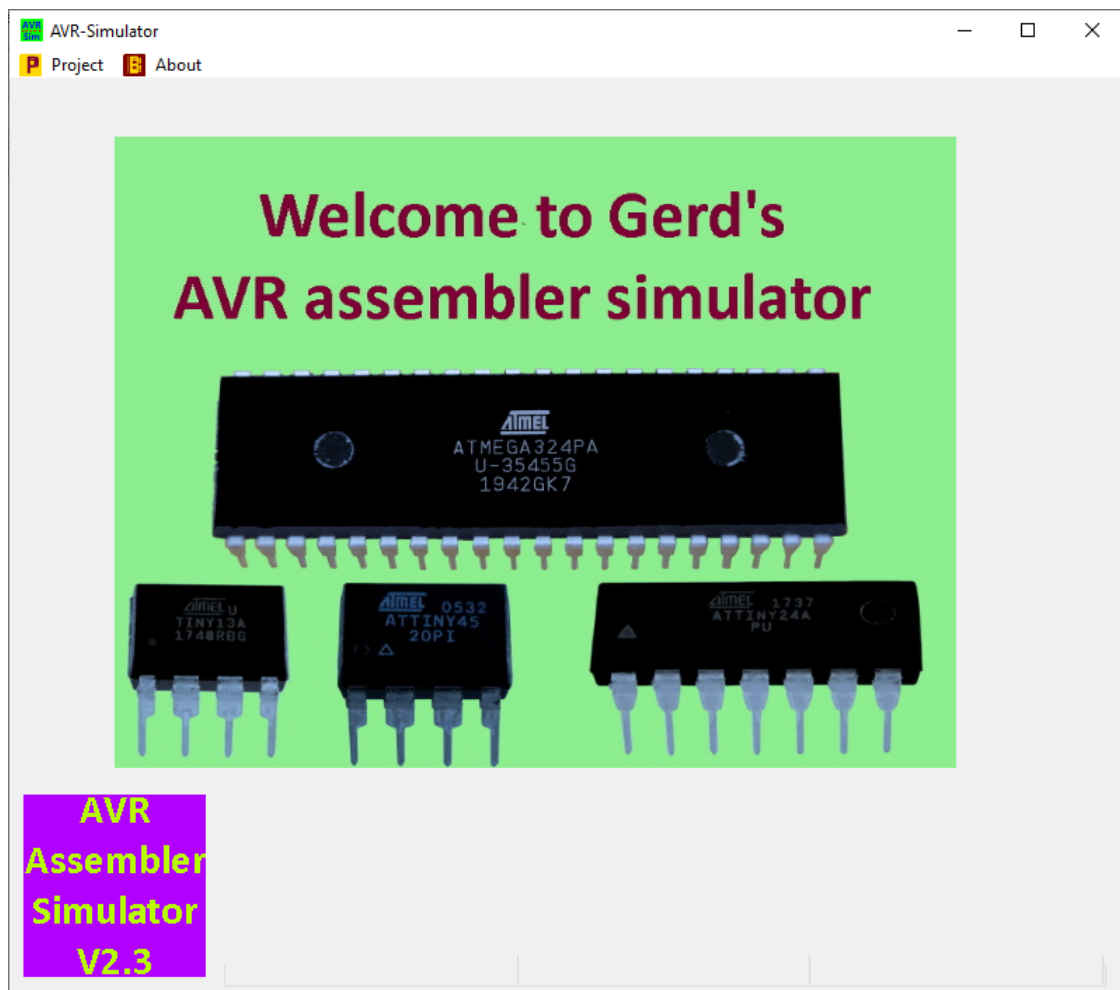


Installation and use of the the AVR assembler simulator `avr_sim`



by

Gerhard Schmidt, Kastanienallee 20, D-64289 Darmstadt

gavrasm@avr-asm-tutorial.net

Version 2.3 as of January 2021

Content

1	Application conditions and hints	
1.1	What this software does	3
1.2	Applicability	3
1.3	Hints for using avr_sim	4
1.4	Lengthy timer operations	5
2	Compiling avr_sim	
2.1	Installing Lazarus	6
2.2	Compiling avr_sim	6
3	Installation	
3.1	Installing gavasm	8
3.2	Running avr_sim for the first time	8
4	Use	
4.1	Opening files	9
4.2	Device check and package selection	10
4.3	Editing files	11
4.3.1	The editor	11
4.3.2	The editor's functions	12
4.3.3	The editor's context menu	13
4.3.4	Search for and/or replacing text	15
4.3.5	Saving files	15
4.3.6	Working in parallel with an external editor	15
4.3.7	Syntax highlighting	16
4.4	Selecting the device's clock frequency	18
4.5	Viewing device properties	19
4.6	Starting a new project	20
4.7	Adding a new include file to an existing project	25
5	Assembling	
5.1	Starting the assembly process	26
5.2	Errors during assembling	28
5.3	Setting/removing breakpoints	28
6	Simulating	
6.1	Starting simulation	31
6.2	Viewing I/O ports	32
6.3	Viewing and manipulating timers	35
6.3.1	Viewing timers	35
6.3.2	Changing timer values	37
6.4	The watchdog timer	37
6.5	Viewing and manipulating ADC content	39
6.5.1	ADC channels	39
6.5.2	Resistor matrix as voltage input source	39
6.5.3	Active AD conversion	42
6.6	Scope display	43
6.7	Viewing EEPROM content	45
6.8	Viewing SRAM content	46
6.9	Alerts	47
7	Not yet implemented, but under consideration	
8	Not implemented and not planned	
9	Changes introduced in version 2.3	
9.1	Changes to the integrated assembler gavasm	51
9.2	Changes concerning the editor	51
9.3	Changes concerning the simulator	52
9.4	Changes of the handbook	53
10	Known issues	

1 Application conditions and hints

1.1 What this software does

This software

- emulates micro controllers of the types AT90S, ATtiny, ATmega and ATxmega,
- allows to edit assembler source code written for these controllers,
- assembles the source code by running the integrated assembler gavrasm,
- simulates the generated program code as if it would run within the controller itself (in single steps or continuously),
- displays the state of the hardware inside the controller (ports, timers, AD converter, EEPROM, etc.) and allows to manipulate this.

1.2 Applicability

This software has limiting conditions as follows:

1. It is solely applicable for AVR assembler source code for AT90S, ATtiny, ATmega und ATxmega devices of ATMEL[®]/Microchip[®]. Other controller types such as PIC or ARM and their associated assembler dialects or compiler languages such as C are not supported.
2. It builds on relevant hardware properties of those controllers, such as the size and location of controller internals like flash memory, EEPROM or SRAM memories, availability and size of I/O-Ports, timers/counters, ADC channels, availability and addresses of interrupt vectors, type specific constants such as port register addresses and port bit names, etc. etc. It works only if the AVR device type is clearly specified, either with the directive `.include "(type abbrev.)def.inc"` or the directive `.device "(device name)"`. Sub types such as "A" or "P" suffixes are generally correctly recognized, and their hardware is correctly reflected, if there is an include file from ATMEL[®] or Microchip[®] and a device databook available.

The different availability of I/O port bits in different packages such as PDIP, SOIC or QFN are correctly recognized if the respective package type is selected.

3. It builds on the command line assembler gavrasm. Source code that is incompatible with this assembler can not be simulated and has to be converted (see the ReadMe file of gavrasm for the respective versions under this link: [gavrasm](#), also for hints on compatibility).
4. It is not tested systematically for all AVR controllers and all their internal hardware. There might be bugs (missing hardware, non-standardized symbols, etc.), especially with the older device types AT90S.
5. As like for all free software of this type no guarantee for bug-free functioning can be granted.

1.3 Hints for using avr_sim

The avr_sim executable is available in three sub-versions: one as Linux-64-Bit and two as Windows-64-Bit versions. The 64-bit-Windows version is available in the following two modes:

1. without integrated range checks and debug information (the executable file is much smaller and starts more rapidly),
2. with integrated range check and debug information ("debug" in the file-name, approx. 6 times larger).

If you encounter strange errors when working with Windows subversion 1, you can try the second version and identify internal programming errors, such as Range-Check errors. In that case please send feedback to me, with as-exact-as-possible information on the circumstances when the error reproduce-ably occurs, so that I can identify and fix the error.

If you need 32 bit versions for Windows and Linux see chapter 2.2 on how to compile avr_sim under Lazarus.

With avr_sim you can execute any AVR assembler source code (assumed that the conditions in chapter 1.2 are met). It assembles the source code with gavrasm to an executable flash hex code in standard .hex format and to a

standard EEPROM hex file .eep and steps or runs through the executable instructions.

The .hex and .eep files are in INTEL hex standard format, so can be used to feed flash and EEPROM burner software as is.

If only parts of the code shall be tested under avr_sim, unused parts of the source code can either be skipped (for single lines) or can be commented out with ";" in case of smaller parts. Larger code parts not to be simulated can be disabled with the directives ".if (condition)", ".else" and ".endif". These parts of the code can be selected and separated from code that is not simulated but should be part of the final executable code for the controller.

1.4 Lengthy timer operations

When simulating timer operations with large prescaler values simulation can last very long. Stopping at certain timer values, if no interrupt is executed, can be achieved by exchanging the *SLEEP* instruction by additional code that detects certain conditions (e. g. timer value TCNT = 255) and where breakpoints can be set to stop the running simulation.

2 Compiling avr_sim

If you have downloaded the pre-compiled version for Windows and Linux as executable (avr_sim_vv_Win64.zip or avr_sim_vv_Lin64.zip or the respective debug version) you can skip this chapter and proceed with the [Chapter on Installation](#).

If you downloaded the source files (Windows: avr_sim_vv_win_src.zip, Linux: avr_sim_lin_vv_src.zip) the following describes the compilation of those source files.

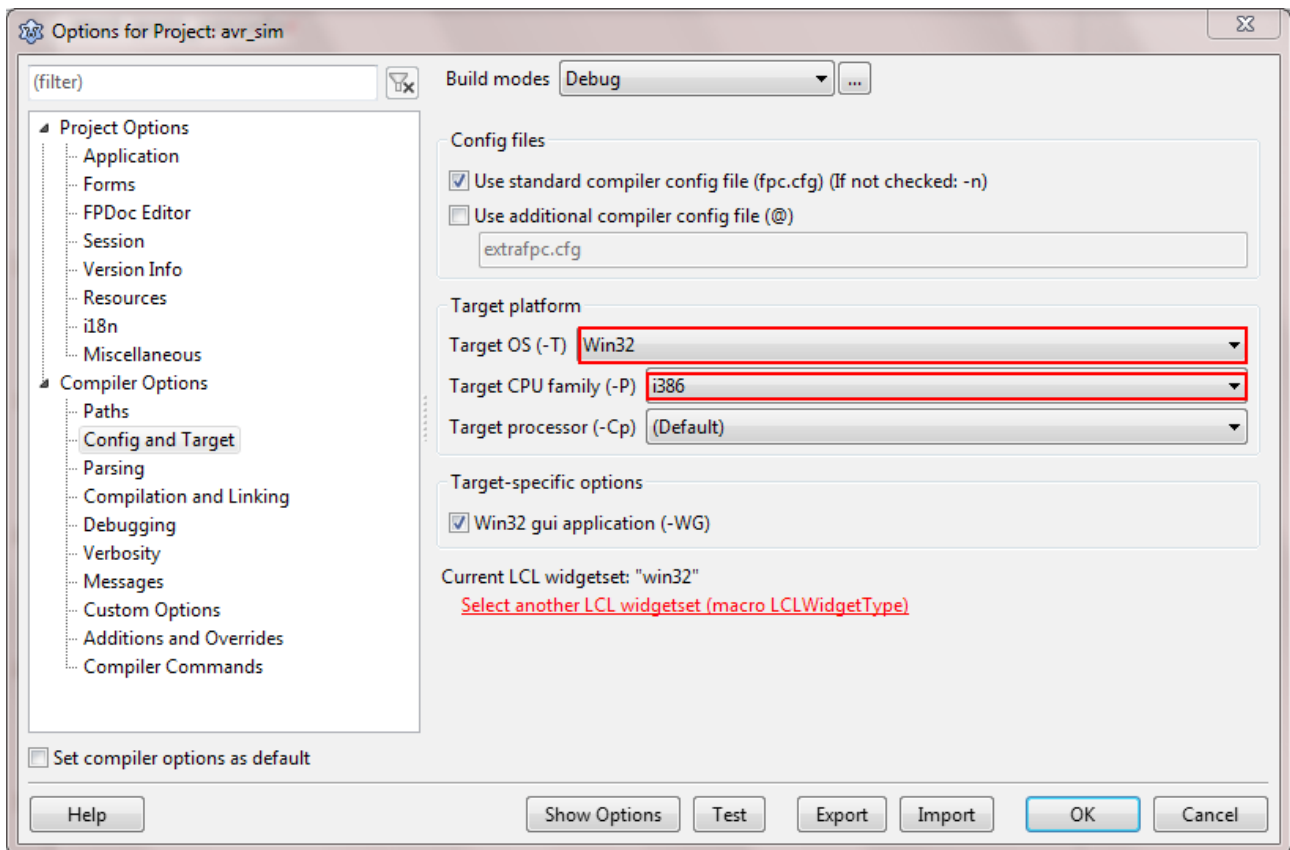
2.1 Installing Lazarus

To compile avr_sim you need the Open Source software Lazarus. Please download Lazarus and the corresponding Free Pascal Compiler FPC for your operating system. Lazarus is available for many different operating systems and their sub-versions and can be downloaded from their websites.

2.2 Compiling avr_sim

If you work under Windows or Linux use the respective source files for those operating systems. If you install for a different operating system, try out both source files or find a specific version compiled for your operating system on the internet.

Open the file "avr_sim.lpi" which should be associated with Lazarus source code. As first step change the target operating system and processor type. The default in Windows source code is Win64, in Linux Lin64. To change that, open "Project options" from the Lazarus menu and navigate to "Config and Target" in the "Compiler options". Choose your target operating system and your target CPU family from the two drop-down menus.



Picture courtesy of Dusan Weigel, modified

Also make sure that "Use standard compiler config file" is checked. Otherwise an error message of the type "Cannot find unit system" will occur if you compile with activated debug switches in the section "Debugging".

Then go through the forms of avr_sim and change their design, if necessary. Click on START-COMPILE to generate the executable.

If you clicked on START-START instead and the source code has been compiled without errors, avr_sim starts and you will be asked for the standard path for AVR source code. This simply eases locating source code files and is executed only once.

If you clicked START-START and your message window lists errors of the type "UNDEFINED SYMBOL" and avr_sim.lpr opens in a tab, you can delete all files with an ".o" and ".ppu" extension from the source code directory. That provokes re-compilation of those sources and the error messages disappear.

3 Installation

3.1 Installing gaviasm

An extra version of gaviasm is not necessary any more, gaviasm is completely integrated into avr_sim. The integrated version is identical to the most recent stand-alone version of gaviasm.

3.2 Running avr_sim for the first time

If you start avr_sim for the first time it requires you to locate the path where you store AVR assembler projects (can have sub-paths for several different projects, a correct path eases selection of assembler projects). You'll have to select the path actively, so that its name appears in the respective folder line.

This path is stored in a text file named avr_sim.cfg, which, under Windows, is located in the folder C:\Users\[user_name]\AppData\Roaming\avr_sim. Under Linux it is located in the folder ".avr_sim" in the home directory of the current user. If you have already generated this config file with a previous version of avr_sim, this file will be moved there. The executable can be located in a folder where the user has no write permissions to, e. g. in Linux in "/usr/bin/".

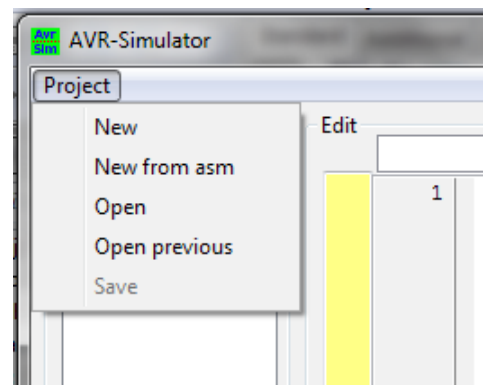
4 Use

avr_sim can handle assembler files (extension .asm of file type plain text) and avr-sim specific project files (extension .pro of file type plain text). If you open an assembler file (not an included file!) avr_sim generates a default project file for that and stores that in the folder where the assembler file is located.

4.1 Opening files

The menu item "Project" offers

1. to generate a "new" assembler project (this opens a form and generates a new assembler file – see [New project](#)),
2. to open an existing assembler source code file "New from asm", opens the assembler file in the editor window and generates a project file for that (by asking to overwrite an already existing one),
3. to "open" an existing project file, or
4. to "open a previously" generated project file.



Note that all include files, except the def.inc file, need to be located in the project folder. Relative paths, such as `.include "...\\location\\example.inc"` (resp. `../location/example.inc` in the Linux version) can be used.

avr_sim can start a given project file directly if you associate .pro files once with the avr_sim executable. Right-click on a .pro file, select "Open with ...", and "Select standard program", tick the entry field "Associate file type permanently" and navigate to the avr_sim executable. If you then click on a project file, avr_sim opens that project immediately. Danger in Windows: if you want to associate .pro files with a different version of avr_sim, do **not delete the old executable!** This would end up with a completely defective association. To repair this you need to edit the Windows Registry file. Another work-around is

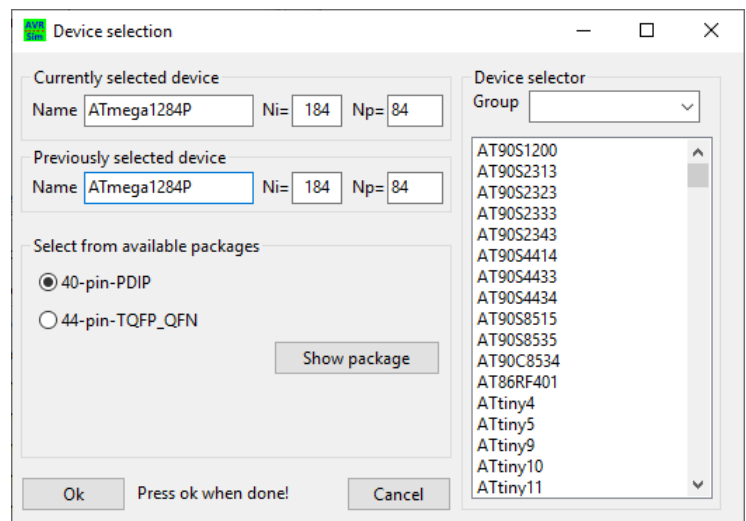
to rename the `avr_sim.exe` file to `avr_sim_nn.exe`: by doing that you can associate the `.pro` files with a certain version of `avr_sim`.

In the same manner `avr_sim` can be associated with `.asm` files. Either decide to associate `.asm` files directly with `avr_sim` or, with a right-click, select "Open with ..." to start the `.asm` file with `avr_sim`. If a project file already exists you are asked if you want to overwrite this. If yes, potential info in that is lost.

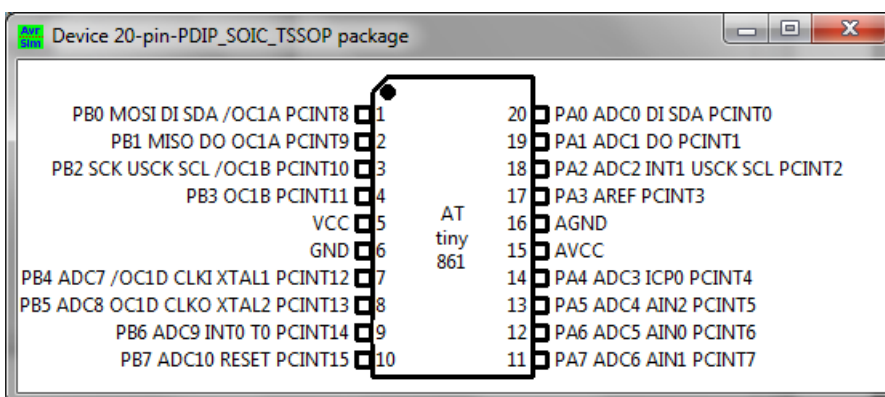
4.2 Device check and package selection

The device type is derived from the `.include "xxxdef.inc"` directive in the source code. In case that the device type is not included in the database of `avr_sim`, a selection window opens with "0" as device number.

If the AVR type is available in different packages, the dialog on the right side opens. The available packages are offered for selection in the group "Select from available packages" (here: 40-pin-PDIP and 44-pin-TQFP-QFN).



If you want to see the pin assignments of the package, click on the button "Show package" and you'll get a picture of the device type in the selected package.



The selected package type displays port pins, counter in- and outputs, `INTn-` and `PCINTn` inputs, a.s.o..

Close the package se-

lection with the "Ok" button if your selection is finished.

The selected device and package type are stored in the `.pro` file, so you are

only asked once for the package. If you want to select another package type, just click on "Target device:" in the project window or delete the respective line from the .pro file with a text editor and the dialog re-appears at project start.

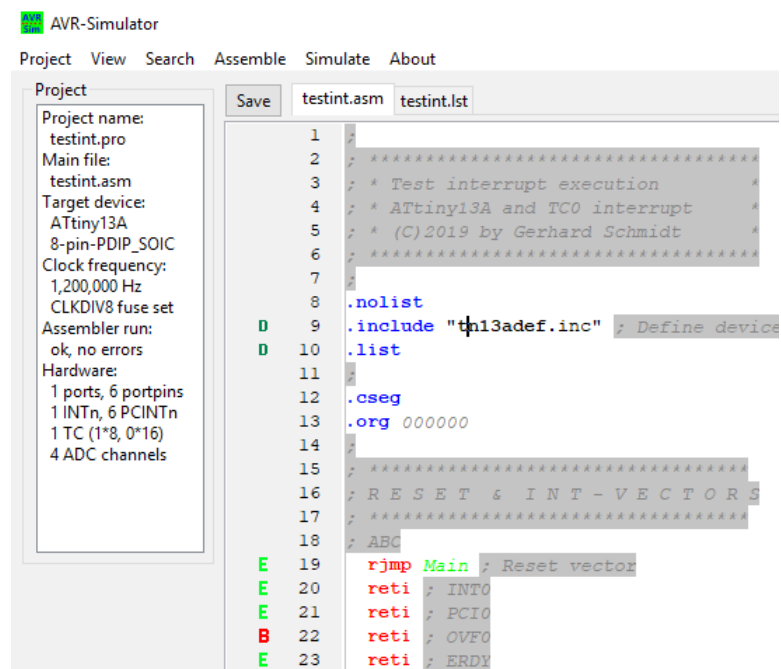
If you want to change the previously selected type and/or package, this can be done with that dialog. Just enter the device name manually in the "name" editor field of the section "Currently selected device". Devices with parts of the currently edited name appear in the list, from which devices can also be selected by clicking on them. If you want to extend the list, use the "Group" drop-down-field above the list. Clicking selects the type and ends manual input. If you are manually inputting the type and the edit field is yellow, hit the return key to end editing. If you want to undo editing, just click into the "Name" field below in the section "Previously selected device", which restores the previously selected device and package.

If you change the type of device, the .include line in the source code that defines the type is also changed and the cursor of the editor is set to this line. And, of course, the project file is rewritten and re-loaded.

4.3 Editing files

4.3.1 The editor

Any of the above selections opens the main assembler file in the editor. Files that are included (except the standard def.inc files for the device) are select-able by tabs. Changing the selected tab opens the respective include file.



If changes were made to the file either press the "Save" button or select a different file and answer "Yes" to save the file (if "No", the changes are ignored). Unlike other software, edited files in avr_sim are not kept internally. They are

reloaded whenever the tab changes. Therefore you are always asked if you want to save the changes whenever you change a file's content and you change the tab. This allows to use external editors on files that are currently not loaded into the editor window, such as included files (see chapter 4.3.6 for more details).

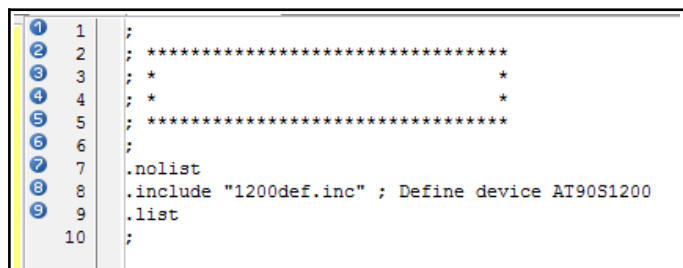
4.3.2 The editor's functions

The editor allows to perform the following useful functions with those key assignments:

- Alt-Backspace or Ctrl-Z: undo the last change,
- Ctrl-C: copy the marked text to the clipboard,
- Ctrl-V: insert the text in the clipboard at the current cursor location,
- Ctrl-A: mark the whole text,
- Ctrl-X: cut out the marked text,
- Shift-key Ctrl-N: set the markers N (1..9),
- Ctrl-N: go to the markers 1 to 9,
- Ins: toggle between overwriting and inserting.

All key functions of the editor are displayed in a window if the function key F1 is pressed.

A convenient feature are the bookmarks that can be set with Shift-Ctrl-1 to Shift-Ctrl-9. Those positions in the source code can be reached immediately with Ctrl-1 to Ctrl-9.



Depending from the operating system the Shift-Ctrl-0 combination does not work, if it is reserved by other software, but all other combinations work fine.

All edited files have their own bookmarks, those are stored in the project file and are restored when the project is loaded.

The function key F2 inserts the currently selected device package as ASCII text at the current cursor position. Only the port names of the pins, not their alternative pin functions are displayed. If you saved the device's pin assignments during the device selection process, the associated assignments are used (with small letters instead of all capital letters).

```

71 ; *****
72 ;   H A R D W A R E
73 ; *****
74 ;
75 ; Device: ATmega8, Package: 28-pin-PDIP
76 ;
77 ;
78 ;       1 /          |28
79 ; RESET o--|RST      PC5|---o not used
80 ;   D0 o--|PD0       PC4|---o LE5
81 ;   D1 o--|PD1       PC3|---o LE4
82 ;   D2 o--|PD2       PC2|---o LE3
83 ;   D3 o--|PD3       PC1|---o LE2
84 ;   D4 o--|PD4       PC0|---o LE1
85 ;   VCC o--|VCC      GND|---o GND
86 ;   GND o--|GND      AREF|---o AREF
87 ; XTAL1 o--|PB6      AVCC|---o AVCC
88 ; XTAL2 o--|PB7      PB5|---o SCK
89 ;   D5 o--|PD5       PB4|---o MISO
90 ;   D6 o--|PD6       PB3|---o A3 / MOSI
91 ;   D7 o--|PD7       PB2|---o A2
92 ;   A0 o--|PB0       PB1|---o A1
93 ;
94 ;

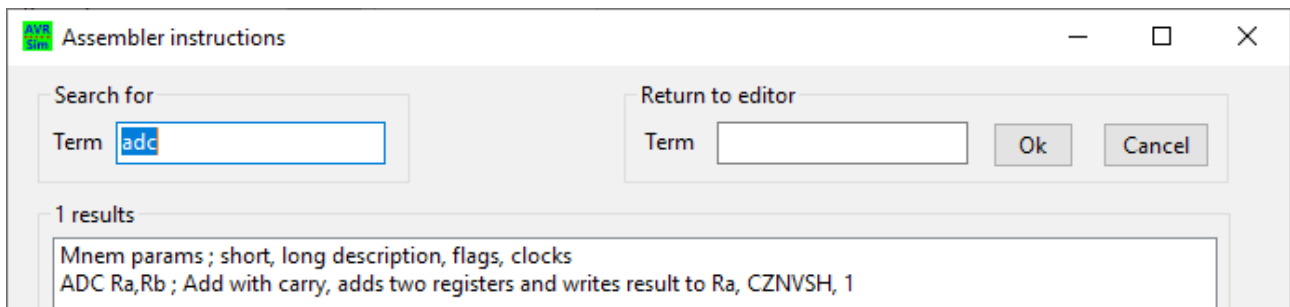
```

4.3.3 The editor's context menu

Right-clicking within the editor's window opens the context menu. Depending from the state that you are currently in, the context menu offers the following entries:

1. Editor help: opens a window where all editor keys and functions are displayed,
2. Break: set or clear a breakpoint in the currently edited line, visible only if the source code was at least once assembled and if the line has executable code,
3. Add include: add a new include file at the current cursor position,
4. Colors: opens the editor's highlighting configuration window, see below,
5. Instructions: opens a window where all AVR instructions are listed and can be searched for.

The search for instructions works as follows. If a text is marked when opening the instruction window (e. g. "adc" in this case), this text is searched for and lines with that text are displayed.



The text consists of the mnemonic and its parameters, a semicolon-separated short text plus a longer explanation what the instruction does, the flags in SREG that the instruction alters and the number of clock cycles consumed.

The following abbreviations are used for parameters:

- Ra, Rb: any register,
- Re: even register,
- Rh: registers R16 to R31,
- Rp: register pairs R25:R24, X(R27:R26), Y(R29:R28) and Z(R31:R30),
- Rxz: register pairs X, Y or Z,
- Ryz: register pairs Y or Z,
- Rm: registers R16 to R23
- b: bit 0 to 7,
- P: port 0 to 63,
- Pl: port 0 to 31,
- cNNN: constant within values from 0 to NNN-1 or from $-(\text{NNN}/2-1)$ to $\text{NNN}/2$,
- addr: address 16-bit,
- addr24: address 24-bit.

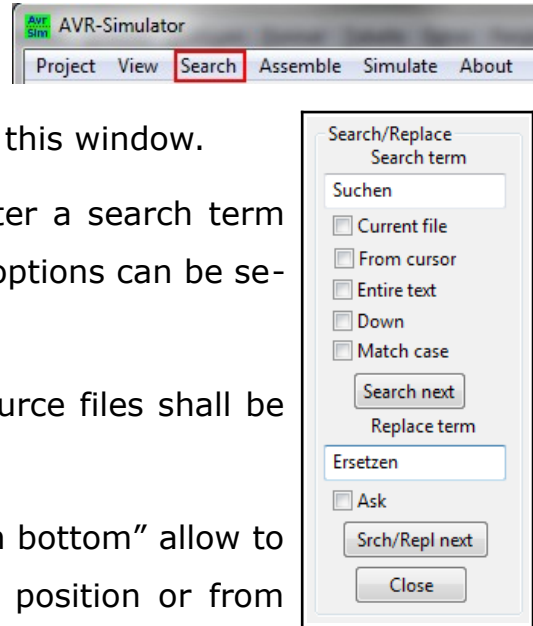
If you select one of the lines in the list field, the mnemonic and the parameters are copied to the return field and, when "Ok" is pressed, are inserted into the editor's text at the current position.

4.3.4 Search for and/or replacing text

The search and replace window opens by clicking on "Search" in the main window. Repeated clicking on this entry closes and opens this window.

The search and replace window allows to enter a search term and, if desired, a replace term. The following options can be selected:

- whether the currently selected or all source files shall be searched and replaced,
- "From cursor" or "From top" resp. "From bottom" allow to search/replace from the current cursor position or from its beginning or end,
- "Entire text" or "Selected text" limit the process for all or only for the selected part,
- "Down" resp. "Up" determine the direction,
- If "Match case" is selected, small and capital letters are discriminated.



When replacing, "Ask" means a Yes/No dialog while "Replace all" replaces without asking.

4.3.5 Saving files

Whenever you change the edited file by clicking another tab the edited file is saved (if any changes were made to its content). Before saving, the previous file content is copied to a file named [filename].[ext]~. If you have to go back to the previous content, use this file to reconstruct the previous content. Note that any tab change overwrites the previous content of both files.

4.3.6 Working in parallel with an external editor

If you sometimes work with an external editor please consider the following:

- avr_sim's editor loads the current versions of files whenever the tab selector changes. So having the editor displaying the list file, an external

editor can be used to change the .asm and any .inc files. After finishing external work on files, just change back to the tab of the desired file and the (externally changed) content is re-loaded. If large changes were made it might be useful to re-load the whole project and to re-assemble. This re-news the executable qualification of all files and ensures that breakpoint setting and removal works correct.

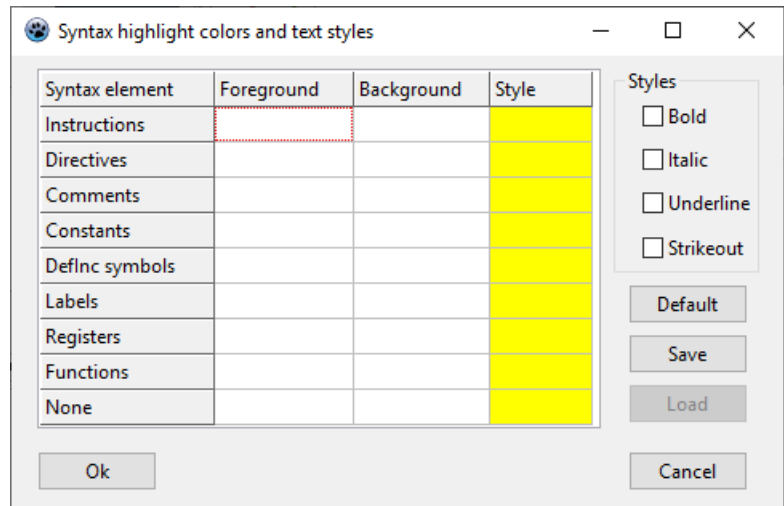
- If you want to keep the current content of an edited file while temporarily working on it externally, select the tab of this file and do the external work. If finalized, you'll be asked if you want to reload the externally changed file. If you answer No, the external changes are overwritten by the version in avr_sim's editor.
- Note that the Studio assembler 2 (in the last usable version 4.19) deletes, not by default but if selected, the .lst file and, if enabled, overwrites the list file using a different format. Listings that were not generated by gavasm are detected and re-assembling with gavasm is required. The more modern versions of the Studio (unusable versions such as 7.0) do not allow to generate assembler list files (at least I have not found this option), so these Studio versions cannot interfere with gavasm and avr_sim's list files.

The text editor used has to store plain text without additional formatting information such as ANSI-escape-sequences for text colors, etc. Assembler source code files shall only use 7-bit-ASCII characters, so do not use any country-specific character sets in your source code. The code that the editor produces shall be ANSI, other text formats such as UTF8 or others are not working correct.

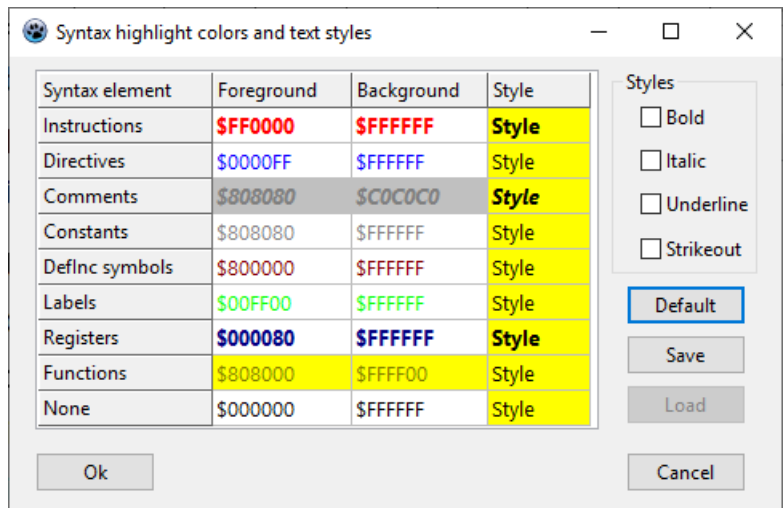
4.3.7 Syntax highlighting

From version 1.7 on the editor is able to highlight syntax elements in AVR assembler language. That means that the editor can display specific elements of the source code in selectable foreground and background colors as well as with selectable text styles.

A right-click within the editor field and selecting the entry "Colors" opens this window. For each syntax element listed the colors can be selected by clicking into the foreground or background column, which opens a color dialog. The text style can be changed by enabling and disabling the elements in the styles section and by clicking into the syntax element's style column.



By clicking onto the element's name (left column) the applicability of the entry is switched on and off. By clicking on the first line of the first column ("Syntax element") all entries are disabled.



The button "Default" sets all colors to a standard color set shown above. This leads to the displayed appearance.

Leaving this dialog with the "OK" button enables syntax highlighting, "CANCEL" disables it.

Please be aware that register names (with .def), constant symbols (with .equ or .set) and labels are only updated when enabling the syntax-highlighting. So, after larger changes to the source code switch

```

153 ; Reset- und Interrupt-Vektoren
154 ; (Vektoren beim ATmega16 sind Doppelworte!)
155
156 .CSEG
157 .ORG $0000
158     rjmp main ; Reset Vektor
159     nop
160     reti ; INTO Vektor
161     nop
162     reti ; INT1 Vektor
163     nop
164     reti ; TC2Comp Vektor
165     nop
166     reti ; TC2Ovf Vektor
167     nop
168     reti ; TC1Capt Vektor
169     nop
170     rjmp TC1CmpAIsr ; TC1CompA Vektor
171     nop
172     rjmp TC1CmpBIsr ; TC1CompB Vektor
173     nop
174     reti ; TC1Ovf Vektor
175     nop
176     rjmp TC0OvfIsr ; TC0Ovf Vektor
177     nop
178     reti ; SPI, STC Vektor

```

syntax-highlighting off and on again to update these lists. Note that the def.inc symbols are only updated during project loading.

Syntax-highlighting is disabled by default, but if you press the "Save" button and leave the dialog with "Ok" it writes the current color set to the avr_sim.cfg file and avr_sim starts with that set and enables syntax-highlighting. If a color set in avr_sim.cfg is available, you can read it with the "Load" button. If you leave the dialog with the "Save" and "CANCEL" button, the entry in avr_sim.cfg is deleted.

Each project also stores the current color and style set in its project file and re-stores the color set when re-opening the project file. The settings in the project file overwrite the settings in avr_sim.cfg, so each project can use its own color and style set.

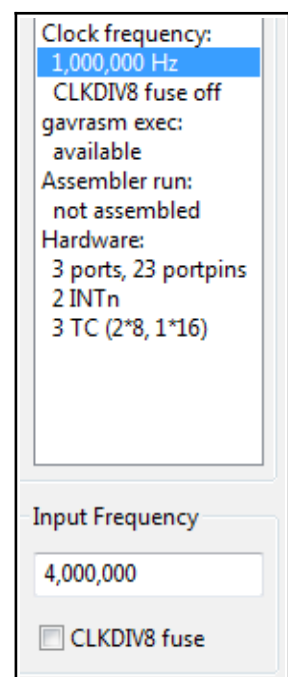
Please note that the bold style does not yet work as designed.

4.4 Selecting the device's clock frequency

The clock frequency at this stage is derived from device specific default data and is used in simulation. If an external Xtal or a clock generator is attached and activated by fuse settings, click on the line "Clock frequency:", input the desired frequency in Hz and enter it with "Return". That changes the clock frequency and is stored (and reloaded) in the project file.

Changes of the clock frequency come immediately into effect if the simulator is active. The display of the frequency in the simulator window and in the timer window is updated immediately.

If in your device the CLKDIV8 fuse is set, set the respective option, too. If, during runtime, the software changes the clock prescaler (by writing to the port register CLKPR) this change is recognized by avr_sim and the frequency is changed during simulation.

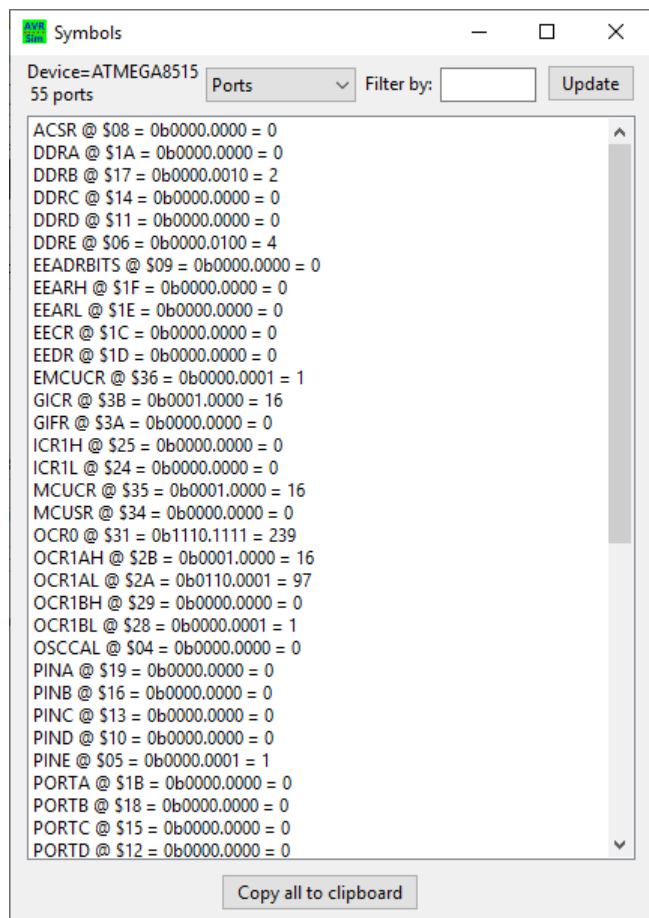
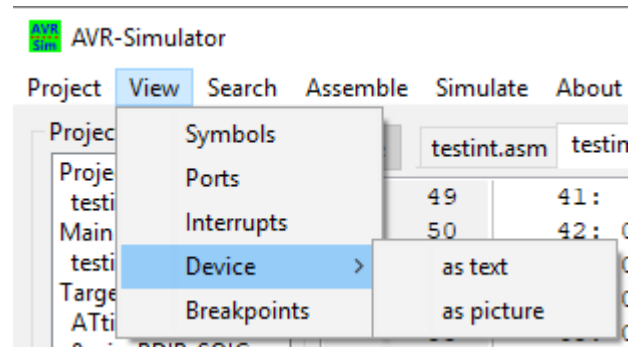
The image shows a settings dialog box for the AVR simulator. It has a light gray background and a thin black border. The top section contains several status indicators: "Clock frequency:" followed by "1,000,000 Hz" (highlighted in blue), "CLKDIV8 fuse off", "gavrasm exec: available", "Assembler run: not assembled", and "Hardware: 3 ports, 23 portpins, 2 INTn, 3 TC (2*8, 1*16)". Below this is a section titled "Input Frequency" with a text input field containing "4,000,000". At the bottom, there is a checkbox labeled "CLKDIV8 fuse" which is currently unchecked.

If your screen resolution allows a larger form, the main window with the editor can be re-sized. It then uses larger character fonts, too.

4.5 Viewing device properties

The menu item "View" offers to display

1. all symbols that are defined for that device in the def.inc (uses the gavasm internal symbols),
2. all ports that are defined for that device (subset of the symbol set) with their addresses and their current content in decimal and in binary format,
3. all interrupts that the device can handle (subset of the symbol set),
4. the selected device, either as text (with the pins and their assignments) or as picture,
5. if breakpoints are set, the breakpoint editor can be opened with this entry.
6. If the device type has been selected, the entry "Fuses" shows the fuses of that type. Note that this view is incomplete and shows only the signature bytes of the device.

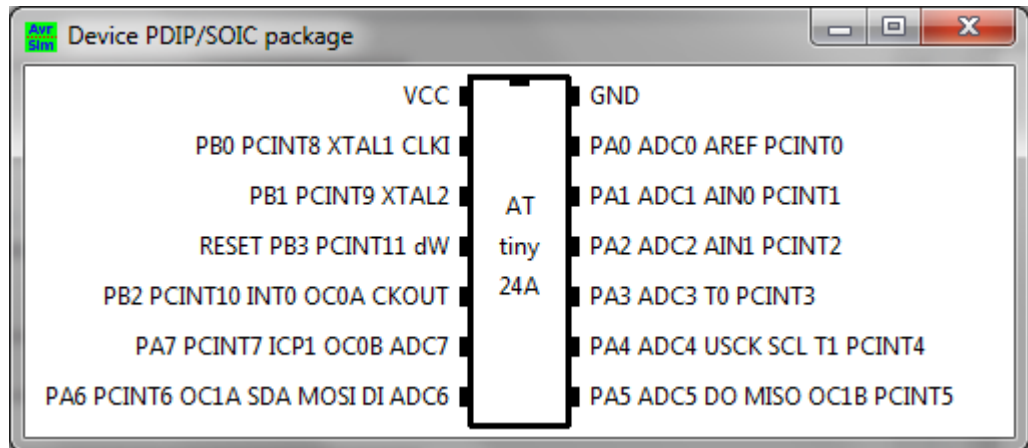


The view window for symbols, ports and interrupts offers a filter property so that a limited symbol set can be displayed.

During simulation the "Update" button allows to display the current content of the displayed ports dynamically.

If the "Device" entry does not show up, make sure that you have the directive `.include "typedef.inc"` or `.device "(AVR type)"` in your code, from which the device can be derived. As many basic information is depending from the device (memory sizes, internal hardware, etc.) simulation does not work in this case.

If "View", "Device" and "as picture" is selected from the "View" menu all pin assignments of the device



are shown for the device in its selected package. If a different package type has been selected, those are displayed accordingly.

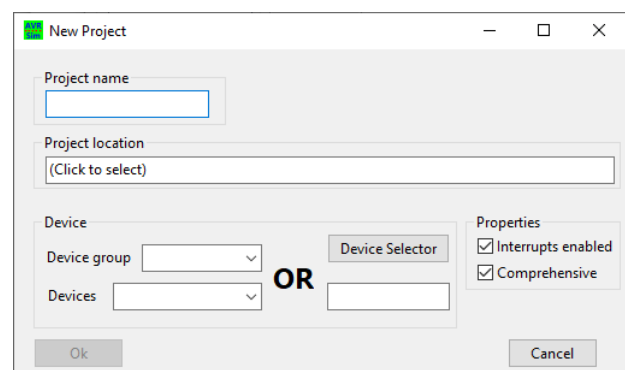
By clicking on the picture this can be saved either as a PNG or as a BMP graphics file.

The text entries in the symbol window can be copied to the clipboard with the button "Copy all to clipboard".

4.6 Starting a new project

By clicking "New" in the "Project" menu this window opens.

Select a project name and, by clicking into the project location entry field (and selecting a path for the project). If project name and path refer to an already existing file, an error message occurs.



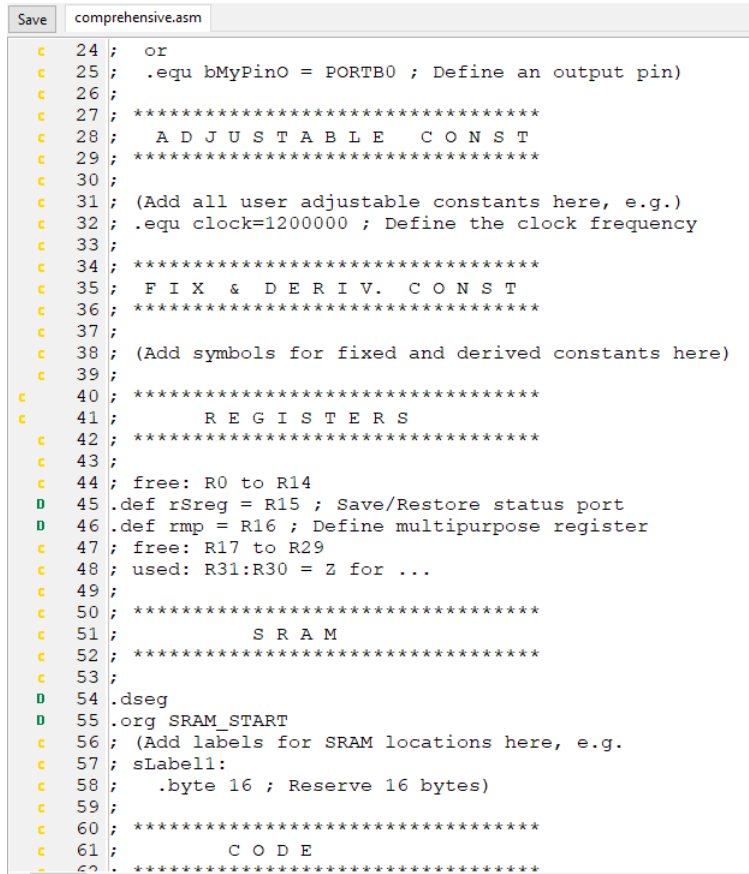
With the two property selections

- Interrupt or linear, and

- Comprehensive or short,

the type of the generated standard template can be pre-selected. If you select "Comprehensive", the generated source code file has a comprehensive structuring into sections. The single sections "Ports and Portpins", "Adjustable" and "Fixed constants", "Registers", "SRAM" etc. allow to structure the source code in such sections.

The generated code is lengthy (more than 100 lines) but allows to structure all information so you'll find it when necessary. Use this option for larger projects that require an overview, the necessary typing work is largely reduced.



```

Save comprehensive.asm
24 ; or
25 ; .equ bMyPinO = PORTB0 ; Define an output pin)
26 ;
27 ; *****
28 ; A D J U S T A B L E   C O N S T
29 ; *****
30 ;
31 ; (Add all user adjustable constants here, e.g.)
32 ; .equ clock=1200000 ; Define the clock frequency
33 ;
34 ; *****
35 ; F I X & D E R I V.   C O N S T
36 ; *****
37 ;
38 ; (Add symbols for fixed and derived constants here)
39 ;
40 ; *****
41 ;       R E G I S T E R S
42 ; *****
43 ;
44 ; free: R0 to R14
45 .def rSreg = R15 ; Save/Restore status port
46 .def rmp = R16 ; Define multipurpose register
47 ; free: R17 to R29
48 ; used: R31:R30 = Z for ...
49 ;
50 ; *****
51 ;       S R A M
52 ; *****
53 ;
54 .dseg
55 .org SRAM_START
56 ; (Add labels for SRAM locations here, e.g.
57 ; sLabel1:
58 ;     .byte 16 ; Reserve 16 bytes)
59 ;
60 ; *****
61 ;       C O D E
62 ; *****

```

If you selected “Interrupts”, too, the source code already holds all interrupt vectors of the selected device behind the .CSEG directive. The first entry for the RESET vector holds a jump instruction, while all other interrupt vector entries are equipped with a RETI to terminate the respective interrupt. Activating an interrupt therefore requires to add an interrupt service routine in the next code section and to replace the RETI in the interrupt vector list with a jump instruction.

```
Save comprehensive.asm
c 60 ; *****
c 61 ; C O D E
c 62 ; *****
c 63 ;
D 64 .cseg
D 65 .org 000000
c 66 ;
c 67 ; *****
c 68 ; R E S E T & I N T - V E C T O R S
c 69 ; *****
E 70 rjmp Main ; Reset vector
E 71 reti ; INT0
E 72 reti ; PCIO
E 73 reti ; OVFO
E 74 reti ; ERDY
E 75 reti ; ACI
E 76 reti ; OC0A
E 77 reti ; OC0B
E 78 reti ; WDT
E 79 reti ; ADCC
c 80 ;
c 81 ; *****
c 82 ; I N T - S E R V I C E R O U T .
c 83 ; *****
c 84 ;
c 85 ; (Add all interrupt service routines here)
c 86 ;
c 87 ; *****
c 88 ; M A I N P R O G R A M I N I T
c 89 ; *****
c 90 ;
L 91 Main:
E 92 ldi rmp,Low(RAMEND)
E 93 out SPL,rmp ; Init LSB stack pointer
c 94 ; ...
E 95 sei ; Enable interrupts
```

If you do not need structured sections, because you just need a brief quick and dirty template with only the most necessary stuff or if you do not like the default comprehensive structure, choose this option. It provides a very short template.

```
Save shortversion.asm
c 1 ;
c 2 ; *****
c 3 ; * (Add program task here) *
c 4 ; * (Add AVR type and version here) *
c 5 ; * (C)2019 by Gerhard Schmidt *
c 6 ; *****
c 7 ;
D 8 .nolist
D 9 .include "tn13adef.inc" ; Define device ATtiny13A
D 10 .list
c 11 ;
D 12 .cseg
D 13 .org 000000
c 14 ; *****
c 15 ; M A I N P R O G R A M I N I T
c 16 ; *****
c 17 ;
L 18 Main:
L 19 Loop:
E 20 rjmp loop
c 21 ;
c 22 ; End of source code
c 23 ;
c 24 ; (Add Copyright information here, e.g.
c 25 ; .db "(C)2020 by Gerhard Schmidt " ; Source code readable
c 26 ; .db "C(2)20 0ybg reahdrS hcimtd " ; Machine code format
c 27 ;
```

avr_sim uses AVR-type specific properties very intensively, so in the “New project” window you’ll have to specify the type. You can use the two drop-down windows to the left to select the type. Without this selection, the window lists missing type information.

From version 1.8 on the AVR type selection can be done with a device selector

window. In that window device types, packages and other hardware properties of the AVR can be selected by setting necessary properties of the device. The left list contains all devices that are in line with those settings. If you click on a device type in the left list, its pin assignments appear on the right list. If the device selected is fine, click on "Ok" to select it.

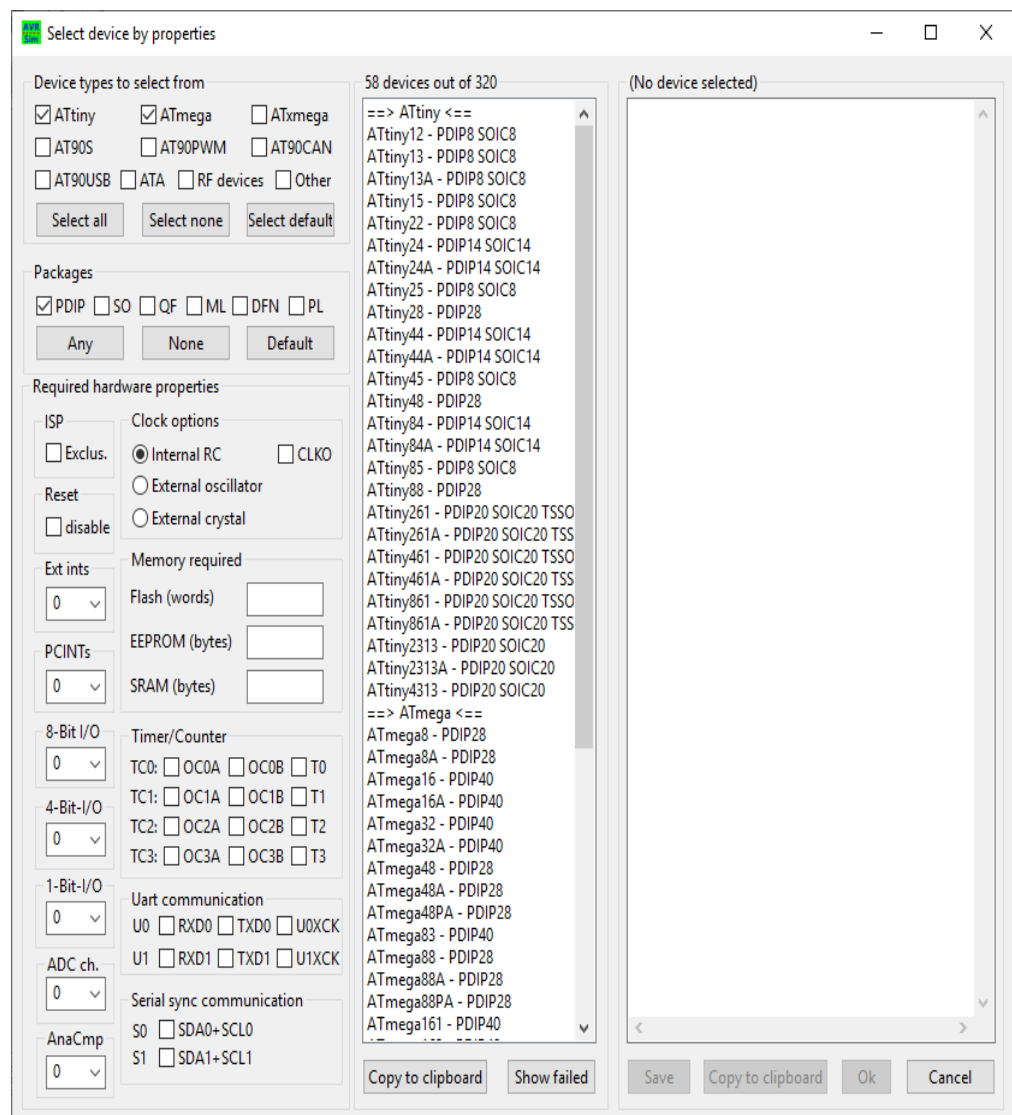
Select the device types to be selected from by ticking all those in the top selector window. The desired package types can be configured below.

The properties to select from are:

- "ISP Excl." fixes the pins for MOSI, MISO and (U)SCK exclusive use.

- "Reset disable" frees the reset pin from this task so that alternative functions can be used. Do that only if you do not program your AVR in ISP mode.

- The number of ex-



ternal interrupt pins INTn and PCINTn pins can be selected from the two respective drop-down fields.

- If you need complete 8-bit-ports, upper and lower 4-bit-ports and single pins for I/O, select those from the drop-down fields.
- Also if you need ADC channel input pins or analog comparer input pins (two per channel).
- Also click on one of the three clock options. This checks if the devices has internal RC oscillators and reserves respective pins in case of external clocks or crystals/resonators. If you additionally need the clock output pin CLKO, select this too.
- If you need flash, EEPROM and/or SRAM memory of a certain minimum size, input these in the edit fields. Decimal as well as hexadecimal numbers (preceded by 0x or \$) are accepted.
- Necessary pins for Timer/Counter output for TC0 to TC3 as well as input pins T0 to T3 for the timers can also be selected.
- If you need asynchronous or synchronous serial communication, select the necessary in- and output pins for that.

The list in the middle shows all device types that fulfill those hardware requirements in their respective package. The list can be copied to the clipboard. The button "Show failed" opens a window with all devices that were excluded from the list, with the main (and prioritized first) reason for exclusion. Package selection lines can be removed with the button. This window also displays the selection criteria in a short form. If you need these criteria, mark those lines and copy those with Ctrl-C to the clipboard.

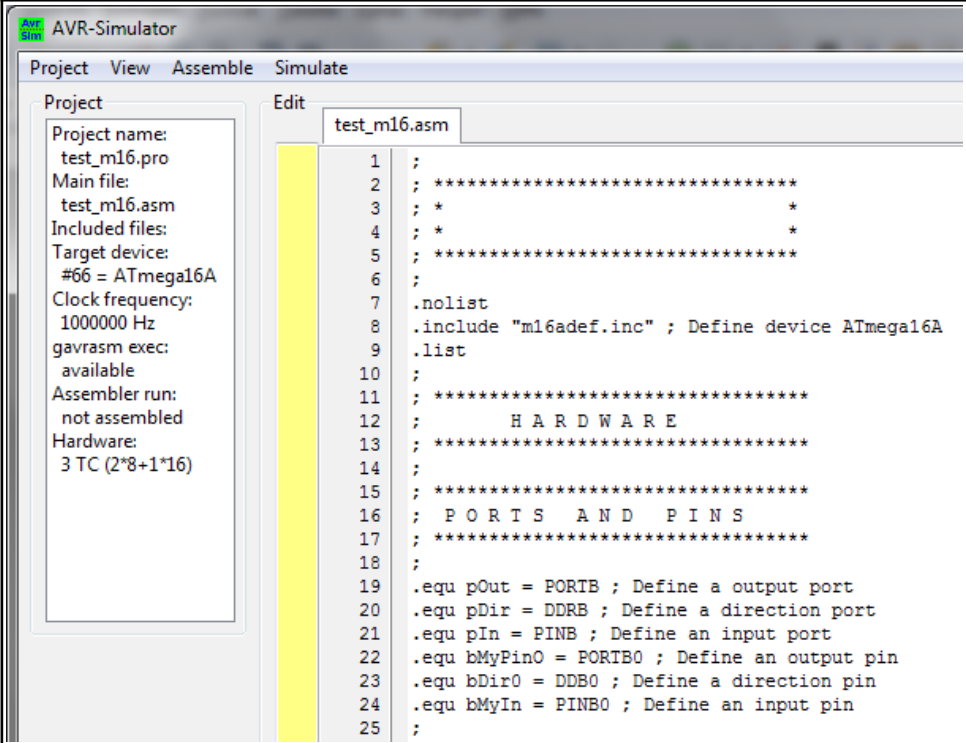
If a device is marked in the device list, its pin assignments appear in the window to the right. All pins that are assigned to one of the functions are listed with their assignment and all alternative pin functions are removed. The assigned function is shown in lower-case characters (e. g. "Vcc" instead of "VCC", "Reset" instead of "RESET", "Pcint3" instead of "PCINT3". All unassigned pins remain in capital letters. Those pin assignments can be saved with the "Save" button, which creates a file named "hw_[avr-type]_[package].txt" in the project folder. When inserting the ASCII representation of the device with F2,

those assignments are kept. The window content can also be copied to the clipboard. Clicking "Ok" selects the device, "Cancel" returns without selection. Both buttons close that window. This returns the control to the "New" dialog.

If you filled in all necessary details in the "New" dialog and error messages disappear, press the "OK" button.

This opens the dialog for the package selection, if the selected device has more than one available packages. The package dialog is described below.

The generated standard file can be edited, saved, assembled and simulated. This is



The screenshot shows the AVR-Simulator window. On the left is the 'Project' dialog with the following details:

- Project name: test_m16.pro
- Main file: test_m16.asm
- Included files:
- Target device: #66 = ATmega16A
- Clock frequency: 1000000 Hz
- gavrasn exec: available
- Assembler run: not assembled
- Hardware: 3 TC (2*8+1*16)

On the right is the 'Edit' window showing the source code for 'test_m16.asm':

```
1 ;
2 ; *****
3 ; *
4 ; *
5 ; *****
6 ;
7 .nolist
8 .include "m16adef.inc" ; Define device ATmega16A
9 .list
10 ;
11 ; *****
12 ; H A R D W A R E
13 ; *****
14 ;
15 ; *****
16 ; P O R T S   A N D   P I N S
17 ; *****
18 ;
19 .equ pOut = PORTB ; Define a output port
20 .equ pDir = DDRB ; Define a direction port
21 .equ pIn = PINB ; Define an input port
22 .equ bMyPinO = PORTB0 ; Define an output pin
23 .equ bDirO = DDB0 ; Define a direction pin
24 .equ bMyIn = PINB0 ; Define an input pin
25 ;
```

such a typical source code file generated.

4.7 Adding a new include file to an existing project

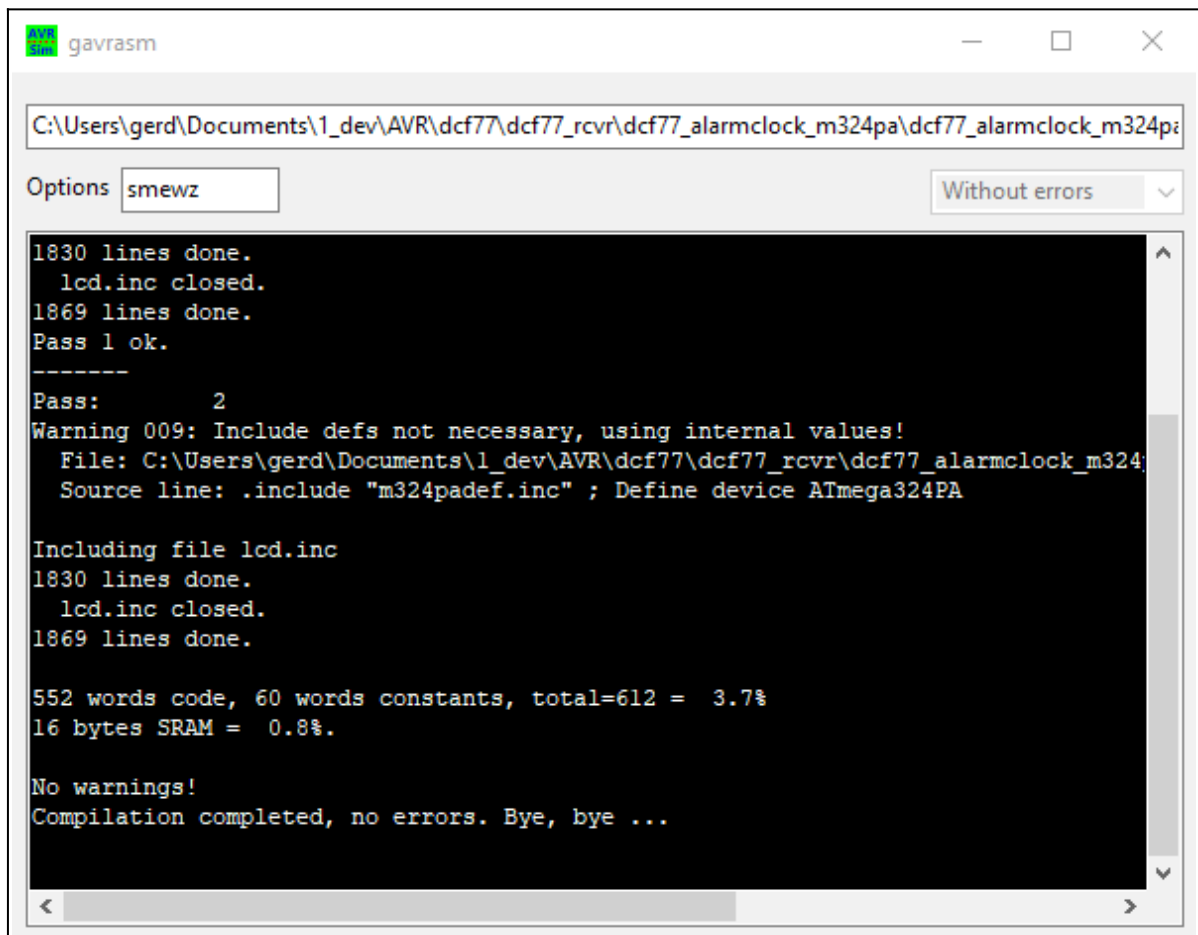
If a new include file shall be added place the cursor to the beginning of the line where the include directive shall be added with the .include directive.

Then right-click with the mouse onto the editor field. Select "Add include" and, in the opening dialog, a file name. The new file is generated in the folder where the project resides. All included files that have an include directive entry (.include "[inc name]") are recognized during the load process of the main file. Those are added automatically to the tab entries and are directly accessible. Excluded are *.def.inc files which are used to identify the selected AVR type but have no tab entry. Nested includes are possible to any depth.

5 Assembling

5.1 Starting the assembly process

By pressing the menu item "Assemble" the internal gavasm assembles the source file (and all associated include files).

A screenshot of the 'gavasm' application window. The title bar shows 'gavasm' with standard window controls. The address bar displays the file path: 'C:\Users\gerd\Documents\1_dev\AVR\dcf77\dcf77_rcvr\dcf77_alarmclock_m324pa\dcf77_alarmclock_m324p'. Below the address bar, there is an 'Options' field containing 'smewz' and a dropdown menu set to 'Without errors'. The main text area shows the following assembly output:

```
1830 lines done.
  lcd.inc closed.
1869 lines done.
Pass 1 ok.
-----
Pass:      2
Warning 009: Include defs not necessary, using internal values!
  File: C:\Users\gerd\Documents\1_dev\AVR\dcf77\dcf77_rcvr\dcf77_alarmclock_m324
  Source line: .include "m324padev.inc" ; Define device ATmega324PA

Including file lcd.inc
1830 lines done.
  lcd.inc closed.
1869 lines done.

552 words code, 60 words constants, total=612 =  3.7%
16 bytes SRAM =  0.8%.

No warnings!
Compilation completed, no errors. Bye, bye ...
```

After gavasm ended, a message signals errors or success and the generated listing plus eventually the generated error file ".err" is added to the editor tabs.

Assembling with gavasm sets the options -smewz, which means

- **s** enables the symbol list at the end of the listing, where you can inspect the AVR type (T), constants (C), registers (R) and label addresses (L) of the assembled code,
- **m** enables that code generated by macros is listed line by line (this enables the simulator to exactly identify the code line executed),

- **e** switches extended error messages on, so the .err output, displayed in another editor tab, has all information that are necessary to jump to the line in the source code that produced the error,
- **w** enables wrap-around, so in AVR types with more than 4 kWord flash a jump back from or near the flash end to the beginning of the flash is possible with the instruction RJMP,
- **z** switches the gavasm-specific date information constants **Now_I** (days since 1.1.1990), **Now_Y** (current year, 0..99), **Now_M** (current month) and **Now_D** (current day) on, so these can be used to insert date information in the produced flash code.

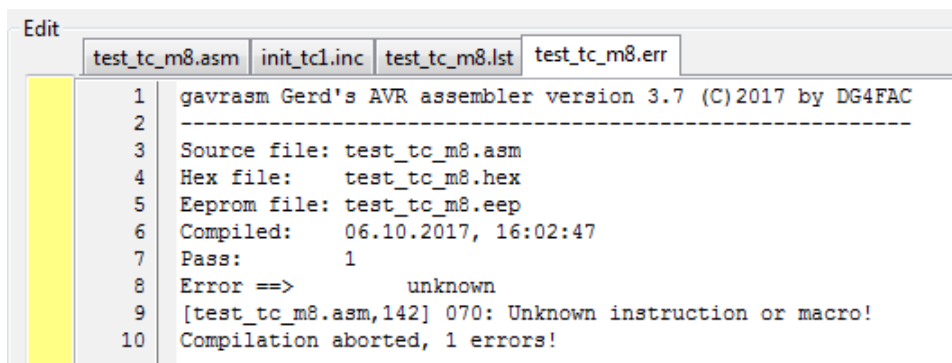
If you want to avoid the Warning 009: use the gavasm-specific .device "[AVR device" directive and remove the .nolist, the -.include "[device]def.inc" and the .list directives. Note that this solution is incompatible with other assemblers. If you want to use the simulator's assembler gavasm as well as other assemblers with the same source code file, then ask if a symbol with the device name is defined (which gavasm generates), then gavasm is assembling. Use the following formulation to get it compatible in any case:

```
.ifndef ATmega8a ; symbol is defined if gavasm is assembling
.device "ATmega8a" ; gavasm is assembling
.else ; a different assembler is at work
.nolist
.include "m8adef.inc"
.list
.endif
```

Note that this solution only works with more modern assemblers that are capable to process .IF directives.

If assembling found no errors, the list file is displayed in the editor window. If you place the cursor into a line of the list file, the context menu offers to change immediately to the line in the source file, where the list file entry results from.

5.2 Errors during assembling



```
Edit
test_tc_m8.asm  init_tc1.inc  test_tc_m8.lst  test_tc_m8.err

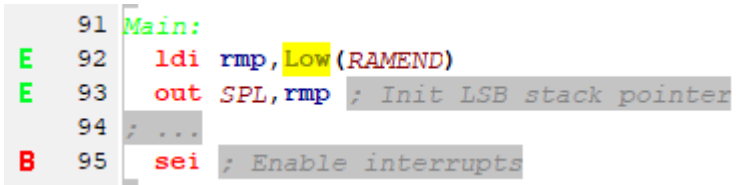
1  gavrasm Gerd's AVR assembler version 3.7 (C)2017 by DG4FAC
2  -----
3  Source file: test_tc_m8.asm
4  Hex file:    test_tc_m8.hex
5  Eeprom file: test_tc_m8.eep
6  Compiled:   06.10.2017, 16:02:47
7  Pass:       1
8  Error ==>   unknown
9  [test_tc_m8.asm,142] 070: Unknown instruction or macro!
10 Compilation aborted, 1 errors!
```

If during assembly the assembler found source code errors, the error file .err is among the tab entries. By right-clicking onto

the line with "[filename,line]" the editor places the cursor to the error location.

5.3 Setting/removing breakpoints

Lines that have an "E" in the side window are identified as executable instructions. By setting the cursor into those lines of the editor (not into the side window) and right-clicking with the mouse a con-



```
91 Main:
E 92 ldi rmp, Low(RAMEND)
E 93 out SPL, rmp ; Init LSB stack pointer
94 ; ...
B 95 sei ; Enable interrupts
--
```

text menu offers to toggle breakpoints. Those breakpoints are stored in the project file and are reloaded when an existing project file is opened. Note that those breakpoints are only displayed after assembling the project.

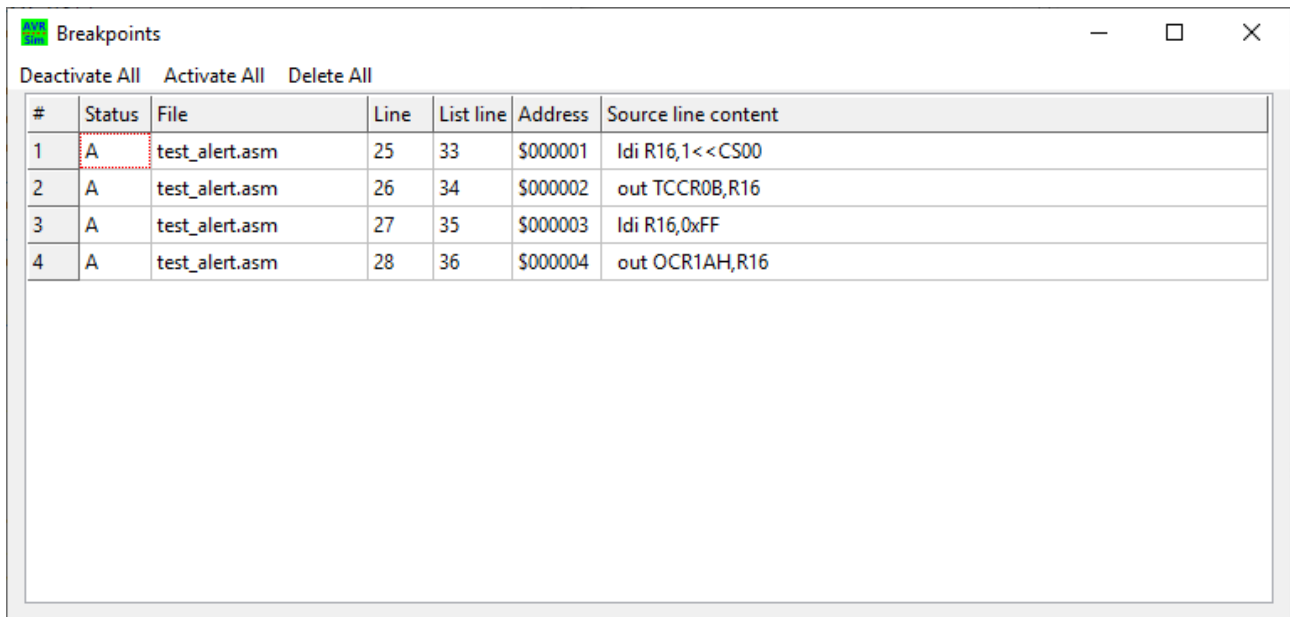
Breakpoints are copied to the list file, that is also available as tab file following assembling. In the list file breakpoints can be enabled and disabled as well, those changes are copied to the original .asm and .inc files where they result from.

Breakpoints can be changed "on the run" during an active simulation process going on. The changes come immediately into effect.

The list of breakpoints can be viewed: click "View" and "Breakpoints". Displayed are:

- the status of the breakpoint: "A" for activated, "D" for deactivated,
- the source file name, where the breakpoint is located,
- the line number in that source file,

- the line number in the list file,
- the associated hexadecimal address, and
- the content of the line where the breakpoint is located.



Breakpoints

Deactivate All Activate All Delete All

#	Status	File	Line	List line	Address	Source line content
1	A	test_alert.asm	25	33	\$000001	Idi R16,1<<CS00
2	A	test_alert.asm	26	34	\$000002	out TCCR0B,R16
3	A	test_alert.asm	27	35	\$000003	Idi R16,0xFF
4	A	test_alert.asm	28	36	\$000004	out OCR1AH,R16

By clicking into the status field of a breakpoint this can be deactivated and activated again. Note that only activated breakpoints are stored in the project file and are reloaded at project start.

Clicking into the number of the breakpoint (below the column header "#") deletes this breakpoint. Clicking into a breakpoint's entry in the "Line" column opens a dialog window that enables to move that breakpoint to another line in the source code. Note that only lines that produce executable code (in the source code file marked with an "E", in the list file with a hexadecimal formatted instruction code). Erroneous setting to a different line throws an error message and deactivates this breakpoint. A similar move of the breakpoint can be started by clicking into the cell with the header "List line".

Clicking onto one of the headers (except for the line content) sorts the breakpoints by increasing content in that column.

The menu entries allow to activate ("Activate All"), deactivate ("Deactivate All") or delete ("Delete All") all breakpoints. A maximum of 24 different breakpoints

can be handled.

All changes made come immediately in effect, even during a running simulation.

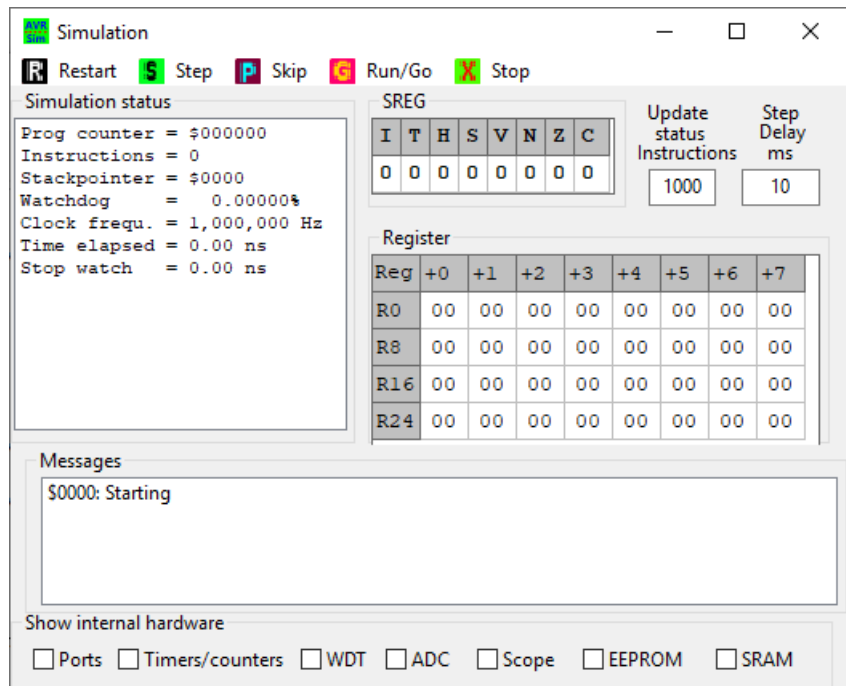
When changes to the sources text are made, avr_sim tries to reflect those changes. Following successful re-assembling all breakpoints are checked. If breakpoints had been moving, those are re-located first downwards, then upwards in the source code. If relocation fails, the related breakpoint(s) are deactivated and can be manually corrected in the "View/Breakpoints" window.

A special situation occurs if breakpoints in lines within a macro are to be set. While setting the breakpoint within the macro definition is not possible, because the macro definition does not produce any executable code, setting it in list-file leads to an error when trying to locate the line within the source code (the list displays the macro's name). Please note that the breakpoint's address might be wrong.

6 Simulating

6.1 Starting simulation

By clicking on the menu item "Simulate" the simulation window opens. It displays all basic properties of the program. All display windows (Ports, Timers/Counters, ADC, etc.) that were open in the previous session re-open automatically.



The menu items in this window provide the following:

1. Restart or Ctrl-R: Starts the program with a reset (program counter = 0), clears all hardware.
2. Step or Ctrl-S: Perform a single step with the next executable instruction (the editor windows shows a right arrow on this instruction, either in blue or in red (if a breakpoint is located there)).
3. Skip or Ctrl-P: Jumps over the next executable instruction. Please use this carefully because jumping over RET or RETI instructions leaves the stack in disorder.
4. Run/Go or Ctrl-G: Runs indefinitely until
 - a breakpoint is encountered, or
 - the menu item "top" is clicked, or
 - the PC counter runs onto an illegal address location.
5. Stop or Ctrl-X: Stops the running execution.

All changes made on SREG, registers, SRAM and other hardware are displayed

during execution. Performing a single step clears all those marked changes.

The content of the registers can be manipulated by entering hexadecimal numbers into the register fields, followed by the return key. Only valid hex numbers are accepted. Note that re-starting the simulation clears all register content.

In interrupt-controlled operating modes the share of SLEEP instructions performed is displayed if at least one instruction has been performed.

Elapsed time, the stop watch and the SLEEP share are cleared if a right-click is made on those entries. Please note that clearing elapsed time also clears the scope event storage.

The watchdogs time is displayed as percentage of the total time.

The edit field below "Update status" allows to input the number of instructions that trigger an update to the display if a Run is active. Smaller numbers increase display update speed, but also increase the time that is needed for simulation. The entry can be changed on the fly and comes immediately into effect.

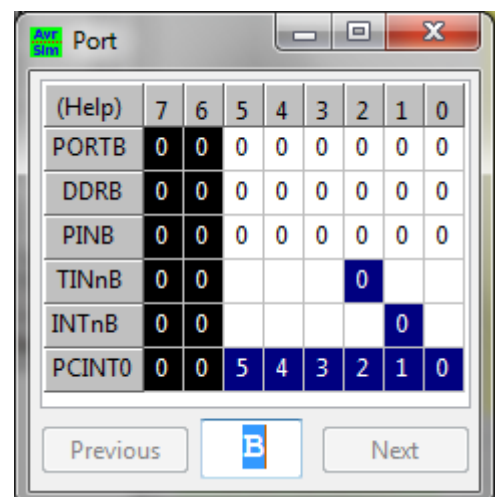
The edit field "Step delay" allows to add extra time to each simulation step when running by setting it to numbers above 0 or 1 ms. Changes come into effect immediately.

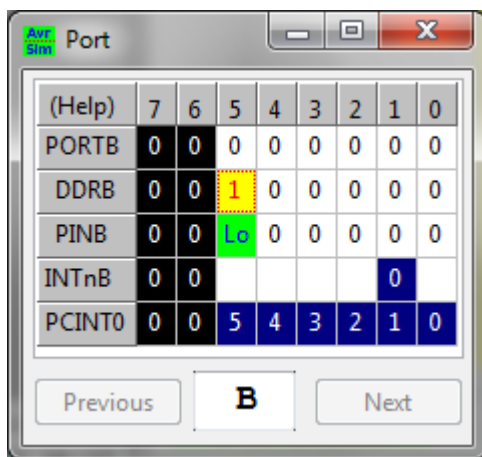
The message window provides information on unimplemented or illegal instructions etc.. Identical messages at the same program counter address are displayed only once.

6.2 Viewing I/O ports

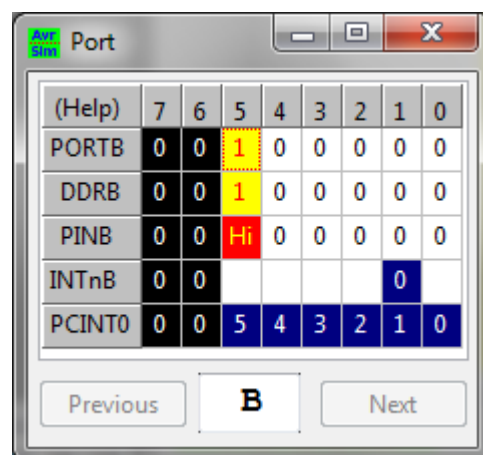
By activating the "Port" checkbox the port view opens.

Here the port B of an ATtiny13 is shown. The white-on-black bits are not available in the device.

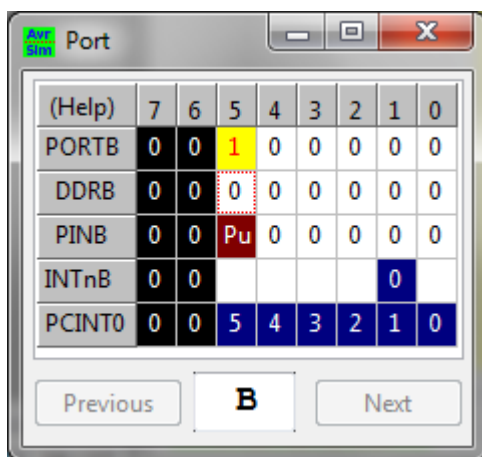




By manually clicking on a bit in the lines "PORTB", "DDRB" clears or sets the respective bit immediately and



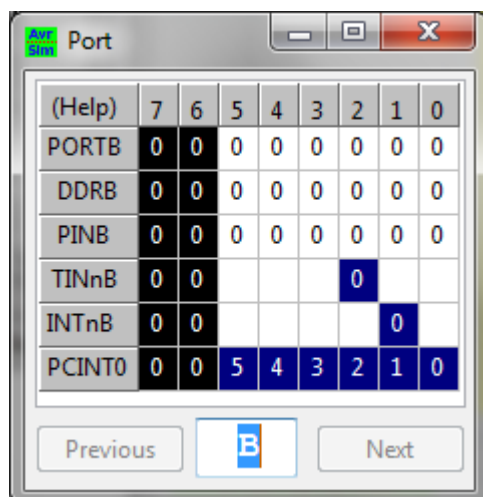
marks it as "Changed". If the DDR bit in the port is set, the respective PIN bit in the read port register PIN follows the PORT bit.

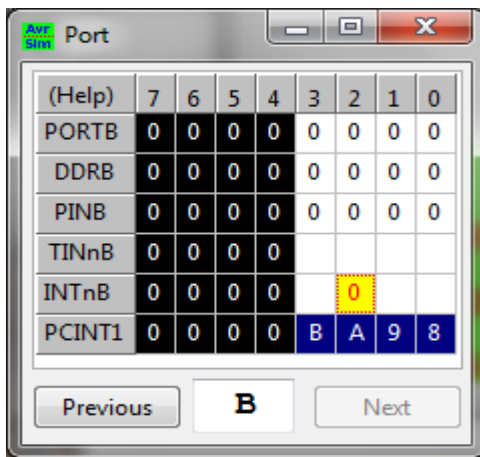


If the PORTB bit is set while the respective DDRB bit is cleared the PINB bit is marked as pulled-up.

If neither the PORT nor the DDR bit is set, the PIN bit can be pressed to toggle the PORT bit, in devices that support toggling via the PIN. OUT or SBI instructions on the PIN port have the same effect in those devices.

The line TINn shows the bits that influence the counter n. If the TINn bit is selected as source in the respective counter one can click on it to advance the counter. If TINn is not selected as input in counter n, the bit appears white-on-blue. If TINn is active on falling edges it appears yellow-on-violet. If active on rising edges it is red-on-lightgray.





The line "INTnB" indicates the external interrupt bits INT0, INT1, etc., if available in the device. The white-on-blue formatting indicates that the INTn bit is available in the device.

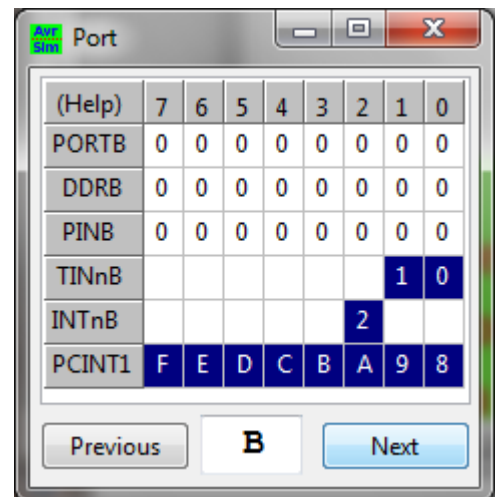
By clicking on those bits immediately the respective interrupt is requested, if the I-bit in the status register is set. The red-on-yellow formatting shows that an INT0 interrupt is currently

pending. The background color turns from yellow to red, when the interrupt is executed (if the I bit in the status register is set and the next step is executed). On execution of the RETI instruction the interrupt condition terminates.

INTn interrupts can be selected by software as sensitive for a low bit state on the input pin (ISC=00), for toggling (ISC=01), for falling edges (ISC=10) or for rising edges (ISC=11). Different colors show the respective state of the INTn interrupt.

If software activates the external INTs their colors changes again, e.g. from dark green to light green in toggle mode. The font color is red. Note

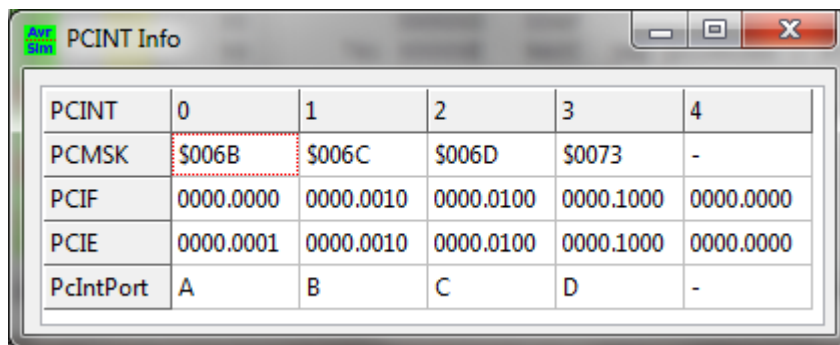
that between the activation of an INTn (or PCINTn) condition and activation of its execution are four clock cycles delay over which the signal has to remain stable. If the condition is terminated earlier, an error message ("Spurious INTn condition") is displayed in the message window and the INTn/PCINTn is not executed.



The color explanation occurs by pressing the (Help) entry in column 1 line 1 of the grid.

The respective PCINTs appear on the line "PCINTn". Those are always in the toggle mode (dark green, or light green if activated by their respective mask and interrupt enable bits). The PCINT bits 10 to 15 are displayed as "A" to "F", bits 16 to 23 as "G" to "N", 24 to 31 as "O" to "V" and 32 to 39 as "a" to "h".

By clicking the "Prev" and "Next" buttons the other ports can be displayed, if available in the selected device type.



PCINT	0	1	2	3	4
PCMSK	\$006B	\$006C	\$006D	\$0073	-
PCIF	0000.0000	0000.0010	0000.0100	0000.1000	0000.0000
PCIE	0000.0001	0000.0010	0000.0100	0000.1000	0000.0000
PcIntPort	A	B	C	D	-

0 1		Portbits low/high
0 1		Changed bits low/high
0 1		Input pins
Lo	Hi	Input pin forced low or high by DDR bit set
Pu		Input pin with pullup enabled (high)
External interrupt pins		
1 0		INT1 and INTO disabled, ISC00 and ISC01 = 0
0 0		INT0 in toggle mode, disabled and enabled
0 0		INT0 on falling edges, disabled and enabled
0 0		INT0 on rising edges, disabled and enabled
1 0		INT1/PCINT1 pending INT0/PCINT0 executing

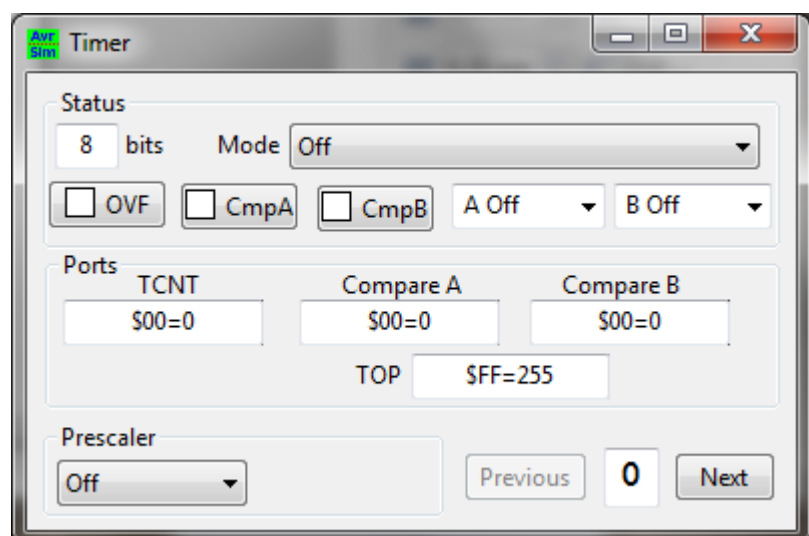
By clicking the row header PCINTn more information on the PCINTs

is provided.

6.3 Viewing and manipulating timers

6.3.1 Viewing timers

By clicking the checkbox "Timers/Counters" on the simulation window the timers and counters are displayed, as far as those are available in the device type. In devices with more than one timer/counter a click on "Previous" or "Next" displays the other timers/counters.



Timer

Status: 8 bits Mode: Off

☐ OVF ☐ CmpA ☐ CmpB A Off B Off

Ports: TCNT \$00=0 Compare A \$00=0 Compare B \$00=0

TOP \$FF=255

Prescaler: Off

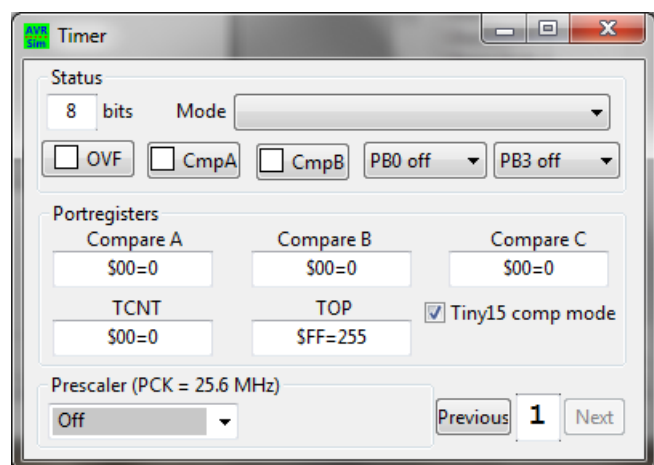
Previous 0 Next

The following information is available:

1. the timer number (below right),
2. the timer resolution 8 or 16 bit (top left),
3. the timer operation mode (top),
4. the three possible interrupt modes (if available), a green square indicates an active int enable flag, a yellow square a pending interrupt request and a red square a currently executing interrupt,
5. the status of the compare match output modes of ports A and B (off, toggle, set, clear, PWM set and PWM clear mode),
6. the timer's TCNT, OCR-A and OCR-B portregister contents, please notify that
 - in modes that change OCR-A or OCR-B on TOP or at BOTTOM the OCR-A and OCR-B value's background is colored in yellow as long as the effective compare value differs from the port's content,
 - these portregisters in 16 bit timers change only if the low byte is written, while the high byte is stored intermediately,
7. the prescaler setting and, if enabled, the current count of the prescaler.

Normally the displayed timer is determined by the simulator: whenever a timer is changed by code execution this timer is displayed. If you enter a timer number in the input field, this timer will be fixed: the selected timer is always displayed and only changes to this timer are visible. To disable fixing just remove the timer number and the simulator takes over control again.

In case of the timer/counter 1 of the ATtiny25/45/85 additionally the Compare C port occurs. If in High-Speed-Mode (PLL switched on) the check



field for Tiny15 compatibility mode occurs which alters the PLL frequency. The

prescaler field shows the frequency of the source.

6.3.2 Changing timer values

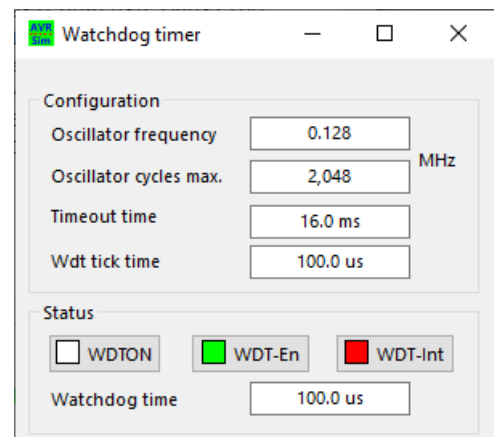
In the Timer/Counter view the following items can be changed:

1. The values of TCCNT, Compare A, B, C and ICR can be changed if available in the device. Just click into the respective edit field, enter either hexadecimal (preceded by a \$, A to F in small or capital letters) or decimal numbers and hit the return key. In case of an error (unreadable number or number larger than allowed) the edit field turns red. The values entered change the respective port register values immediately, but in case of the compare values A, B and ICR those might get effective only later on if the counter reaches TOP, MAX or BOTTOM, depending from the selected operation mode of the timer/counter.
2. The value of the prescaler. Just select the desired entry from the drop-down field. Note that manually entered changes in the text field are ignored.

6.4 The watchdog timer

From Version 2.0 on the watchdog timer, that is available in all AVR types, is available for use, simulation and display. Please note that not all special features of the more advanced AVR types are implemented, only the basic functionality works correct.

To view the watchdog timer check the WDT checkbox in the simulation window. The "Configuration" part of the watchdog window displays



- if the watchdog is unavailable,
- the frequency of the watchdog-timer oscillator (which is depending from hard- and software),

- the software-configured number of oscillator cycles that triggers a watchdog event (depending from the WDPn bits),
- the resulting time that these cycles take, plus
- the time tick with which the watchdog timer is advanced in each instruction cycle (which depends from the current clock frequency).

The "Status" part of the window displays:

- the state of the WDTON fuse: this can be changed by clicking on the color button and selecting either white (WDTON fuse switched off) or a different color (red, WDTON fuse activated). The WDTON status, if activated, is saved in the project file and is restored on loading the project.
- if the watchdog is running, which is the case if either WDTON is activated or if the WDEN bit in the watchdog timer control- and status-port WDCSR is set (please use the WDCE bit to enable write operations within the next four instructions),
- if the AVR type has a WDT interrupt bit WDIE and WDTON is inactive, the current interrupt status is displayed with an additional color button (white: inactive, green: interrupt enabled, yellow: interrupt condition reached, red: interrupt currently executed).
- the editing field displays the elapsed time of the watchdog timer.

The content of the configuration part is read-only and cannot be changed manually. In the status part, you can click on the editing field, where the elapsed time is displayed, which clears the elapsed time (just like as if the instruction "Watchdog Reset" WDR is executed). The WDTON color (either white or red) can be changed, the changes come immediately into effect. Even though the colors of two other color buttons can be changed, those changes will not come into effect (no changes to WDCSR will happen) and are overwritten whenever the next configuration event occurs.

6.5 Viewing and manipulating ADC content

6.5.1 ADC channels

By checking the ADC box in the simulation window the ADC display opens.

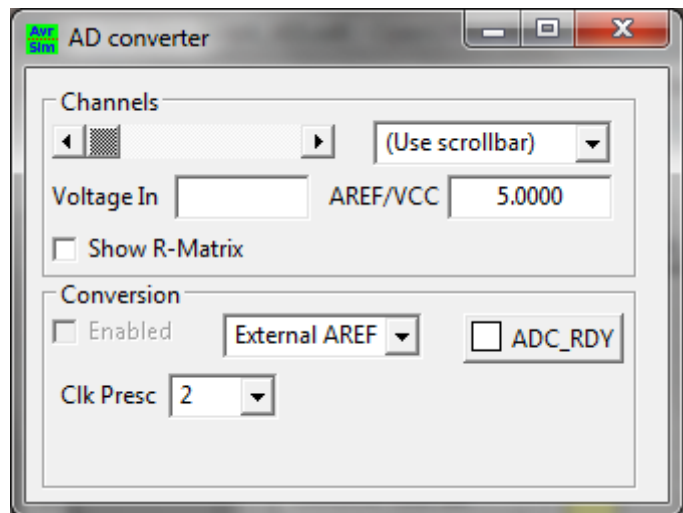
In the channels section of the window the available ADC channels of the device can be selected as ADC source. Note that no differential channels and gains are implemented.

Selection of the channel changes the current ADMUX values. Those changes come only into effect when a new conversion starts.

If the ADSC bit in the ADC control port is set, the conversion is started from another source (e. g. from a timer event). In that case a drop-down-field right to the clock prescaler value shows up and displays the source.

The input voltage on each channel can be changed by inputting voltages in the "Voltage In" edit field. Finalize inputs with the RETURN key. If the AREF input pin or the operating voltage of the device is selected as ADC reference voltage, the respective voltage can be changed in the AREF/VCC edit field. All voltages can be saved and are stored in the project file and are reloaded when the project file is reloaded.

The results of the conversion are displayed in the read-only editor fields ADCH and ADCL.

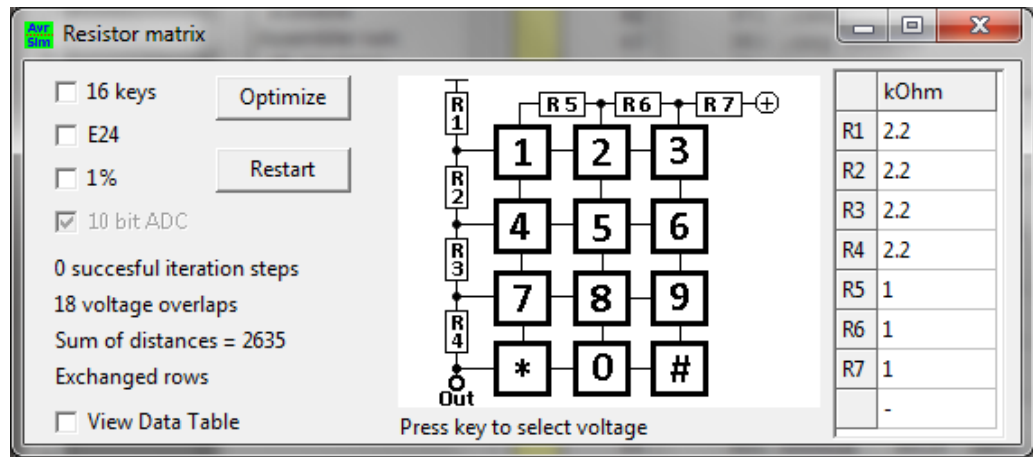


6.5.2 Resistor matrix as voltage input source

The R-Matrix field allows to open a window with a resistor matrix that generates input voltages for the ADC.

The R-Matrix window starts with a standard set of resistors. By pressing the "Optimize" button the resistor network is iterated in 100 steps. If voltage areas overlap

- repeat pressing the "Optimize" button until the

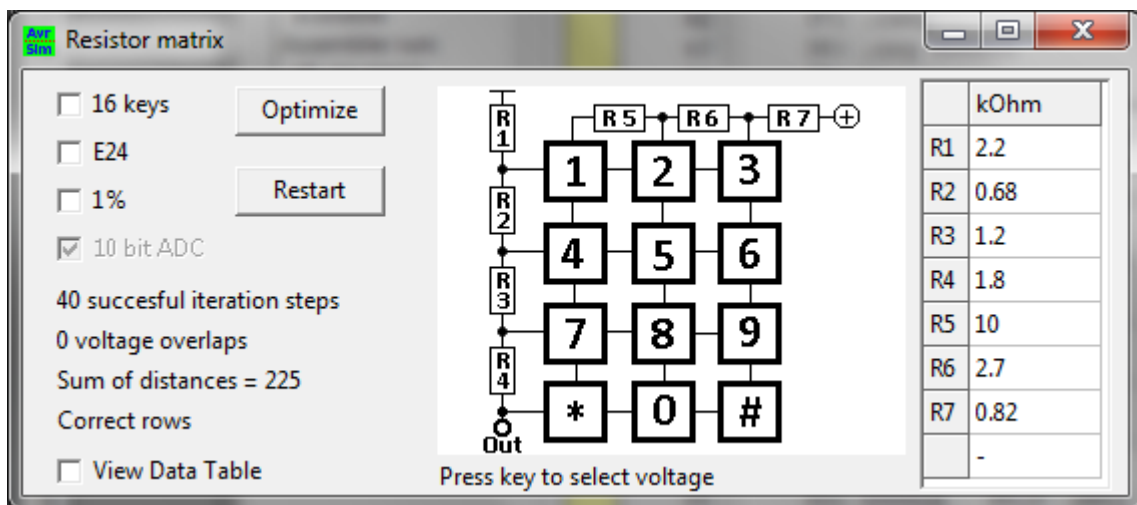


number of successful iteration steps is not increasing any more,

- check the 1% precision box (standard is 5% precision of the resistors),
- restart the random iteration with the "Restart" button.

The iteration process is not always successful. Sometimes it helps to start the optimization with 1% resistors and, if successful, select 5% again. The following shows such a combination.

Further opti-



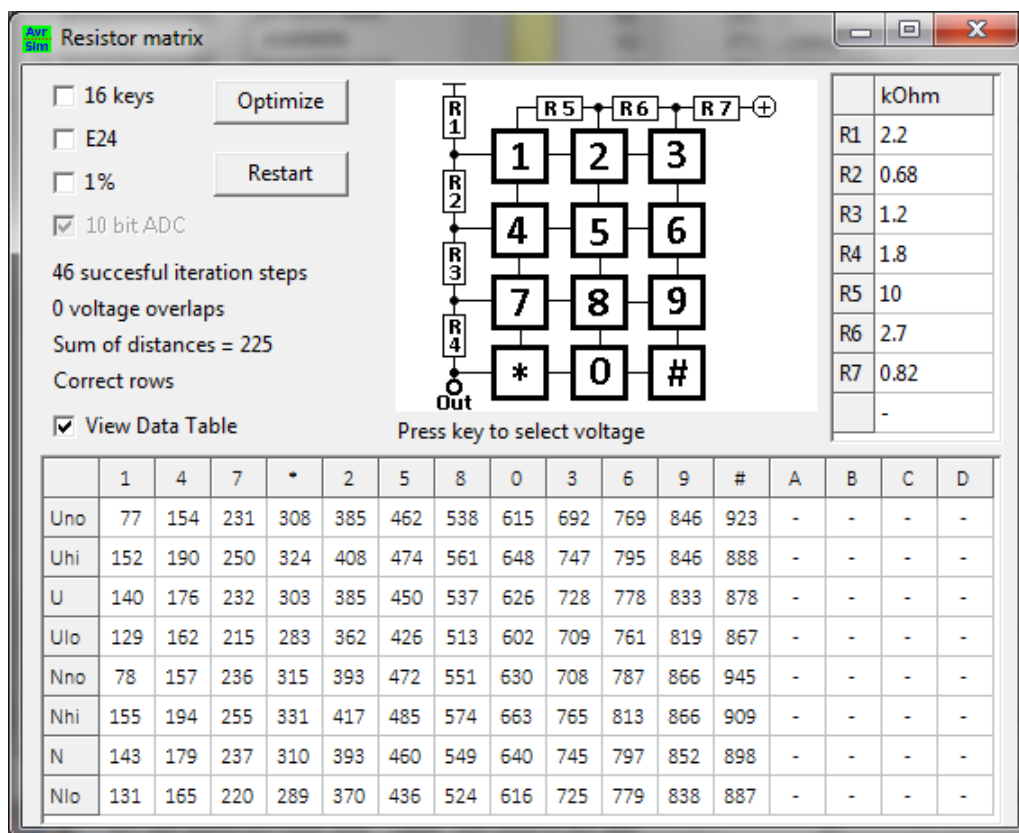
mization can be done by allowing resistor values of the E24 row (default is E12).

If "Correct rows" is displayed the voltages increase by increasing rows and columns. In that case the key row is "1-4-7-*-2-5-8-0-3-6-9-#".

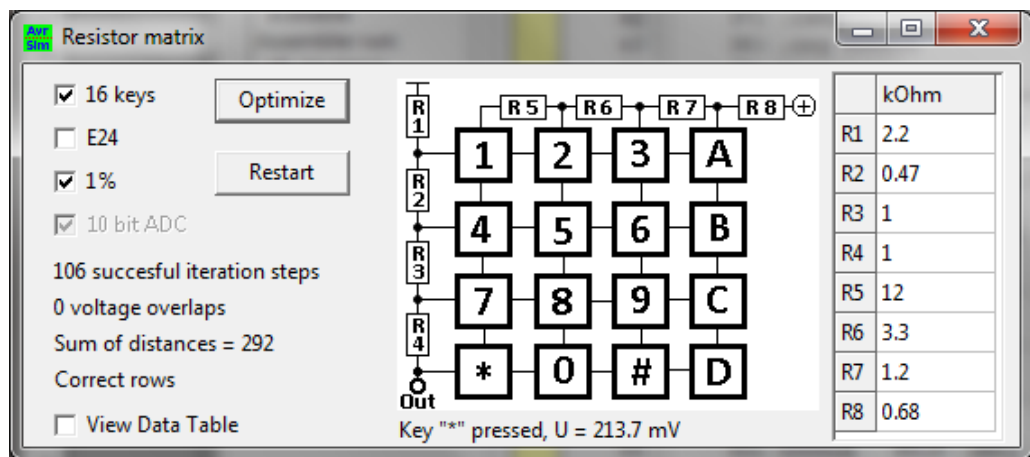
By checking the "View Data Table" box the voltages in mV (at an operating voltage of 1 V) and the resulting ADC values (8-bit resolution if the ADLAR bit in ADMUX is set, 10 bit if not) can be displayed. In this table the following lines

mean:

- Uno, Nno: normal voltage and ADC value of the key in case of equal distribution,
- Uhi, Nhi: upper voltage and ADC value if the resistor tolerance generates the highest result,
- Ulo, Nlo: lower voltage and ADC value if the resistor tolerance results in the smallest values,
- U, N: the nominal voltages and ADC values if the resistors have their nominal value.



If the 16-key box is checked the similar resistor network with 16 input keys is selected. The scheme uses one additional resistor.



With 16 keys in nearly all cases 1% resistor tolerance is necessary. The number of successful iterations increases in some cases without changing resistor values.

The correct row with increasing voltages is "1-4-7-*-2-5-8-0-3-6-9-#-A-B-C-D".

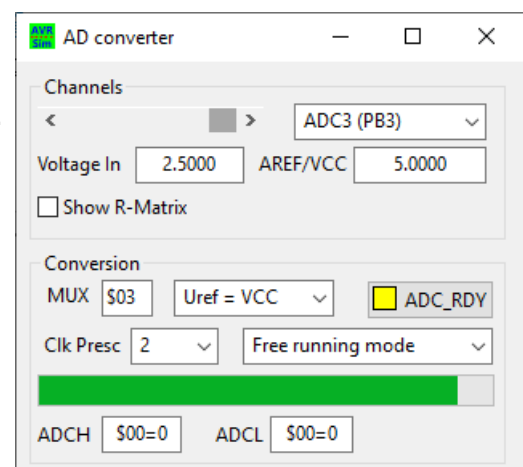
By clicking on the keys in the scheme the resulting voltage of that key is, multiplied by the VCC voltage, transferred to the ADC input voltage edit field.

6.5.3 Active AD conversion

In the "conversion" field of the ADC window the reference source, the status of the ADC interrupt, the clock prescaler value and, if the bit ADATE in ADCSRA is set, the source that starts conversions is displayed.

During ongoing conversions the progress bar displays the number of already converted bits. All entries except the port drop-down and the enable checkbox can be changed, the changes made here change the respective port register bits in ADCSRA and ADCSRB.

If interrupts are enabled, the ADC_RDY field is green, if an interrupt is pending it is yellow and if the interrupt is currently pro-



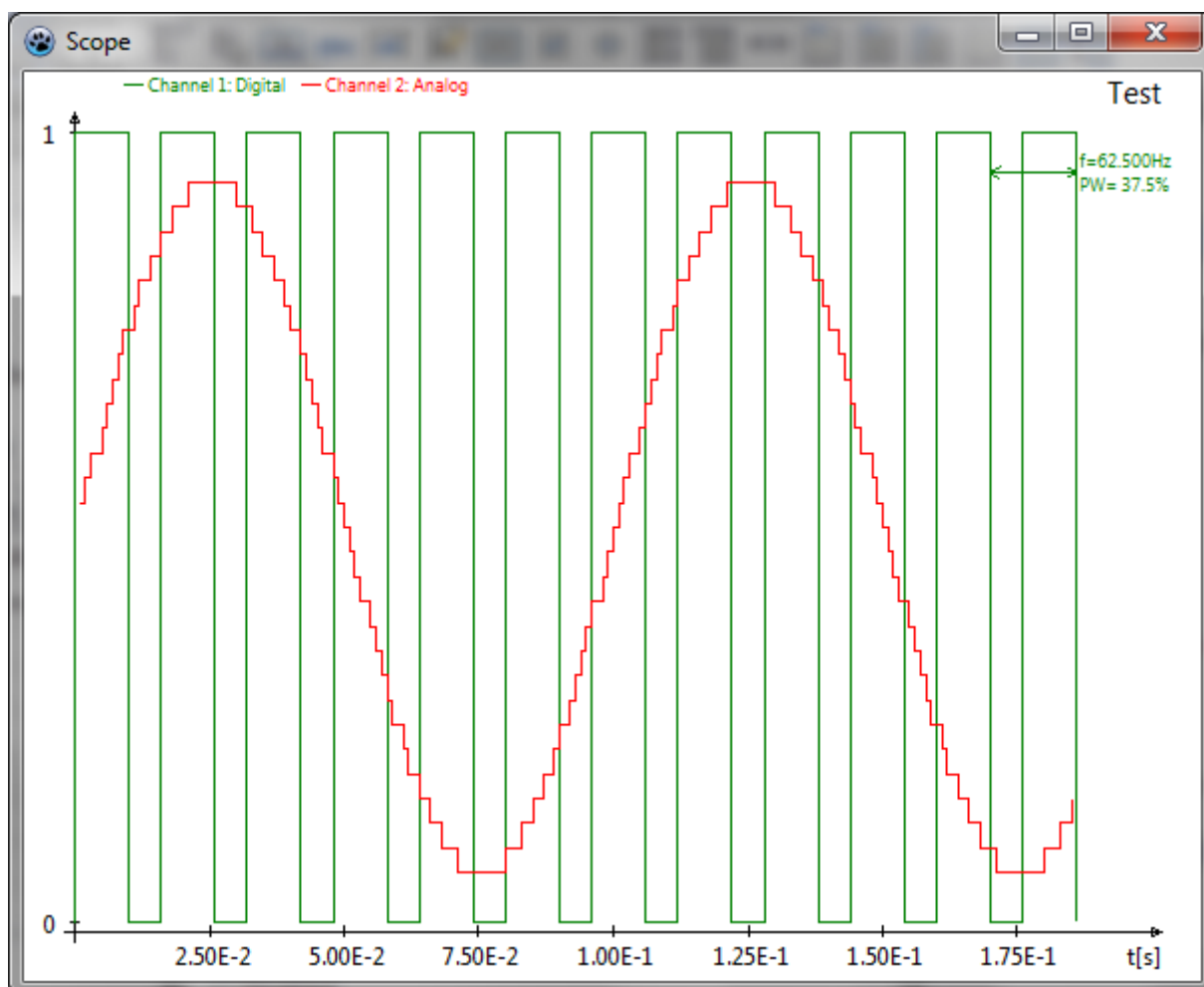
cessed it is red. The fields ADCH and ADCL show the last ADC conversion result.

The example here shows the AD conversion (ADLAR set, MSB) results with input voltages of 0, 1, 2, 3 and 4 V at an operating voltage of 5 V, stored in SRAM.

SRAM							
	+00	+01	+02	+03	+04	+05	+06
\$0100	00	33	66	99	CC	FF	FF
\$0110	FF	FF	FF	FF	FF	FF	FF

6.6 Scope display

For all digital I/O pins, for timer-controlled OC pins and for digital-to-analog converters based on R/2R networks a scope display can be added. It displays any changes in the selected beam channel(s) immediately. The picture shows an example with one digital and one analog signal enabled.



Up to four channels can be configured to be displayed. When "Scope" is enabled in the simulator window, the scope configuration opens.

If a beam or channel is enabled, select either a "Digital" or an "Analog" mode.

For digital signals select either an OC pin (if available in that device) or an I/O pin to be followed. For I/O pins, select the port first, then one of the available portpins for observation.

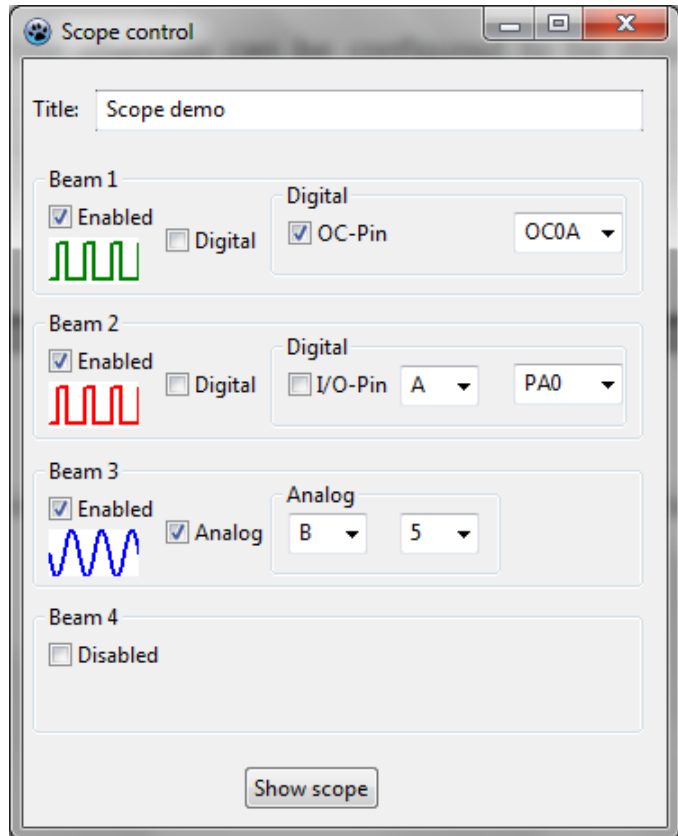
For analog signals select the port to which the R/2R network is connected. From the available bits in this port select the number of bits of the DAC (ranges from 2 to 8 if enough port bits are available in the selected port of the device). If the device has an internal DAC this can be displayed as Beam 5.

Select the colors of the scope display for the channels by clicking on the small windows below the "Enabled" checkbox.

Input of the title is also required. If any one channel is selected and the title is provided press "Show scope" to open the scope display.

If you want to change the scope's properties during execution, stop the running execution and disable and enable "Scope" again. Changes come into effect immediately, using the stored event data. If new channels have been added, either click "Restart" (the program restarts from scratch) or clear the "Time elapsed" in the status window (the event storage is cleared and starts again without restarting program execution).

The displayed times are derived from "Time elapsed" in the simulator window.



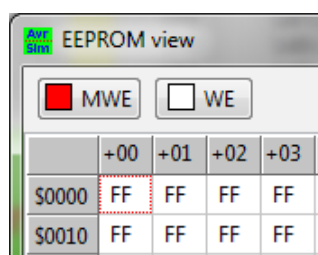
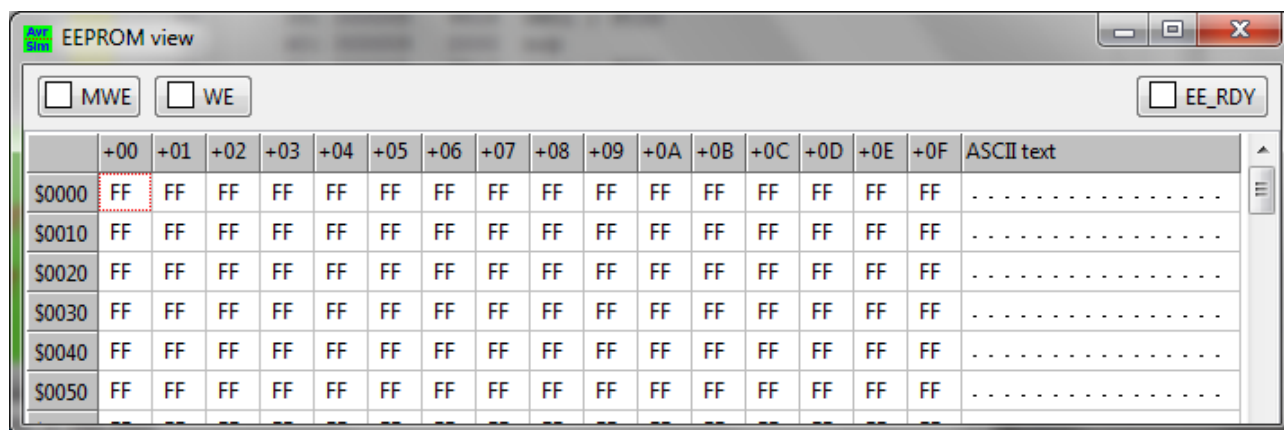
If this is cleared, or if simulation execution is restarted, the event storage is cleared. Up to 1.000 events can be handled, beyond that an error message is displayed.

Scope configuration is stored in the project's configuration file and is restored if the project is re-loaded. The scope window then pops up automatically. If you want to avoid that, open the scope control window, disable all beams and press the "Discard" button. This clears the scope entries in the project file when closing the main window.

The displayed scope picture can be saved in PNG or BMP format by clicking on it. Make sure that no Run is active.

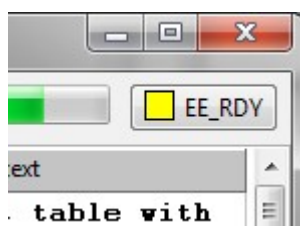
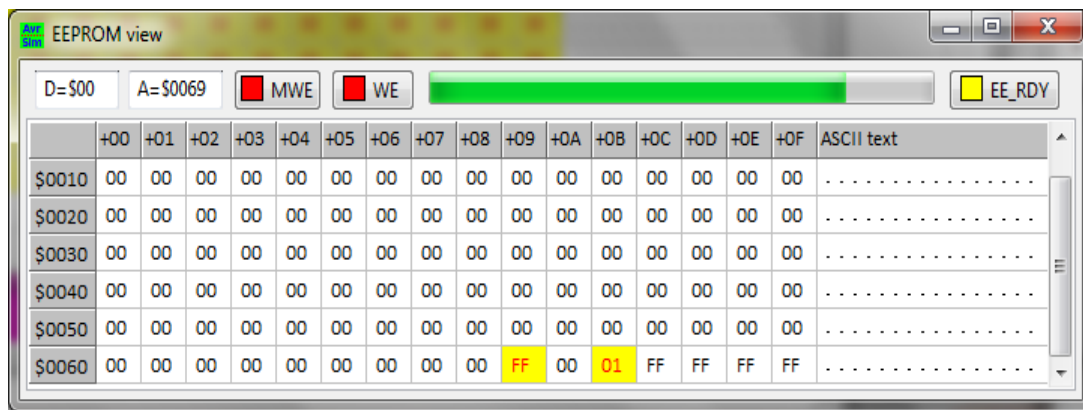
6.7 Viewing EEPROM content

With the checkbox "EEPROM" enabled the content of the EEPROM is displayed.



If the Master-Write-Enable bit (in some devices called Master-Programming-Enable bit) is set, this is displayed like this. This bit is cleared after four clock cycles.

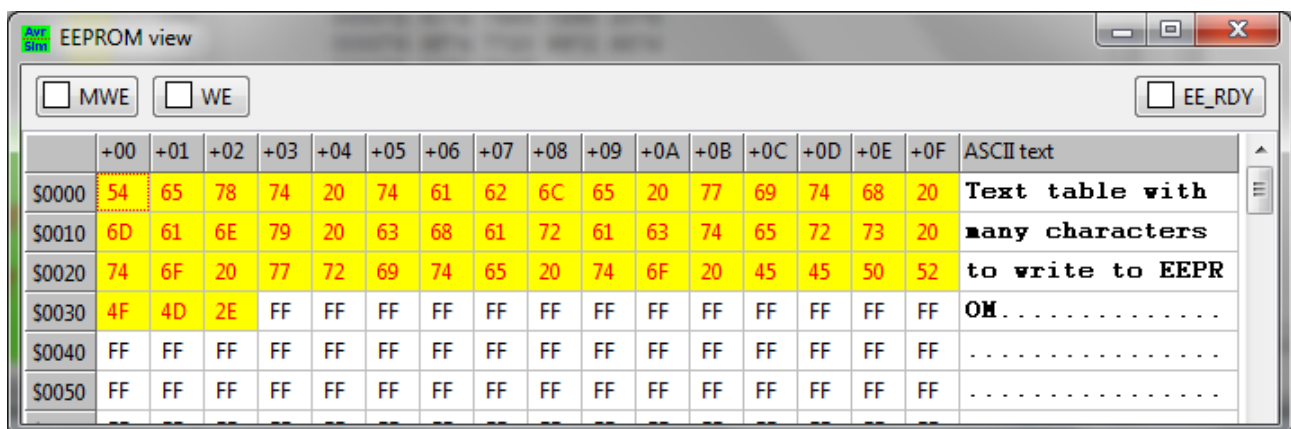
If within these clock cycles the Write-Enable (Programming enable) bit is set, the WE bit is displayed and a progress bar shows up. This bar displays the status of the write progress. The two read-only edit fields show the content of EEDR and EEAR.



The yellow sign shows if the EE_READY interrupt is enabled. A red one shows that the execution of the interrupt is under progress. Please note that the interrupt is repeated as long as the interrupt enable



bit in EECR is set and as no EEPROM write is going on.



Red on yellow characters are changed locations.

EEPROM content is read-only and cannot be changed manually.

6.8 Viewing SRAM content

With the checkbox "SRAM" enabled the content of the SRAM is displayed. Changed values appear in red on a yellow background.

	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	ASCII text
\$0060	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F+...!
\$0070	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
\$0080	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	!"#\$%&'()*+,-./
\$0090	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	0123456789:;<=>?
\$00A0	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	@ABCDEFGHIJKLMNO
\$00B0	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	PQRSTUVWXYZ[\]^_
\$00C0	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	`abcdefghijklmnopqrstuvwxyz
\$00D0	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	pqrstuvwxyz{ }~!
\$00E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

SRAM content is read-only and cannot be manually changed.

6.9 Alerts

Since avr_sim's version 2.3 additional alerts can be set. These alerts can be configured to

1. count the number of the events,
2. to play a tone on the speaker,
3. interrupt the running simulation (break).

Scan and Alert					
Alert 1					
<input checked="" type="checkbox"/> Active	Scan	for what	for	Action	Counter
	Register	R16	Write	Break+Sound+Count	0
					Clear

Scanned for those events can be

1. single registers (R0 to R31),
2. succeeding register pairs (ZH:ZL down to R1:R0),
3. single portregisters (P=0 to P=63),
4. succeeding portregister pairs (here e. g. TCNT1H:TCNT1L),

Scan and Alert

Alert 1

☒ Active

Scan: Portregister-pair

for what: TCNT1H:TCN

what for: Value

Value: 12345

Action: Count+Break+Sound

Counter: 1

Clear

5. single SRAM locations, (e. g. 0x0060), as well as

6. succeeding SRAM pairs (e. g. 0x0061:0060).

Depending from what has to be scanned,

- all write accesses,
- all read accesses,
- all write- or read-accesses,
- writing certain values, or

Scan and Alert

Alert 1

☒ Active

Scan: Register

for what: R16

what for: Value

Value: \$01

Action: Count+Break+Sound

Counter: 3

Clear

- all value changes

lead to such an event. As reaction to such events the following is possible:

1. simple counting,
2. counting and beeping,
3. counting and stopping the simulation (on stop the cursor changes from blue to green),

Scan and Alert

Alert 1

☒ Active

Scan: Portregister-pair

for what: TCNT1H:TCNT1L

what for: Value

Value: 100

Action: Count+Break

Counter: 6

Clear

4. counting, beeping and stopping.

Scan and Alert

Alert 1

☒ Active

Scan: Register-pair

for what: R31:R30=Z

what for: Change

Action: Count+Break+Sound

Counter: 0

Clear

If this function in this version is successful, it is planned to add an additional alert in later versions.

7 Not yet implemented, but under consideration

The following features are planned but are not yet implemented in the current version, but are planned for future versions:

1. Manual changes to timer/counter modes are not yet implemented.
2. Devices with more than 128 kB flash memory are not yet implemented.
3. The AVR8 devices (ATtiny4 to 10) are not yet implemented.
4. Older AVR types, such as the AT90S, are implemented, but are not yet systematically tested for correct execution.
5. As proposed by Kenji I work on an implementation of the fuses into `avr_sim`. The first step has been made, but there is still a lot to do.

8 Not implemented and not planned

The following features are neither implemented nor planned:

1. Serial communication via Async Universal or Sync Serial modes is not implemented and not planned.
2. Driving LCD displays is not and will not be implemented.
3. Those who generate enormous C code and convert it to even more voluminous asm source code will not be very happy with avr_sim and gaviasm: from 10,000 lines of code on neither avr_sim nor gaviasm work correct, just because the line numbers exceed their five-digit limit and cause crashes and malfunctions. I do not plan to extend line numbers to whatever larger limit, instead I recommend: "Better learn a more rational programming language instead of littering the world with inefficient and superfluous code lines!".

9 Changes introduced in version 2.3

The following changes were introduced in this version.

9.1 Changes to the integrated assembler gavrasm

gavrasm has been upgraded to version 4.9. The following changes were made:

1. With the ATmega64RFR2, the 128RFR2 and the 256RFR2 the use of the instructions ADIW and SBIW caused errors. This is corrected.
2. The instructions JMP and CALL were falsely disabled in the ATmega16M1.
3. In the original def.inc file for the device ATtiny48A the bits PORF, EXTRF, BORF and WDRF are not defined. These are now added by the assembler, if not defined in the def.inc.
4. The processing of very large 64-bit-integers had a bug and did not allow the complete range to be used.
5. The assembler now handles `//`, `/*` and `*/` as additional commenting tools.

9.2 Changes concerning the editor

1. When deleting a multi-line text block the cursor appeared at the previous end position of the block, and hence two or more lines below the current position. This is corrected.
2. The context menu entry "Go to source" did not work correct. This is corrected.
3. When opening an asm source code file with the menu entry "Open from asm" with a false `.include "[dev]def.inc"` entry, avr_sim crashed with a range check error. Now this can be corrected in the editor, the changed asm file can be saved and the project should then be re-opened.
4. Activation of deactivated breakpoints failed and affected the next breakpoint in the list. This is corrected.
5. Deleting a breakpoint lead to a deactivation of the first breakpoint in the

list. This was caused by the Lazarus stringgrid component, that already activates the stringgrid change event with column=0 and line=1 when the stringgrid is simply written to. The stringgrid event is blocked now and requires a prior mouse-down event to function.

6. Moving a breakpoint to another location within the list-file can be initiated with a click into the breakpoint's cell under the column "List line".
7. Setting/deleting breakpoints can also be done by clicking into the gutter area of the line (the left-most column in the editor).
8. The editor now handles // and /* as well as */ as commenting tools. The syntax high-lighting for this is not yet perfect (scroll up or down a bit).

9.3 Changes concerning the simulator

1. When editing breakpoints it is now possible to also enter the line in the listing. (Previously only the source code line entries were editable, which caused some additional difficulties in finding the source code line because of the error in the context menu entry "Go to source").
2. Breakpoints are now checked after assembling. Moved lines are searched for upwards, when unsuccessful also downwards. Unless you changed the breakpoints line content, moved breakpoints should be identified correct.
3. After closing the Scope-Control-window with the standard button avr_sim did not work any more, because this window is modal. Closing those hardware-displaying windows now is only possible with the checkbox. That also ensures that the entry of the window states in the project file is correct.
4. When using .DEVICE without enclosing the device name in double " an error message occurred. This is corrected.
5. The instructions ADIW and SBIW did not affect the carry flag. This has been added.
6. In AD conversion devices that have ADCSRB the ADLAR bit was not correctly applied.

7. When using LD Rn,-Z the register pair Z was decreased, but not stored. This was correct in the case of LD Rn,-X and LD Rn,-Y, but not with -Z. This error has been corrected.
8. To Do: Assembling for m414 and similar devices an error message occurs
9. Writing of ones to INTF flags now clears the flag.
10. Changing the clock frequency by writing to the CLKPR port register now is correctly reflected in the project status window. The CLKDIV8 fuse is then inactive.
11. When setting a breakpoint into a macro line in the listing, the breakpoint points to the macro name and the breakpoint is missed. This is not corrected in this version!
12. When simulating the ATtiny25/45/85's Timer/Counter 1, incorrect prescaler values were displayed and in case those were above 1,024 were not applied correctly. This is corrected.

9.4 Changes of the handbook

1. In Chapter [2.2](#) I added a description on how to avoid Linker error messages of the type UNDEFINED SYMBOL.
2. In Chapter 4.6 a more detailed description of the "Comprehensive" and "Interrupts" selection has been added.
3. Added description of the alert function in [chapter 6.9](#).

10 Known issues

The Windows version, when used with a different screen resolution, does not show sub-windows and their components as designed but incomplete. The reason for this behavior is not understood by me. Hints on how this can be avoided in the Lazarus environment are very welcome.

Even though I change the window designs in Linux (due to different fonts used) I might have forgotten one or two of these. If you want to make changes to the design, especially if you use a large screen and/or if you prefer larger fonts, use Lazarus and the source code files and compile your own special `avr_sim` version.

Under my Linux (Suse 42.3) it happens that active simulations are interrupted if I click on another application or onto the window frames of `avr_sim`. The times are cleared and the simulation restarts at zero or one. Hints on where to search for the reasons are welcome.