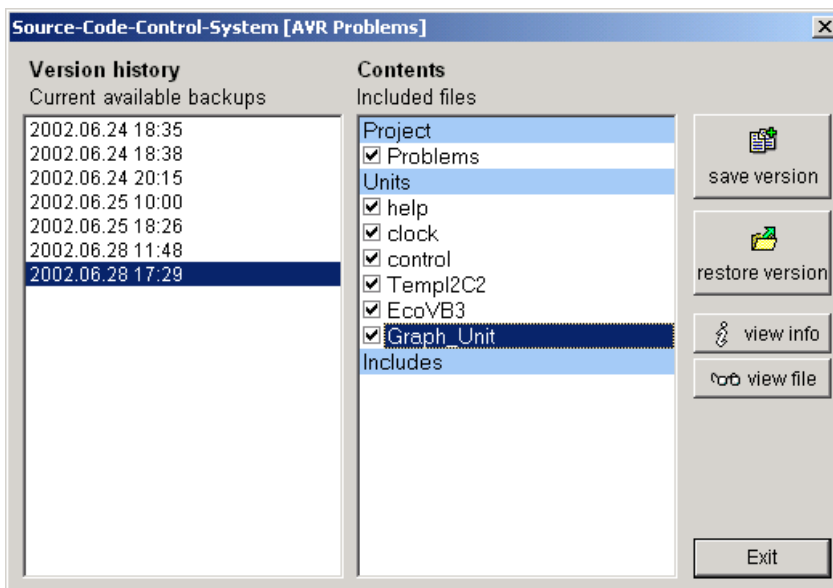
Mit **remove file** wird die Datei die den Highlight Balken hat, wieder aus der Liste entfernt.

Damit ist der Projekt Verwaltung bekannt, welche Dateien zu diesem Projekt gehören. Da es jedoch nicht immer sinnvoll ist, alle diese Dateien auch in eine Versions Sicherung mit einzubeziehen, muss mit der Checkbox einer jeden Datei das SCCS System informiert werden, dass diese Datei auch mit kopiert werden soll.

Damit sind die Vorarbeiten für die Versions Sicherungen erledigt. Im allgemeinen muss an diesen Einstellungen des Projekts nichts mehr geändert werden.



Soll die aktuelle Version abgespeichert werden, muss mit dem Menüpunkt **Source Code Control System** der dafür zuständige Dialog geöffnet werden.

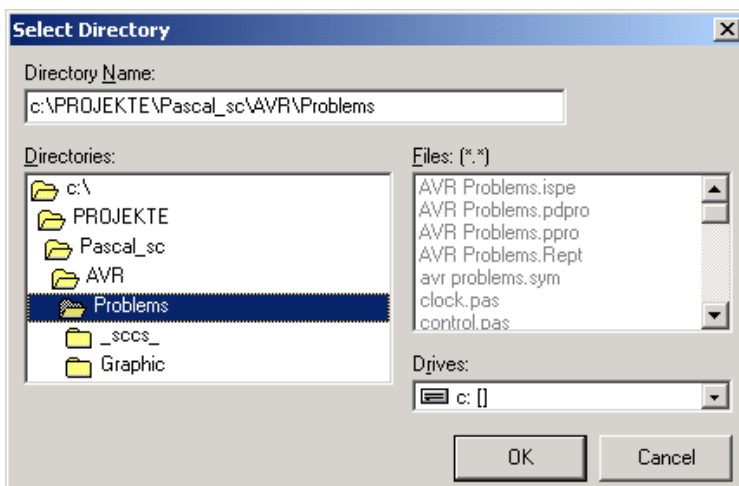


In diesem Dialog werden alle schon gespeicherten Versionen des aktuellen Projekts aufgelistet.

Ein Click auf einen Backup Eintrag im linken Feld listet die darin enthaltenen Dateien im rechten Feld auf. Jedem Backup ist ein Info zugeordnet, das mit dem **view info** Button angezeigt wird. Jedes enthaltene File kann mit dem **view file** Button angezeigt werden. Mit dem Button **save version** wird die aktuelle Version des Projekts mit Datum, Uhrzeit und allen gewünschten Dateien sofort abgespeichert. Jetzt geht noch ein Editor Fenster auch, in dem man eigene Kommentare zu dieser Version eingeben kann. Die neue Version erscheint jetzt auch in der linken Liste.

6.2.2 Version wieder herstellen

Soll ein früherer Stand (Version) eines Projekts wieder hergestellt werden, muss im obigem Dialog der Button **restore version** angeklickt werden. Jetzt erscheint untenstehender Dialog:

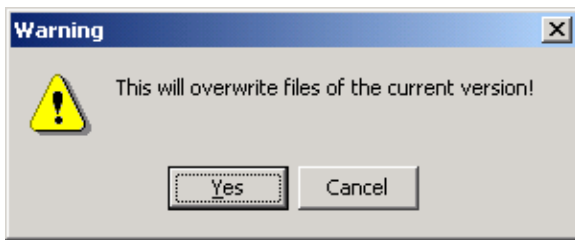


Er zeigt automatisch in die Original Directory des Projekts.

Es bestehen jetzt grundsätzlich zwei Möglichkeiten die Wiederherstellung durchzuführen.

1. Restore in die original Directory
2. Restore in eine andere, existente oder neue Directory.

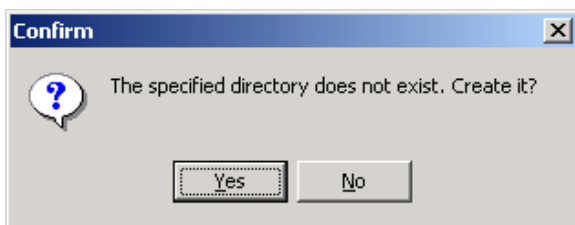
6.2.2.1 Original Directory



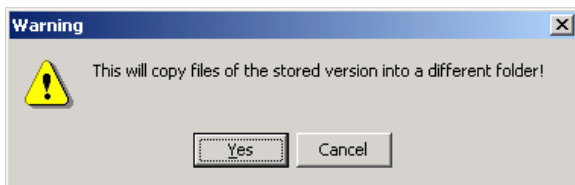
Der einfachste Weg. **Ok** Button des Directory Dialogs klicken.

Es erscheint als Information dieser Dialog
Mit dem **Yes** Button wird die Operation gestartet, mit dem **Cancel** Button wird alles abgebrochen.

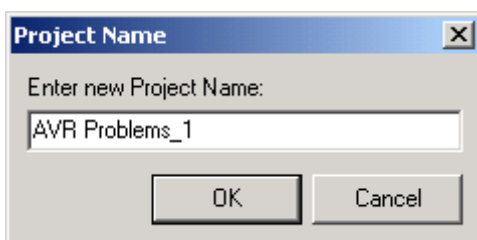
6.2.2.2 Neue oder andere Directory



Hierzu muss im obigen Directory Dialog im obersten Edit Feld der gewünschte neue Pfad bzw. Directory des Projekts angegeben werden. Mit dem **Ok** Button wird das bestätigt. Existiert die Directory noch nicht, erscheint nebenstehender Dialog. Dann muss die Erstellung der Directory mit dem **Yes** Button bestätigt werden.



In beiden Fällen muss die Kopier Operation mit dem nachfolgendem Dialog nocheinmal bestätigt werden.




Da ein neues Projekt mit einer gespeicherten Version des aktuellen Projekts erstellt worden ist, muss dieses neue Projekt auch einen Namen haben, der eindeutig ist. Das System schlägt dazu einen Namen in nebenstehendem Dialog vor. Dieser Name kann verändert werden, darf aber noch nicht als Project Namen schon in Verwendung sein.

Damit ist auch diese Operation beendet.

7 Flash Down Loader / Writer

In der Installation von AVRco ist ein PC-gestütztes DownLoader Programm **FlashLoader.exe** enthalten. Dies ist in die IDE PED32 eingebunden.

Ein Click auf den  Button startet das Programm.

Dieses Tool geht von einer seriellen Schnittstelle auf dem Target aus. Hiermit kann eine neue Applikation in das Zielsystem programmiert werden, wenn dort der Loader Monitor installiert ist. Allerdings muss im Zielsystem auch dafür gesorgt werden, dass zu diesem Zeitpunkt der Loader aufgerufen wurde. Im Beispiel Programm **..E-Lab\AVRco\Demos\SelfProg...** geschieht dies z.B. durch Drücken zweier bestimmter Tasten des Keyboards.

Das PC-Programm kann natürlich auch unabhängig vom der IDE bzw. Editor benutzt werden.

Hierzu gibt es zwei optionale Kommando-Zeilen Parameter:

1. FileNamen mit Pfad (optional). Filename kann ein Hexfile (.hex oder .eep), ein Proj-File (.ppro), ein Pack file (.pack) oder ein Encr File (.encr) sein.
2. FlashLoader ID (optional). Der Parameter beginnt mit einem % und besteht aus der dezimalen ID 0..65536. Wird diese ID vorgegeben, gibt das Programm eine Warnung heraus, wenn diese Vorgabe nicht mit der ID im Zielsystem übereinstimmt.

Wenn ein Flash Download erfolgt, löscht das Tool immer zuerst die letzte Page unterhalb des BootLoaders. Befindet sich hier eine OK-Kennung, so wird diese ungültig und erst wieder gültig, wenn der Download erfolgreich war. Das kann durch den Bootloader abgeprüft werden, wenn der Reset der CPU sofort in den Boot geht. Siehe Beispiel weiter unten.

Das Programm bedient sich des nachfolgenden Protokolls.

FlashLoader Kommando Liste

? Host fordert Loader Kennung an.
Loader antwortet mit
FD FlashDownLoader

A Host schickt page adr in word Darstellung. Alle Flash Aktionen beziehen sich auf diese Page
aa1 page adr loByte
aa2 page adr hiByte
aa3 wenn die CPU mehr als 128kBytes Flash hat muss noch dieses Page Extend Byte folgen
Loader antwortet mit
CR Befehl akzeptiert

B Host schickt EEprom adr in byte Darstellung und EEprom Data (byte).
aa1 EEprom adr loByte
aa2 EEprom adr hiByte
data 1 Byte ins EEprom
Loader programmiert dieses Byte in das EEprom und antwortet mit
CR Befehl akzeptiert

C Host schickt EEprom adr in byte Darstellung.
aa1 EEprom adr loByte
aa2 EEprom adr hiByte

Wenn ein (optionales) Passwort gesetzt ist (siehe weiter unten), muss der Host das Passwort schicken:

pw1 Passwort loByte
pw2 Passwort hiByte
Loader liest das EEprom byte an dieser Adresse im EEprom und schickt es als Antwort zurück



AVRco Tools

- D** Host lädt eine Page Daten zum Loader herunter
data es folgen (**ps** x 2) Bytes = Pagesize x 2 (mega8..meg16 = 128bytes)
Loader speichert diese Page in sein Array und bildet eine 8Bit Checksumme durch Addition aller Bytes.
Loader schickt die errechnete Checksumme als Ergebnis der Operation zum Host:
cc Checksumme
- E** Host fordert löschen der aktuellen Page
Loader löscht die aktuelle Page im Flash und antwortet mit
CR Befehl akzeptiert
- I** Host fordert Loader Info an.
Loader antwortet mit
I Info Kennung = Loader-ID siehe weiter unten
id1 hiByte Prozessor ID
id2 midByte Prozessor ID
id3 loByte Prozessor ID
ps words pro page
bs1 Bootblock start adr loByte \
bs2 Bootblock start adr hiByte / = Bootblock start adr in word count
bs3 Bootblock start adr extbyte / = Bootblock start adr in word count wenn Flash > 128kB
- P** Host fordert Programmieren des Zwischenspeichers ins Flash
Loader überschreibt die aktuelle Page im Flash mit dem Inhalt des Zwischenspeichers. Ein Verify findet nicht statt. Dies kann der Host durch zurücklesen des Flash in den Zwischenspeicher mit dem R-Kommando und anschliessendem Upload zum Host mit dem U-Kommando selbst tun.
Der Loader antwortet mit:
CR Befehl akzeptiert
- R** Host fordert Lesen der aktuellen FlashPage in den Zwischenspeicher an.
Loader liest die aktuelle Page aus dem Flash und kopiert sie in den Zwischenspeicher.
Der Loader antwortet mit:
CR Befehl akzeptiert
- U** Host fordert den Inhalt des Zwischenspeichers an.
Wenn ein (optionales) Passwort gesetzt ist (siehe weiter unten), muss der Host das Passwort schicken:
pw1 Passwort loByte
pw2 Passwort hiByte
Der Loader schickt (**ps** x 2) Bytes = Pagesize x 2 (mega8..meg16 = 128bytes).
data
und anschliessend noch eine 8Bit Checksumme, gebildet durch Addition aller Bytes.
cc Checksumme
- W** Host schickt eine relative page adr in word Darstellung. Darauf folgt ein word, das in den Zwischenspeicher in diese Adresse geschrieben werden soll.
aa page adr relativ
ww1 loByte
ww2 hiByte
Loader antwortet mit
CR Befehl akzeptiert
- X** Ende der Kommunikation. Der Loader Monitor springt die Anwender Prozedur *FlasLoaderExit* an.
Wird diese Prozedur nicht gefunden, beendet der Flashloader mit einem JUMP zur Adresse 0000h.
Keine Antwort vom Loader.

Loader Identifikation

Das Loader Programm wird in aller Regel nur einmal in die CPU programmiert und bleibt dort unverändert. Es bietet sich deshalb an, hier auch die Hardware Revision o.ä. des Boards/Systems unterzubringen. Diese 16bit Nummer kann vor dem Download abgefragt werden, um sicherzustellen, dass die Download Firmware auch auf dieser Hardware lauffähig ist. Diese Nummer wird bei der BootLoader Generierung mit in den Maschinen Code des Loaders selbst als immediate Konstante eingebracht. Das geschieht durch die Definition einer globalen Konstante mit dem Namen „DownloaderID“:

const

```
DownloaderID : word = 10213;
```

Wird diese Konstante nicht definiert, wird dieser Wert als \$0000 im Loader abgespeichert. Die ID-Nummer kann mit dem Befehl ‚i‘ vom Loader abgefragt werden.

i Der Host fragt die DownLoader ID ab.
Loader antwortet mit
ww1 ID-loByte
ww2 ID-hiByte

Es sind bis zu 4 ID-Words möglich, die dann alle sequentiell mit dem i-Kommando gesendet werden:

const

```
DownloaderID : word = $1234;  
DownloaderID1 : word = $5678;  
DownloaderID2 : word = $9ABC;  
DownloaderID3 : word = $DEF0;
```

Sicherheiten

Wenn verhindert werden soll, dass das Flash oder EEprom über den Loader von unbefugten ausgelesen werden kann, muss noch eine Passwort Konstante definiert werden:

const

```
DownloaderPWD : word = $1A2B;
```

Damit muss das Upload Tool mit dem Upload Kommando für das Flash und das EEprom auch immer dieses Passwort bereitstellen.

Weiterhin ist es dann sinnvoll die neue Firmware nicht im Hex-Format bereitzustellen, sondern im **Pack** oder **Encrypt** Format des E-LAB Programmier Tools. Der AVRco Downloader beherrscht auch diese Formate.



AVRco Tools

7.1 FlashLoader Beispiel

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
```




AVRco Tools

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;

AVRco Tools



©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
