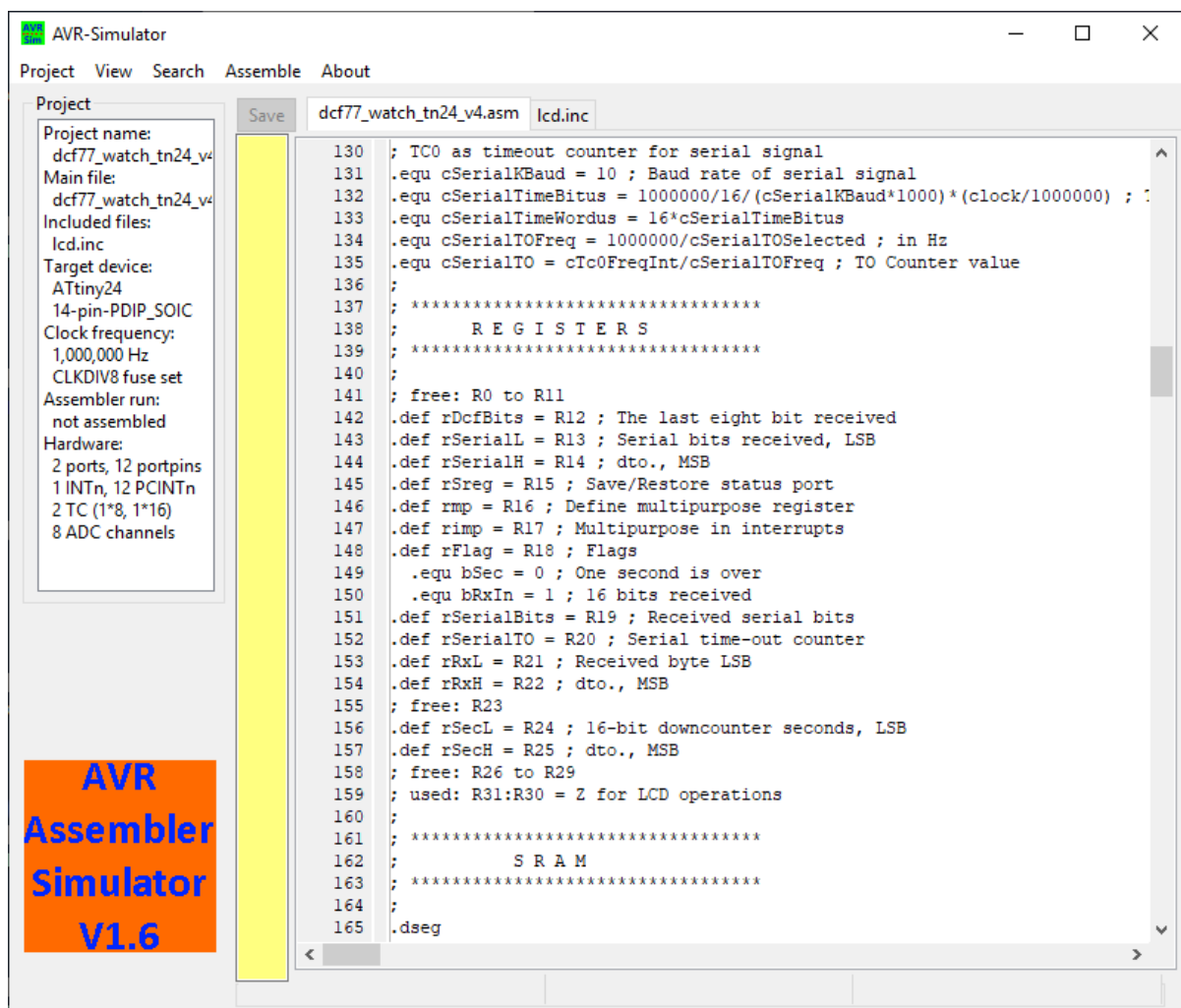


Installation and use of the the AVR assembler simulator avr_sim



by

Gerhard Schmidt, Kastanienallee 20, D-64289 Darmstadt

gavrasn@avr-asm-tutorial.net

Version 1.6 as of July 2019

Content

1	Application conditions and hints	4
1.1	What this software does	4
1.2	Applicability	4
1.3	Hints for using avr_sim	5
1.4	Lengthy timer operations	6
2	Compiling avr_sim	7
2.1	Installing Lazarus	7
2.2	Compiling avr_sim	7
3	Installation	9
3.1	Installing gavrasn	9
3.2	Running avr_sim for the first time	9
4	Use	10
4.1	Opening files	10
4.2	Device check	11
4.3	Editing files	12
4.3.1	The editor	12
4.3.2	The editor's functions	13
4.3.3	Search for and/or replacing text	14
4.3.4	Working in parallel with an external editor	15
4.4	Selecting the device's clock frequency	15
4.5	Viewing device properties	16
4.6	Starting a new project	17
4.7	Adding a new include file to an existing project	19
5	Assembling	20
5.1	Starting the assembly process	20
5.2	Errors during assembling	21
5.3	Setting/removing breakpoints	21
6	Simulating	24
6.1	Starting simulation	24
6.2	Viewing I/O ports	25
6.3	Viewing and manipulating timers	28
6.3.1	Viewing timers	28
6.3.2	Changing timer values	30
6.4	Viewing SRAM content	30
6.5	Viewing EEPROM content	30
6.6	Viewing and manipulating ADC content	32
6.6.1	ADC channels	32
6.6.2	Resistor matrix as voltage input source	33
6.6.3	Active AD conversion	35
6.7	Scope display	36
7	Not yet implemented, but under consideration	39
8	Not implemented and not planned	40
9	Changes introduced in version 1.6	41
9.1	Changes to the integrated assembler gavrasn	41
9.2	Changes concerning the editor	41
9.3	Changes concerning the simulator	41
9.4	Changes of the handbook	42

10	Known issues	43
----	--------------------	----

1 Application conditions and hints

1.1 What this software does

This software

- emulates micro controllers of the types AT90S, ATtiny and ATmega,
- allows to edit assembler source code 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 enables 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 und ATmega devices, in limited manner also for ATxmega, 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 such as 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 with the directive `.include „(type abbrev.)def.inc“`. Sub types such as „A“ or „P“ suffixes are generally correctly recognized if there is an include file from ATMEL® or Microchip® available. The different availability of I/O port bits in different packages such as PDIP, SOIC or QFN are correctly recog-

nized 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](#) 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).
5. As like for all free software of this type no guarantee for bug-free functioning can be given.

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 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.

If only parts of the code shall be tested, unused parts of the source code can be commented out with „;“. With the directives „.if (condition)“, „.else“ and „.endif“ 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. If certain single steps shall not be executed, use the “Skip” mode for those.

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.

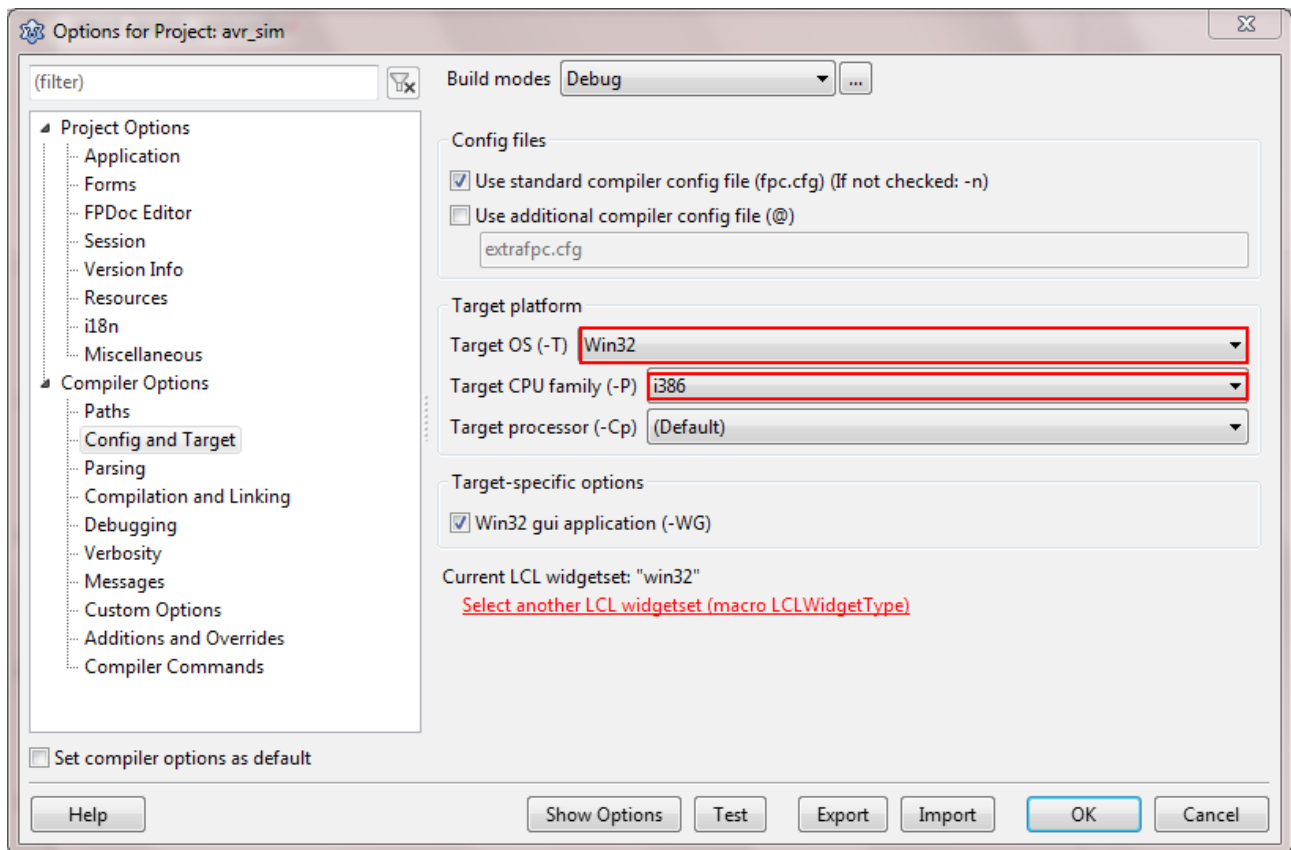
2.1 Installing Lazarus

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 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

Then go through the forms 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.

3 Installation

3.1 Installing gavasm

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

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).

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 where the executable is located, therefore it is better to install avr_sim in a folder where the user has write permissions to.

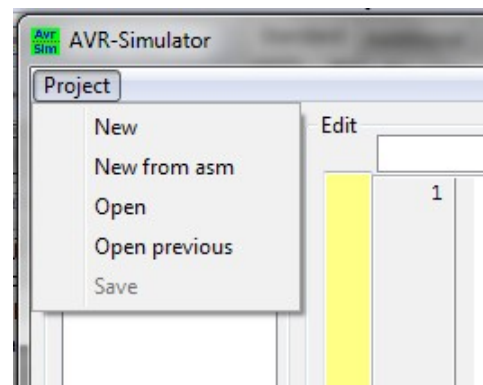
4 Use

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 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, need to be located in the project folder. Relative paths, such as .include „..\location\example.inc“ can be used.

Note that avr_sim is not designed to handle blanks in path or file names. Please replace those by the underscore character.

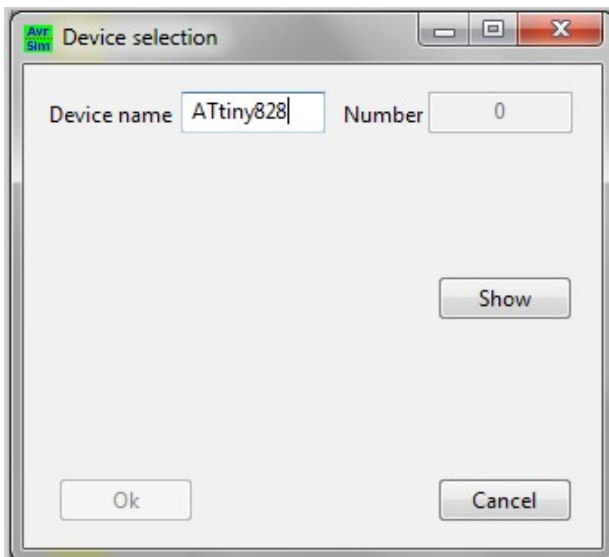
avr_sim can start a given project file directly if you associate .pro files once with the avr_sim executable. Right-click on a .pro file, select „Open with ...“, and „Select standard program“, tick the entry field „Associate file type permanently“ and navigate to the avr_sim executable). If you then 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 re-

pair this you need to edit the Windows Registry file. Another work-around is to rename the `avr_sim.exe` file to `avr_sim_13.exe`: by doing that you can associate the `.pro` files with a certain version of `avr_sim`.

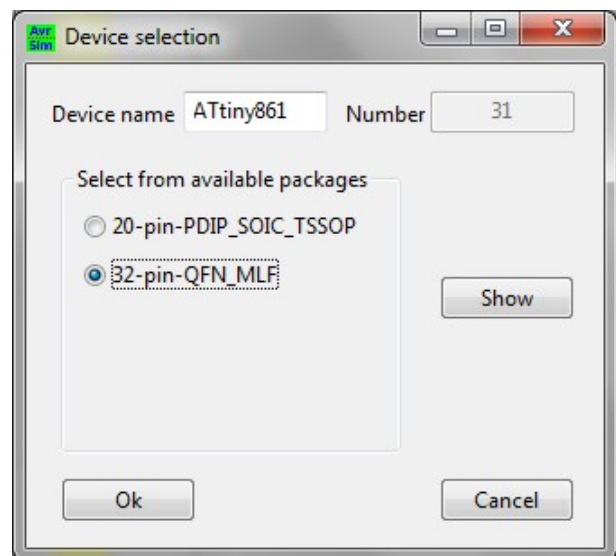
In the same manner `avr_sim` can be associated with `.asm` files. Either decide to associate `.asm` files directly with `avr_sim` or with a right-click select „Open with ...“ to start the `.asm` file with `avr_sim`. If a project file already exists you are asked if you want to overwrite this. If yes,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

4.2 Device check

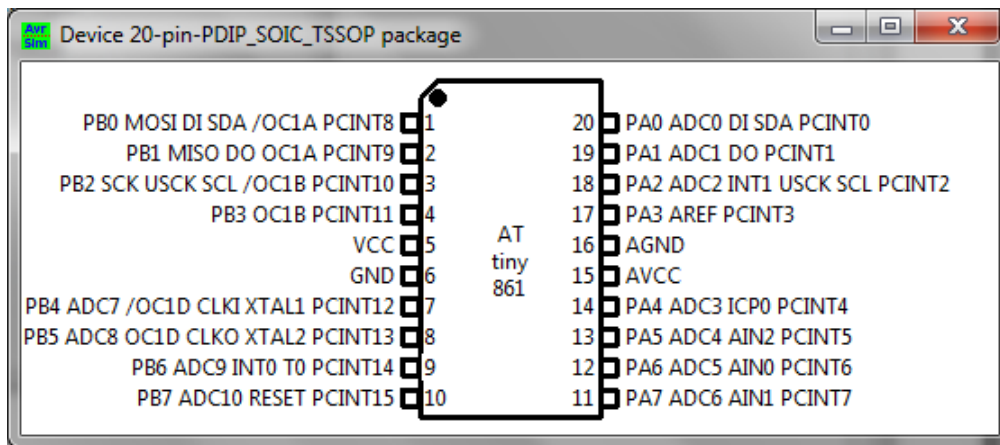
The device type is derived from the `.include „xxxdef.inc“` directive in the source code. In case that the device type is not included in the database of `avr_sim`, a selection window opens with „0“ as device number.



By changing the device name to a similar device type, a different device type can be selected.



If the selected device type comes in different packages, the package type can be selected. If you need assistance for the package type, click on "Show" to display the selected package type.



The selected package type should associate counter input pins, INTn and PCINTn pins to similar port

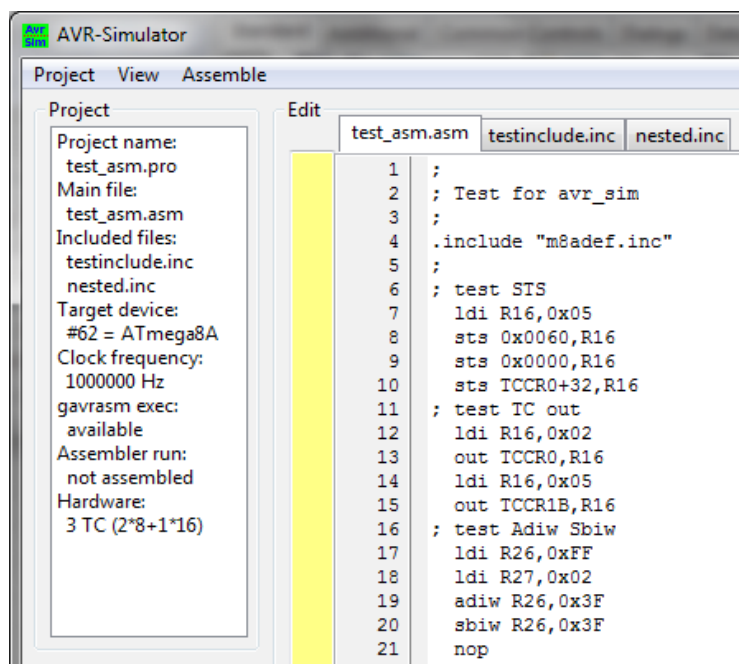
bits in the target device.

If the device number is larger than zero, the device type is known. If there are different package types, select the correct one, because there might be differences in the availability of I/O pins.

If you want to continue, press the Ok button. The device type is stored in the .pro file, so you are only asked once for the package. If you want to select another package type, just delete the respective line from the .pro file with a text editor.

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 selectable 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 an-

swer „Yes“ to save the file (if „No“, the changes are ignored).

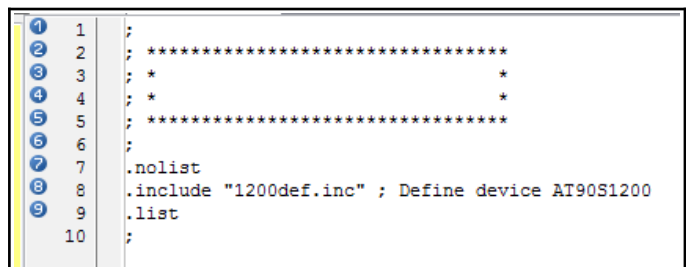
4.3.2 The editor's functions

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

- Alt-Backspace or Strg-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-Strg-0 combination does not work, 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.

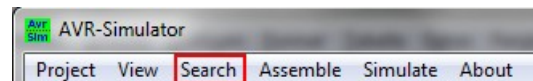
```

71 ; *****
72 ;   H A R D W A R E
73 ; *****
74 ;
75 ; Device: ATmega8, Package: 28-pin-PDIP
76 ;
77 ;
78 ;           1 / 128
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 ;

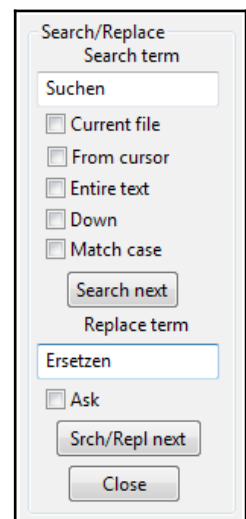
```

4.3.3 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.

4.3.4 Working in parallel with an external editor

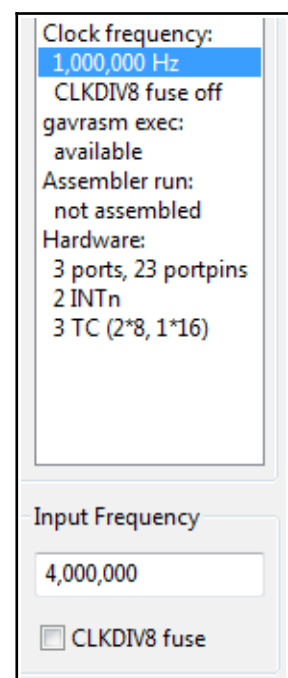
If you sometimes work with an external editor please consider the following:

- 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 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.

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 imme-



Clock frequency:
1,000,000 Hz
CLKDIV8 fuse off
gavasm exec:
available
Assembler run:
not assembled
Hardware:
3 ports, 23 portpins
2 INTn
3 TC (2*8, 1*16)

Input Frequency
4,000,000
☐ CLKDIV8 fuse

diately.

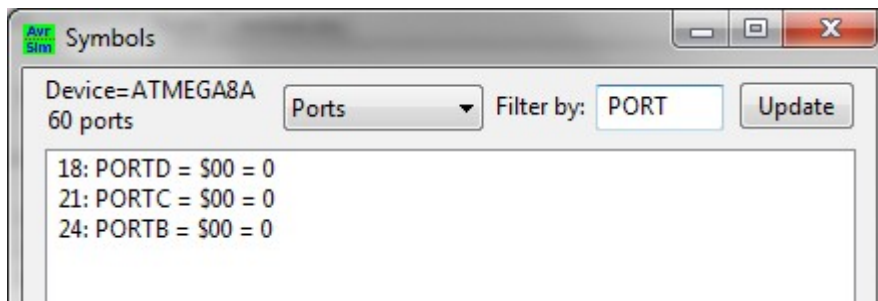
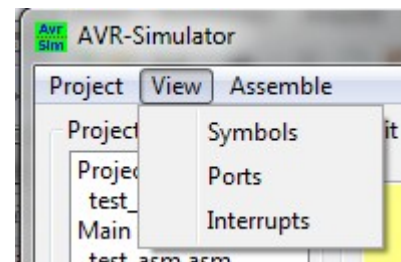
If in your device the CLKDIV8 fuse is set, set the respective option, too. If, during runtime, the software changes the clock prescaler (by writing to the port register CLKPR) this change is recognized by avr_sim and the frequency is changed during simulation.

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

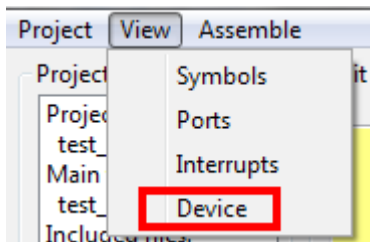
The menu item „View“ offers to display

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),
3. all interrupts that the device can handle (subset of the symbol set).

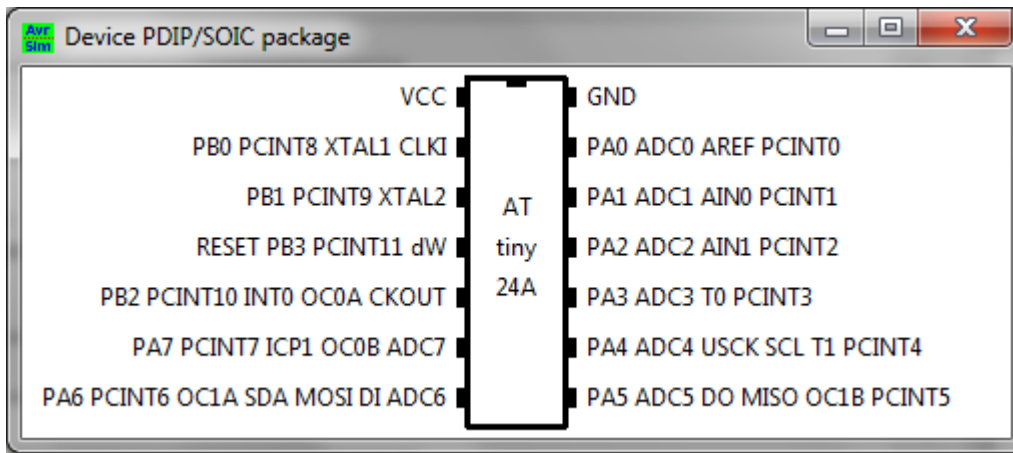


The view window offers a filter property so that a limited symbol set can be displayed.

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

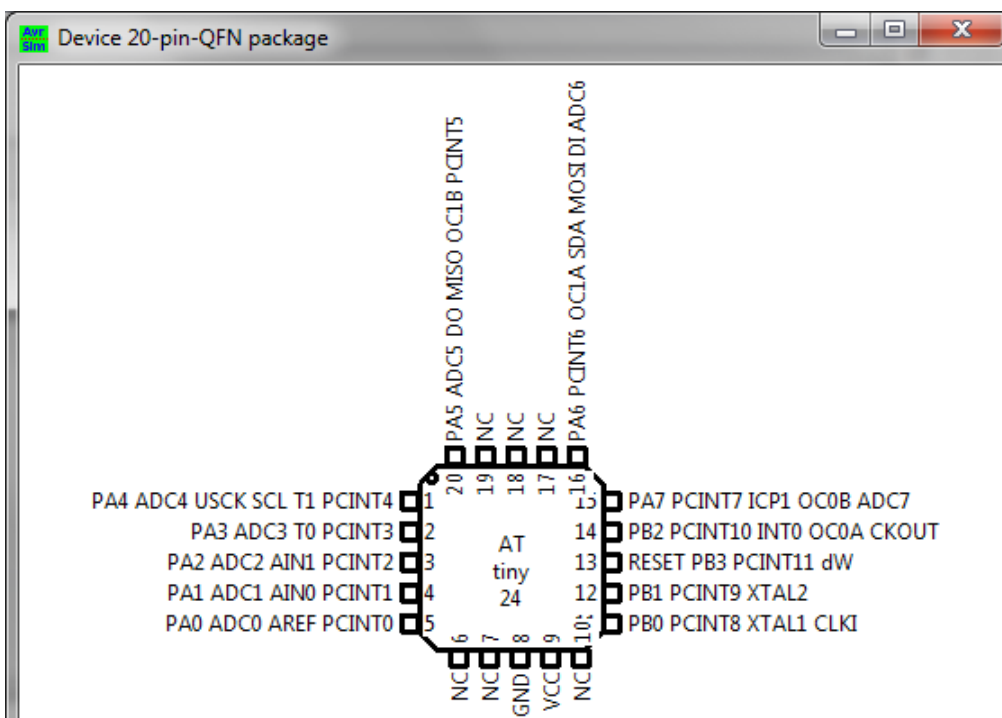


If the menu entry „Device“ shows up, the device and its pin assignments can be painted. If not, make sure that you have the directive `.include "typedef.inc"` in your code, from which the device is derived. As many basic information is depending from the device (memory sizes, internal hardware, etc.) simulation does not work in this case.



If "Device" is selected from the "View" menu all pin assignments of the device are shown for the device in

its PDIP or SOIC package. If a different package type has been selected, those are displayed accordingly.



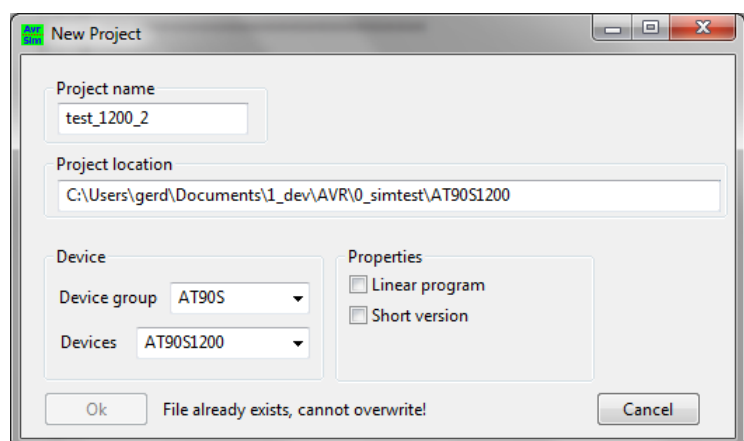
This is the same AVR type in a different package (20-pin QFN).

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

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



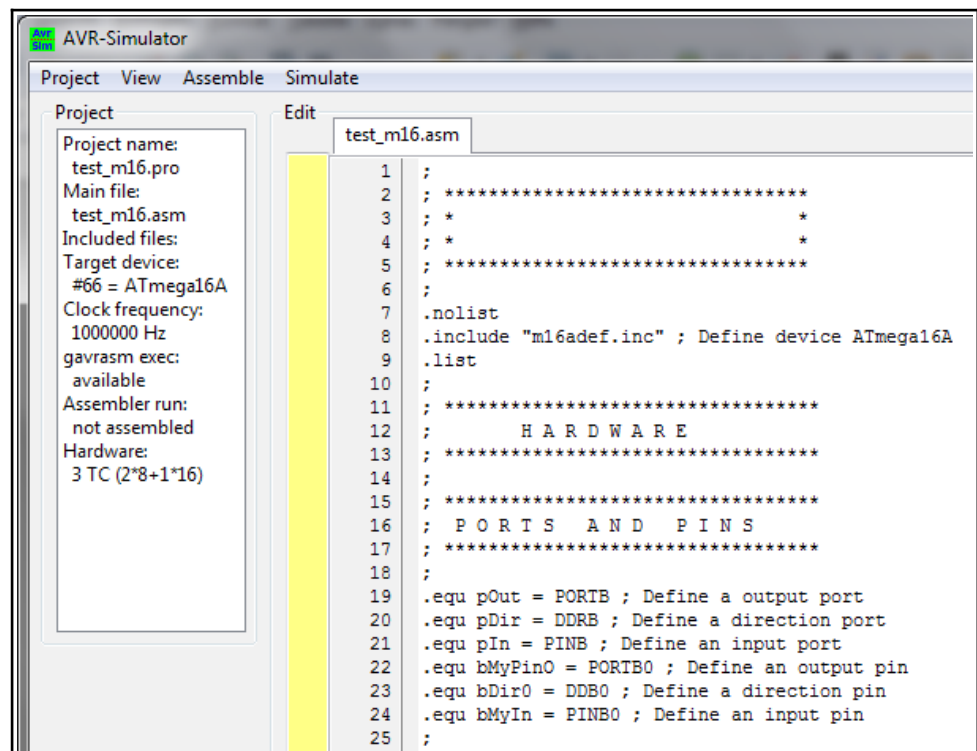
and path refer to an already existing file, an error message occurs. Further, select a device that the project shall be made for. With the two property selections

- Interrupt or linear, and
- Comprehensive or short,

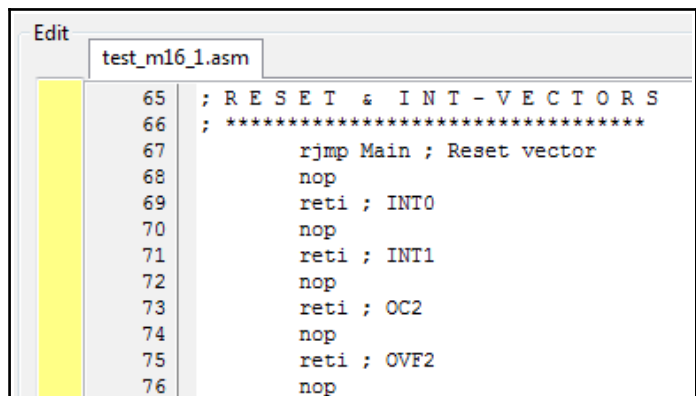
the type of the generated standard template can be influenced.

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

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



Note that if „Interrupts enabled“ and “Comprehensive” has been chosen, the new file already holds all reset and interrupt vectors of the device.



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 `.include`.

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.

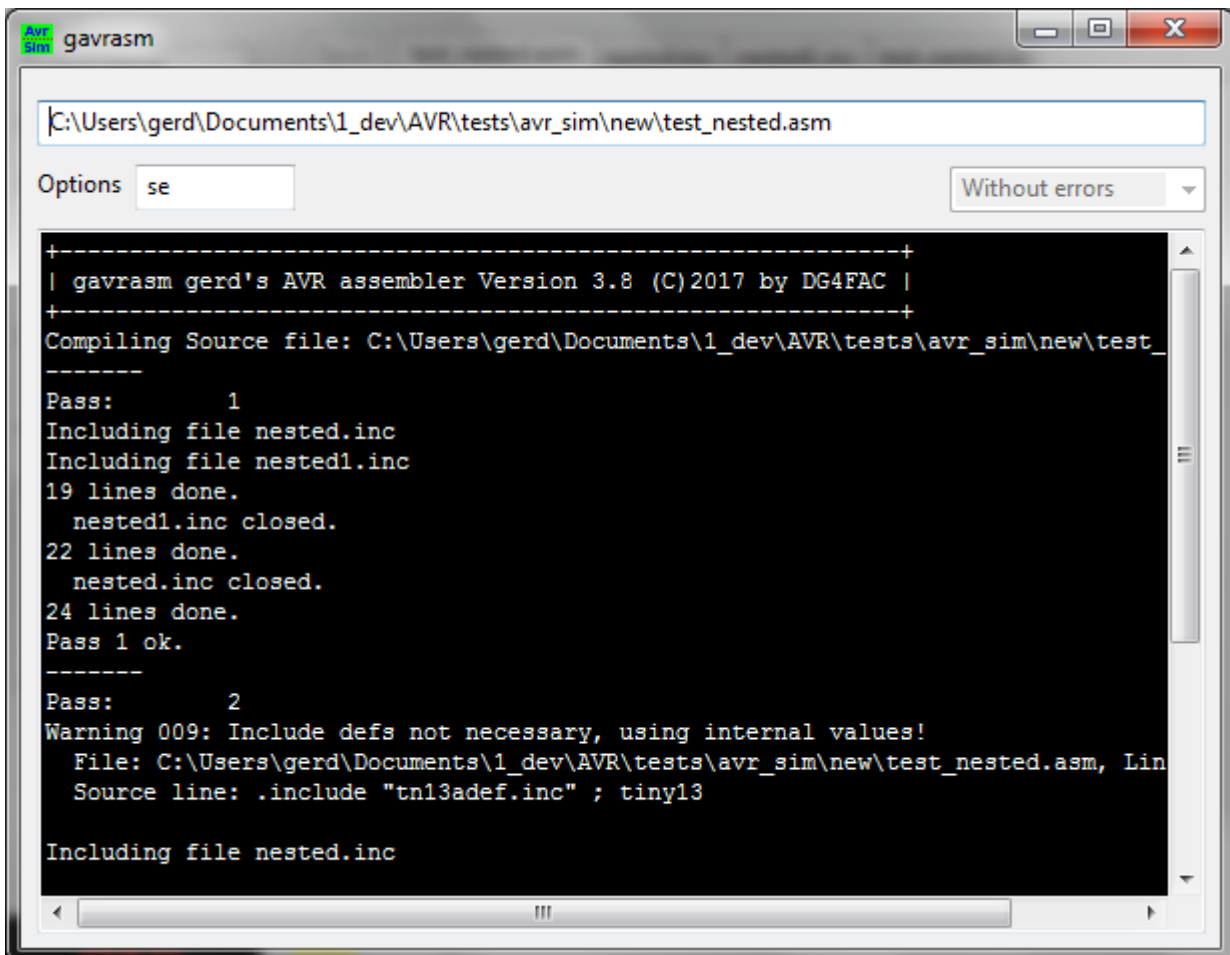
All included files that have an include directive entry (`.include "[inc name]"`) are recognized during the load process of the main file. Those are added automatically to the tab entries and are directly accessible. Excluded are `*def.inc` files which are used to identify the selected AVR type but have no tab entry.

Nested includes (included files that include other include files) are possible.

5 Assembling

5.1 Starting the assembly process

By pressing the menu item „Assemble“ the internal gavrasn assembles the source file (and all associated include files). After gavrasn ended, a message



The screenshot shows the gavrasn application window. The title bar reads 'gavrasn'. The address bar shows the file path: 'C:\Users\gerd\Documents\1_dev\AVR\tests\avr_sim\new\test_nested.asm'. Below the address bar, there is an 'Options' field with the value 'se' and a dropdown menu set to 'Without errors'. The main text area displays the assembly output:

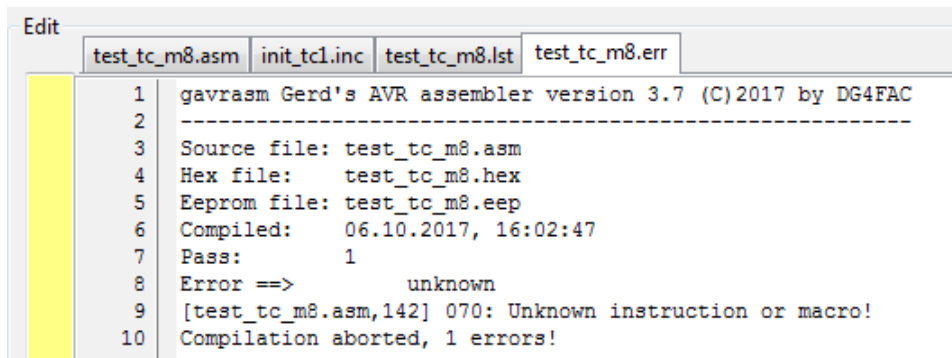
```
+-----+
| gavrasn gerd's AVR assembler Version 3.8 (C)2017 by DG4FAC |
+-----+
Compiling Source file: C:\Users\gerd\Documents\1_dev\AVR\tests\avr_sim\new\test_
-----
Pass:      1
Including file nested.inc
Including file nested1.inc
19 lines done.
    nested1.inc closed.
22 lines done.
    nested.inc closed.
24 lines done.
Pass 1 ok.
-----
Pass:      2
Warning 009: Include defs not necessary, using internal values!
    File: C:\Users\gerd\Documents\1_dev\AVR\tests\avr_sim\new\test_nested.asm, Lin
    Source line: .include "tn13adef.inc" ; tiny13

Including file nested.inc
```

signals errors or success and the generated listing plus eventually the generated error file „.err“ is added to the editor tabs.

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



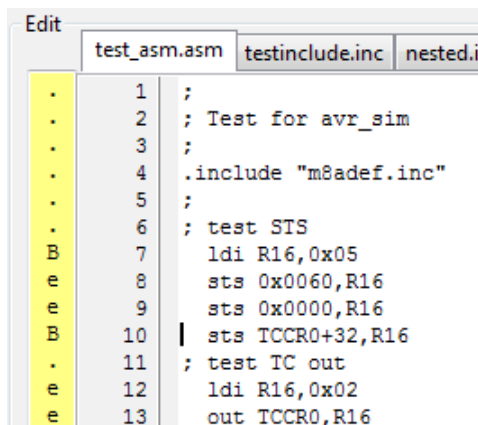
The screenshot shows the 'Edit' window of the AVR assembler. The tabs at the top are 'test_tc_m8.asm', 'init_tc1.inc', 'test_tc_m8.lst', and 'test_tc_m8.err'. The 'test_tc_m8.err' tab is active, displaying the following error message:

```
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



The screenshot shows the 'Edit' window of the AVR assembler. The tabs at the top are 'test_asm.asm', 'testinclude.inc', and 'nested.i'. The 'test_asm.asm' tab is active, displaying the following code:

```
1  ;
2  ; Test for avr_sim
3  ;
4  .include "m8sdef.inc"
5  ;
6  ; test STS
7  ldi R16,0x05
8  sts 0x0060,R16
9  sts 0x0000,R16
10 | sts TCRC0+32,R16
11 ; test TC out
12 ldi R16,0x02
13 out TCRC0,R16
```

Lines 7, 8, 9, 10, 12, and 13 are highlighted in yellow, indicating they are executable instructions. A vertical cursor is positioned at the start of line 10.

Lines that have an „e” in the yellow side window are identified as executable instructions. By setting the cursor into those lines and right-clicking with the mouse a context 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 are only displayed after assembling the project.

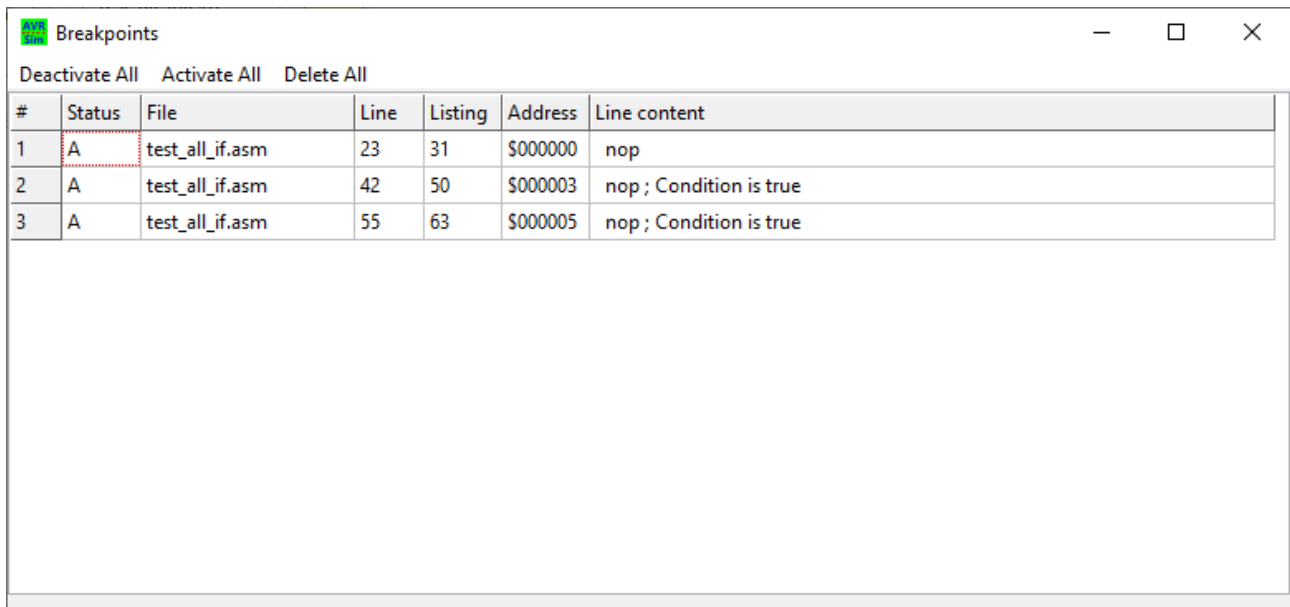
Breakpoints are copied to the list file, that is also available as tab file following assembling. In the list file breakpoints can be enabled and disabled as well, those changes are copied to the original .asm and .inc files where they result from.

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.



#	Status	File	Line	Listing	Address	Line content
1	A	test_all_if.asm	23	31	\$000000	nop
2	A	test_all_if.asm	42	50	\$000003	nop ; Condition is true
3	A	test_all_if.asm	55	63	\$000005	nop ; Condition is true

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 disactivates this breakpoint. 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, deactivate or delete all breakpoints. A maximum of 24 different breakpoints can be handled.

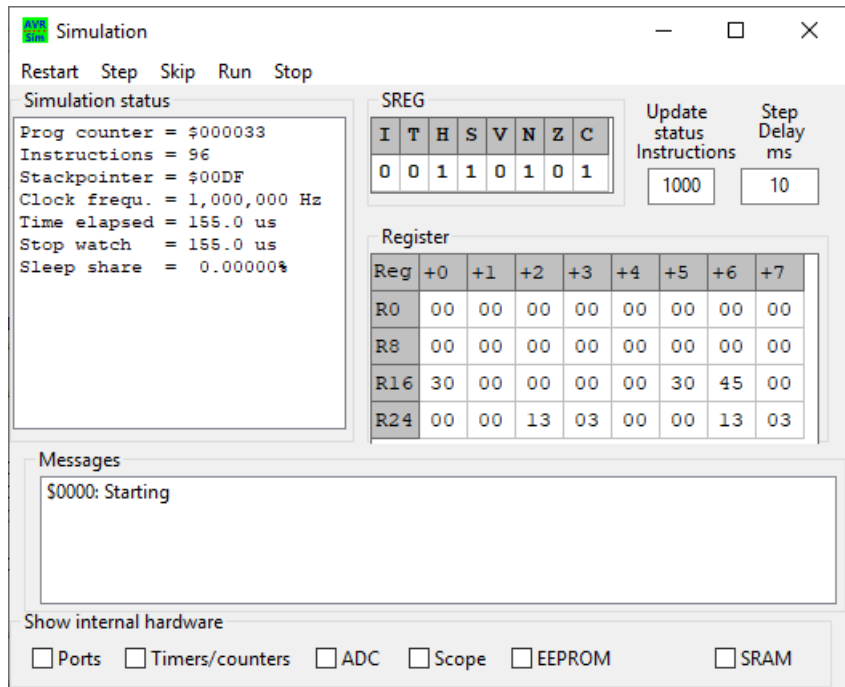
All changes made come immediately in effect, even during a running simulation.

When changes to the sources text are made, avr_sim tries to reflect those changes, but the breakpoint allocation is not always correct. Following successful re-assembling the breakpoints are checked. If errors occur the related breakpoints are deactivated and can be manually corrected in the "View/Breakpoints" window.

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: Starts the program with a reset (program counter = 0), clears all hardware.
2. Step: Perform a single step with the next executable instruction (the editor windows shows a „>“ on this instruction or a „<“ if a breakpoint is located there).
3. Skip: Jumps over the next executable instruction. Please use this carefully because jumping over RET or RETI instructions leaves the stack in disorder.
4. Run: Runs indefinitely until
 - a breakpoint is encountered, or
 - the menu item „Stop“ is clicked, or
 - the PC counter runs onto an illegal address location.
5. Stop: 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.

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

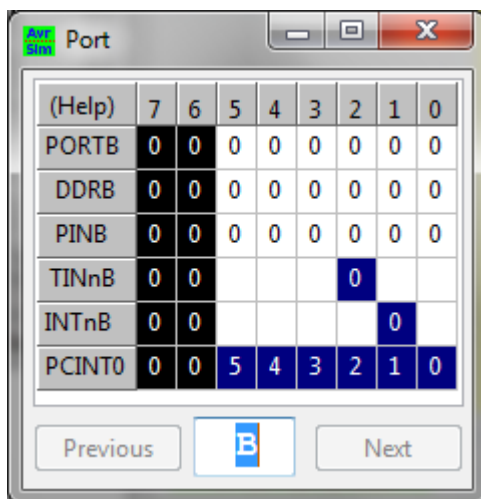
Elapsed time, the stop watch and the SLEEP share are cleared if a right-click is made on those entries. Please note that clearing elapsed time also clears the scope event storage.

The edit field 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 10 ms. Changes come into effect immediately.

The message window provides information on unimplemented or illegal instructions. 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“.

(Help)	7	6	5	4	3	2	1	0
PORTB	0	0	1	0	0	0	0	0
DDRB	0	0	1	0	0	0	0	0
PINB	0	0	Lo	0	0	0	0	0
INTnB	0	0					0	
PCINT0	0	0	5	4	3	2	1	0

Previous **B** Next

If the DDR bit in the port is set, the respective PIN bit in the read port register PIN follows the PORT bit.

(Help)	7	6	5	4	3	2	1	0
PORTB	0	0	1	0	0	0	0	0
DDRB	0	0	1	0	0	0	0	0
PINB	0	0	Hi	0	0	0	0	0
INTnB	0	0					0	
PCINT0	0	0	5	4	3	2	1	0

Previous **B** Next

(Help)	7	6	5	4	3	2	1	0
PORTB	0	0	1	0	0	0	0	0
DDRB	0	0	0	0	0	0	0	0
PINB	0	0	Pu	0	0	0	0	0
INTnB	0	0					0	
PCINT0	0	0	5	4	3	2	1	0

Previous **B** Next

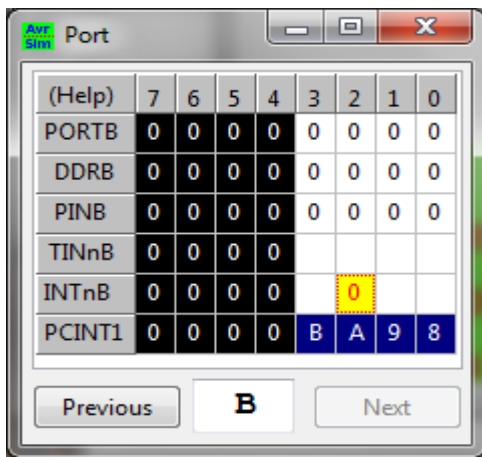
If the PORTB bit is set while the respective DDRB bit is cleared the PINB bit is marked as pulled-up.

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.

(Help)	7	6	5	4	3	2	1	0
PORTB	0	0	0	0	0	0	0	0
DDRB	0	0	0	0	0	0	0	0
PINB	0	0	0	0	0	0	0	0
TINnB	0	0				0		
INTnB	0	0					0	
PCINT0	0	0	5	4	3	2	1	0

Previous **B** Next



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.

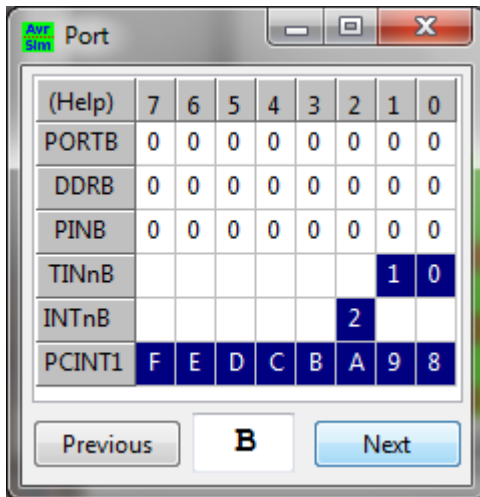
The color turns to red, when the interrupt is executed (if the I bit in the status register is set and the next step is executed). On execution of the RETI instruction the interrupt terminates.

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.

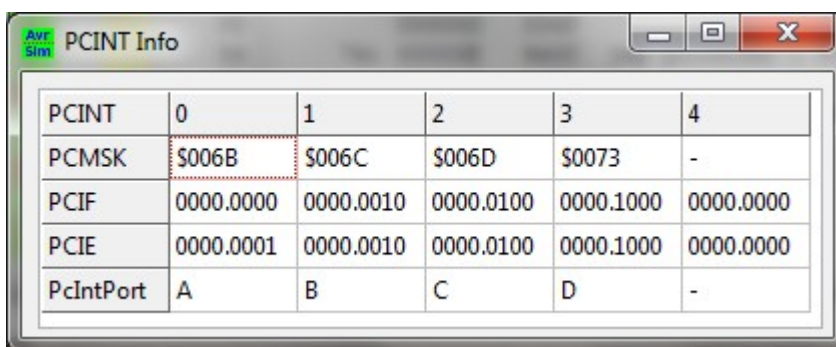
The color explanation occurs by pressing the (Help) entry in column 1 line 1 of the grid.

Portpins	
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 INT0 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



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.

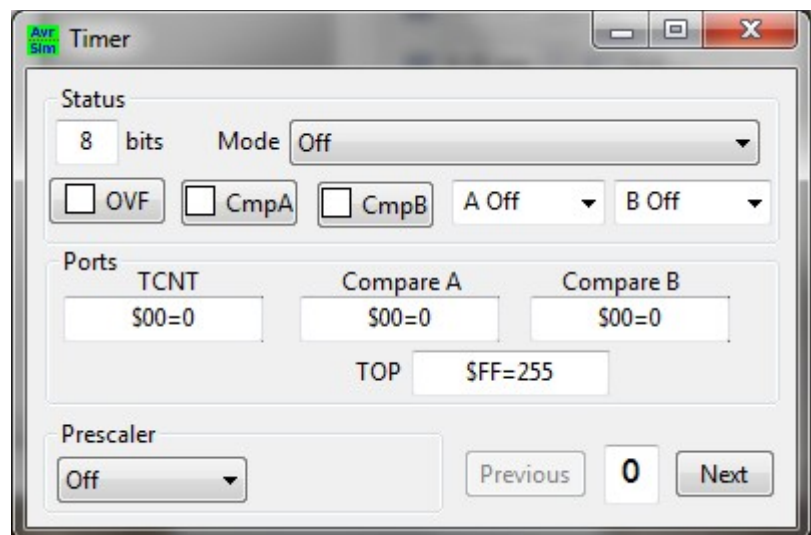


By clicking the row header PCINTn more information on the PCINTs is provided.

6.3 Viewing and manipulating timers

6.3.1 Viewing timers

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



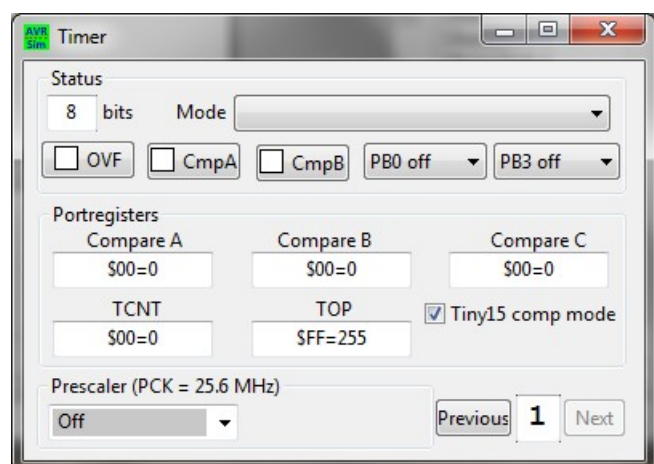
The following information is available:

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.



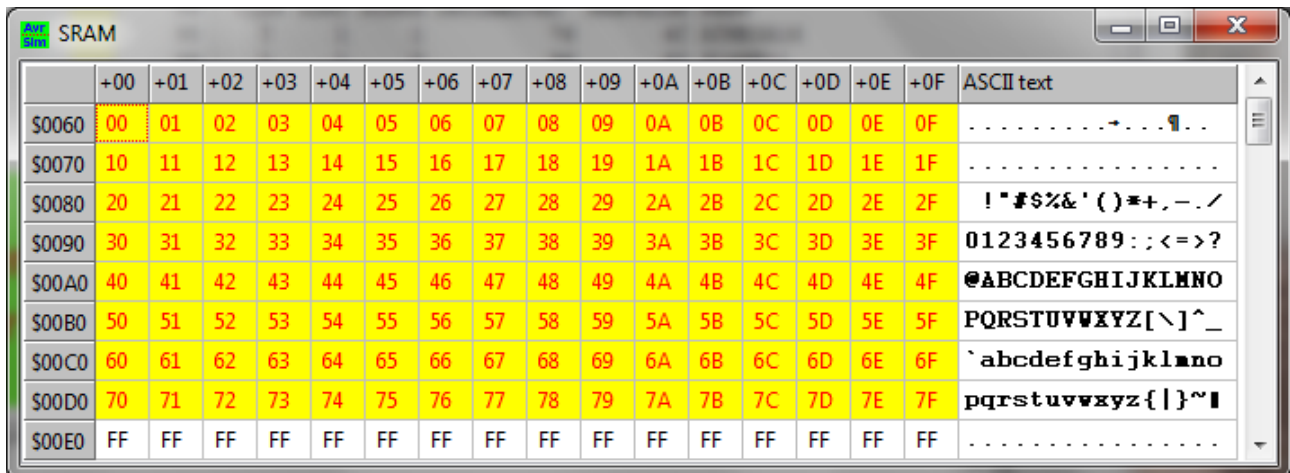
6.3.2 Changing timer values

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 Viewing SRAM content

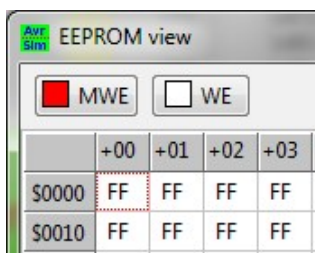
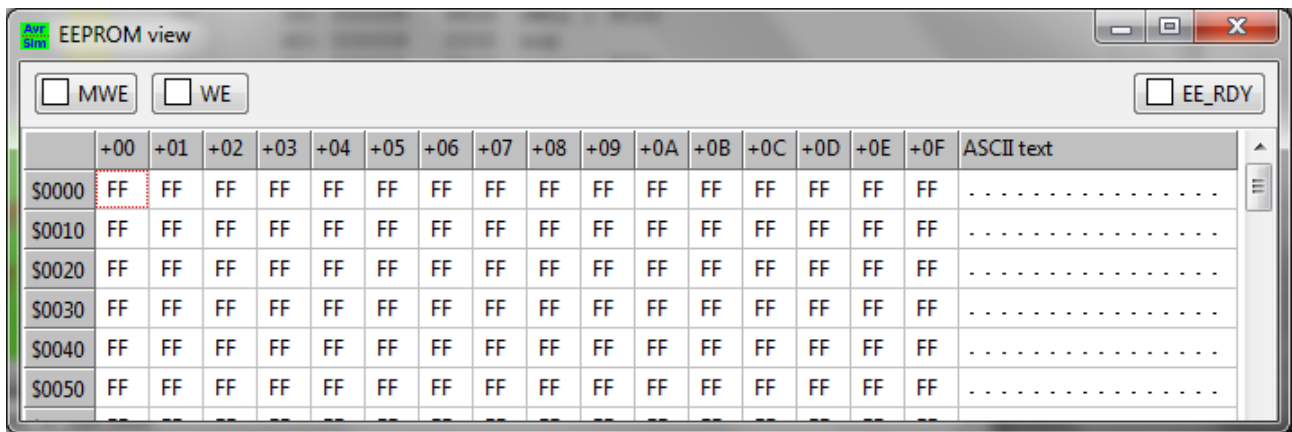
With the checkbox "SRAM" enabled the content of the SRAM is displayed. Changed values appear in red on a yellow background.



	+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	`abcdefghijklmno
\$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

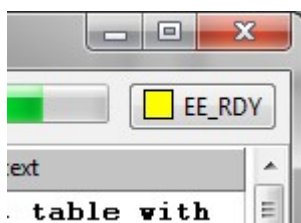
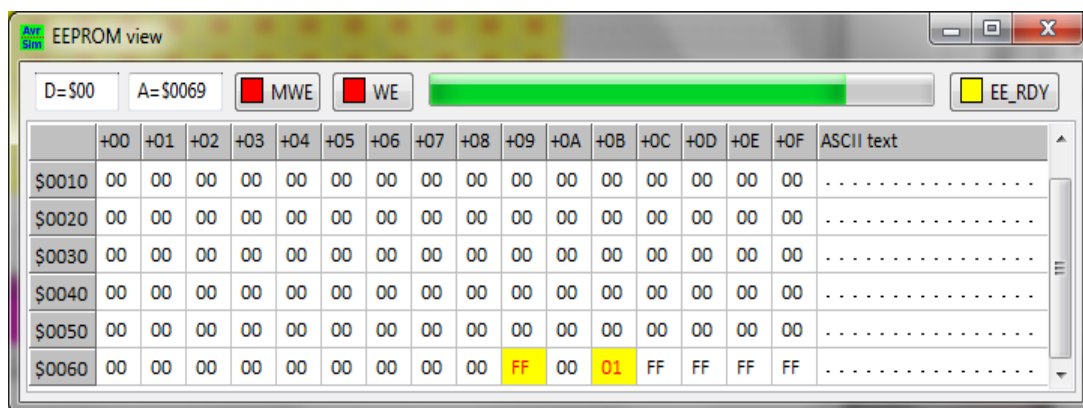
6.5 Viewing EEPROM content

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

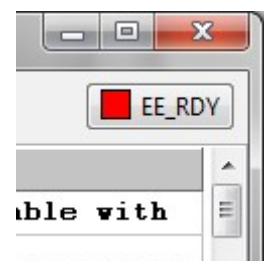


If the Master-Write-Enable bit (in some devices called Master-Programming-Enable bit) is set, this is displayed like this. This bit is cleared after four clock cycles.

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.

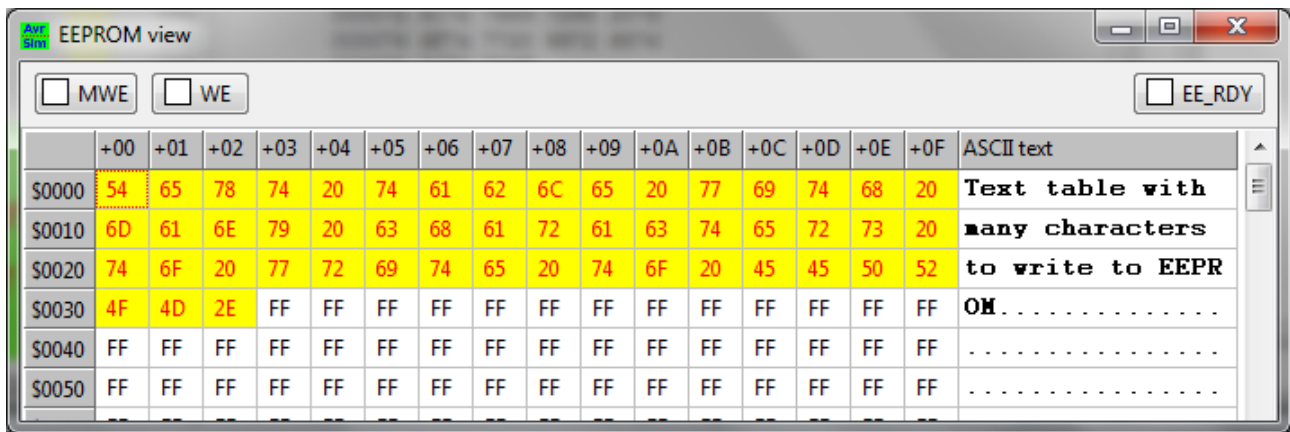


The yellow sign shows if the EE_READY interrupt is enabled. A red one shows that the execution of the interrupt is under progress. Please note that the interrupt is repeated as long as the interrupt enable



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

Red on yellow characters are changed locations.

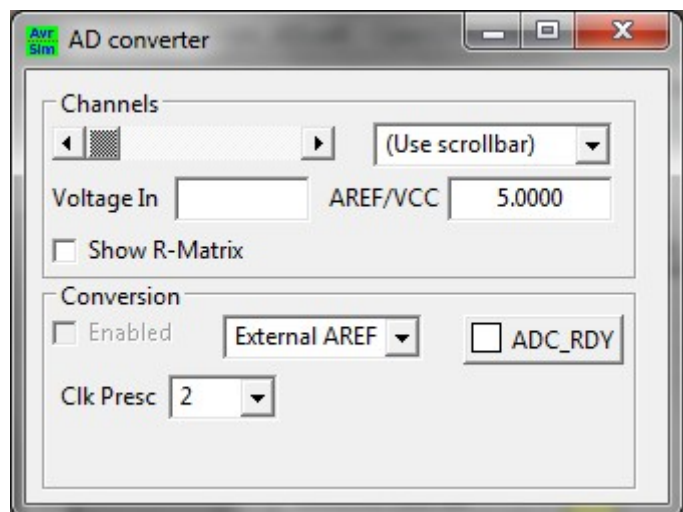


6.6 Viewing and manipulating ADC content

6.6.1 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.



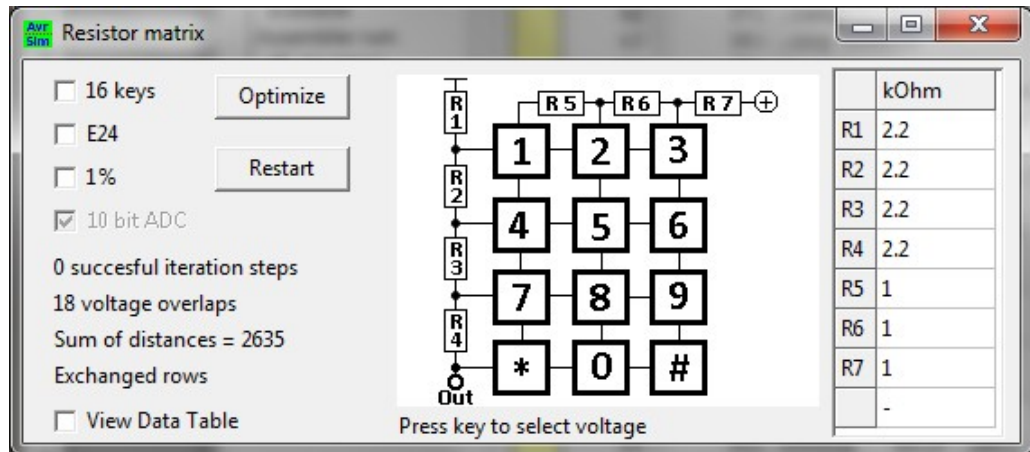
Selection of the channel changes the current ADMUX values. Those changes come only into effect when a new conversion starts.

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.

6.6.2 Resistor matrix as voltage input source

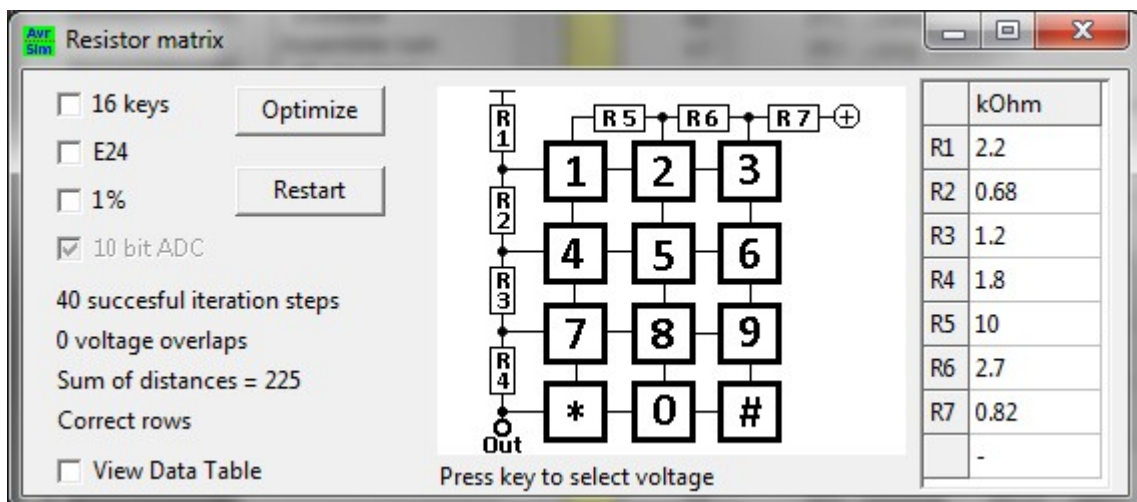
The R-Matrix field allows to open a window with a resistor matrix that generates input voltages for the ADC.



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.

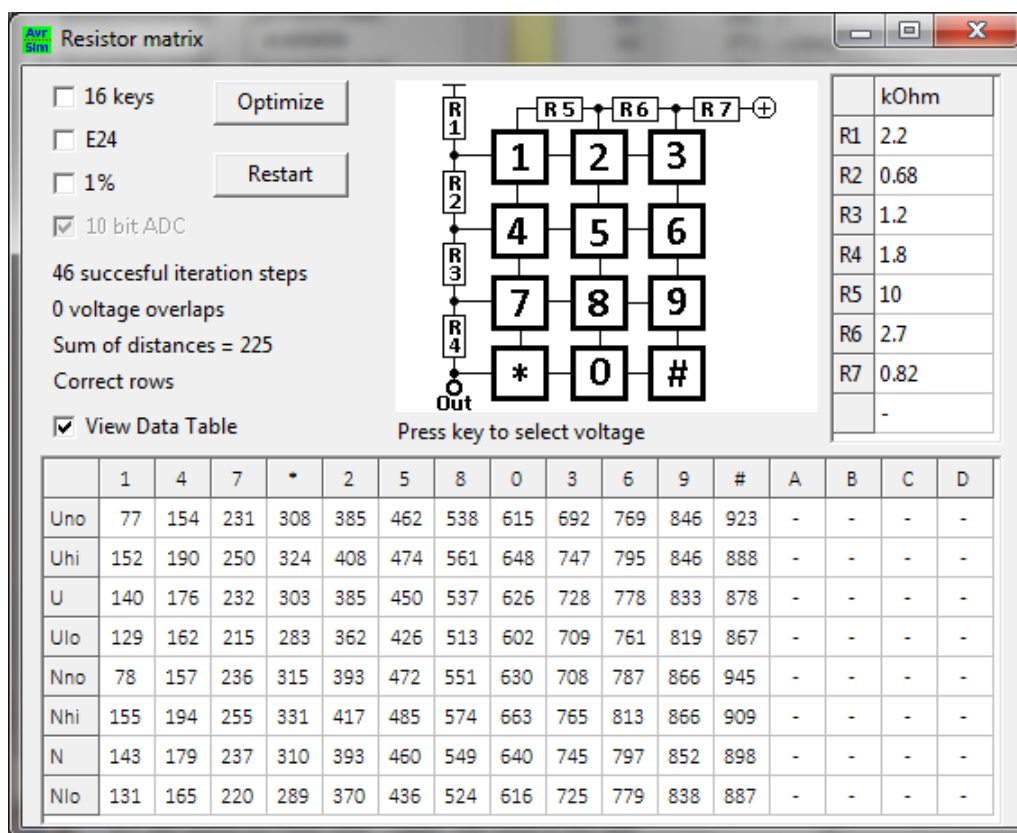


Further optimization can be done by allowing resistor values of the E24 row (default is E12).

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-#“.

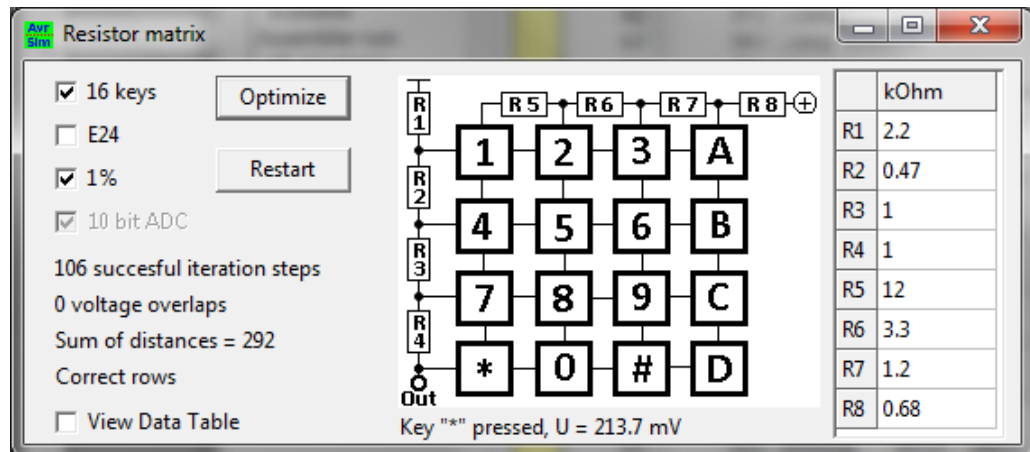
By checking the „View Data Table“ box the voltages in mV (at an operating voltage of 1 V) and the resulting ADC values (8-bit resolution if the ADLAR bit in ADMUX is set, 10 bit if not) can be displayed. In this table the following lines mean:

- 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 in-



creases in some cases without changing resistor values.

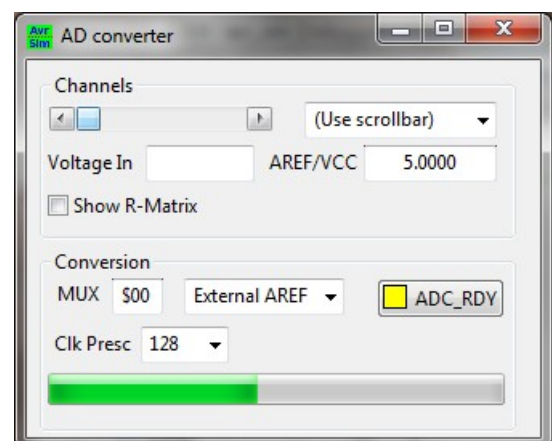
The correct row with increasing voltages is „1-4-7-*-2-5-8-0-3-6-9-#-A-B-C-D“.

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.

6.6.3 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 ADSC 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.

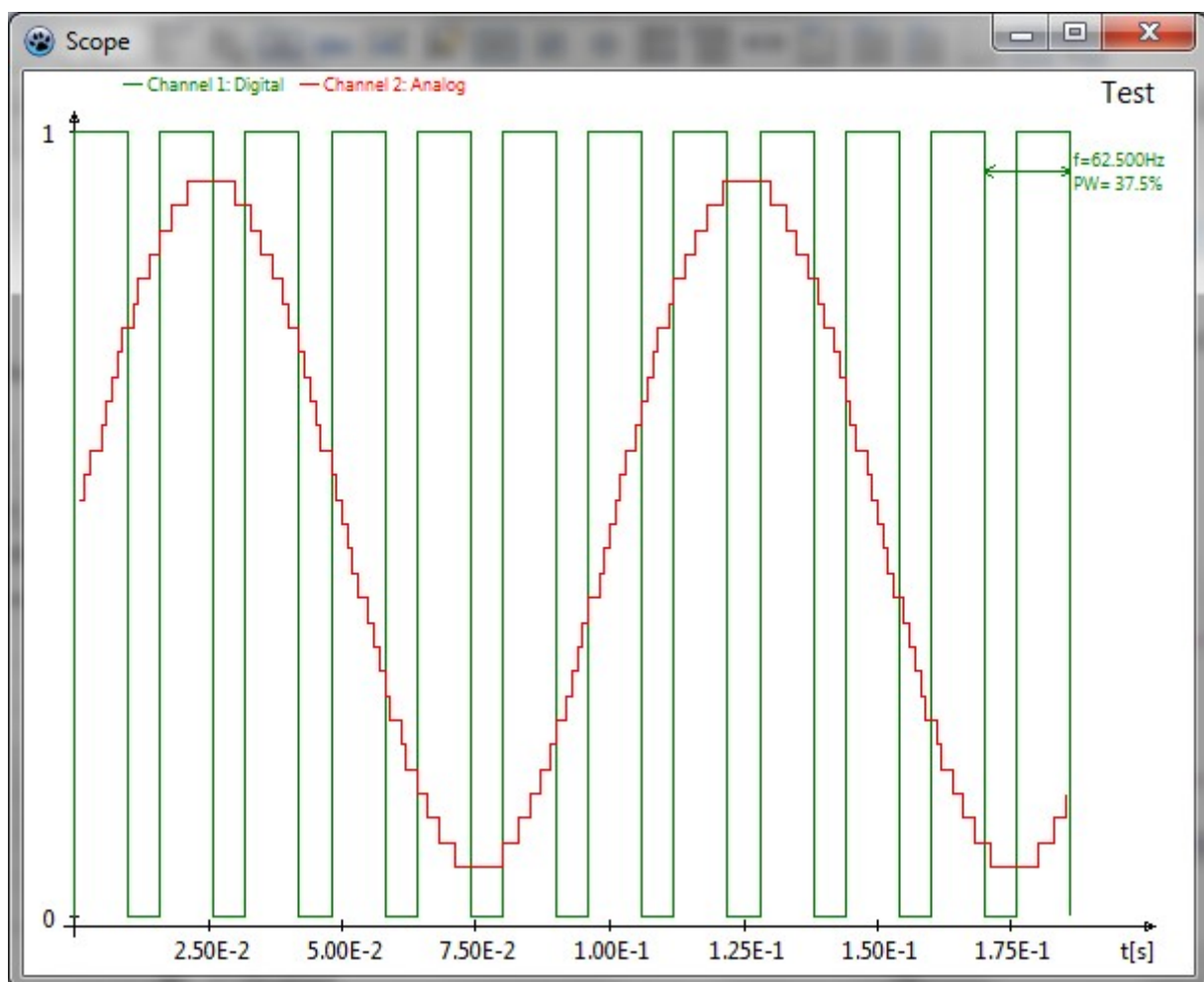


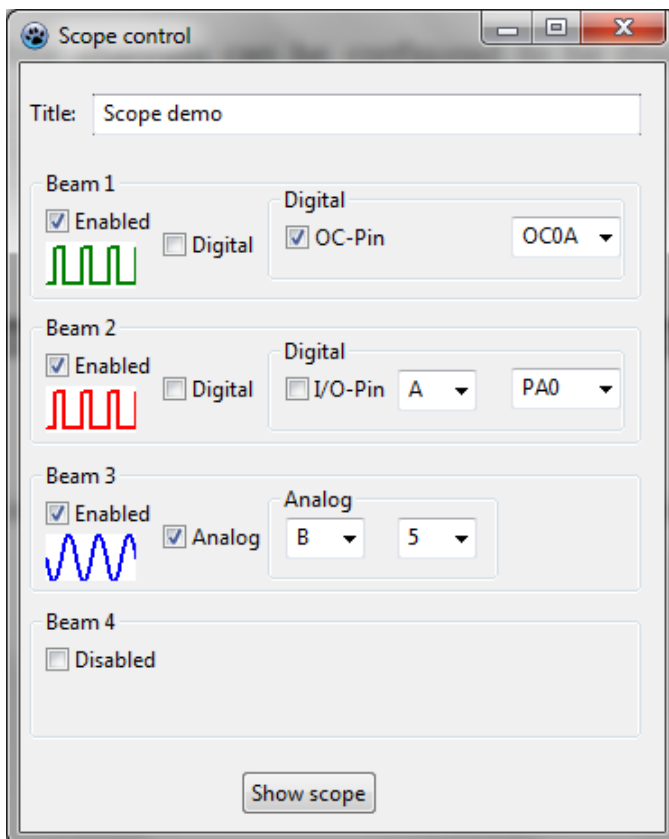
The example here shows the AD conversion (ADLAR set, MSB) results with input voltages of 0, 1, 2, 3 and 4 V at an operating voltage of 5 V, stored in SRAM.

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

6.7 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).

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.

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. ATxmega devices are implemented but no alternative pin functions are assigned.
5. Older AVR types, such as the AT90S, are implemented, but are not yet systematically tested for correct execution.

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 1.6

The following changes were introduced in this version.

9.1 Changes to the integrated assembler gaviasm

The internal assembler gaviasm was upgraded from version 4.2 to 4.3.

1. When a parameter is empty (e. g. with ,,) gaviasm reported a range check error. This has been corrected.
2. All internal number handling was changed from 32-bit integer (LongInt) to 64-bit integer (Int64). This allows to handle larger numbers. The bytes of those numbers are accessible with the functions BYTE1 to BYTE8.
3. The whole processing of .IF-Directives has been completely changed and was intensively tested.

9.2 Changes concerning the editor

1. During editing the program-internal check if breakpoints are still accessible lead to an incorrect failure to locate breakpoints and led to a series of false dialogues. The whole handling of breakpoints has been updated, see the description in chapter [5.3](#)).
2. If a breakpoint has been reached during simulation and a "<" is displayed on the listing's sidebar, this breakpoint was not saved in the project file. This is corrected.

9.3 Changes concerning the simulator

1. When simulating LDD/STD instructions with displacements from 16 upwards the target address calculation pointed to the wrong location. That is corrected.
2. The timer/counter overflow interrupt in an ATtiny12 is now simulated in a correct way.

3. Unintended access to SRAM locations beyond the simulated address space lead to a range check error.
4. The "Skip" entry in the simulation window has been added to jump over the next instruction.

9.4 Changes of the handbook

1. Adjusting the Lazarus compiler to other target operating systems and CPU families has been added in chapter [2.2](#).
2. The handling of breakpoints in chapter [5.3](#) has been changed.

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 I might have forgotten one or two of these. If you want to make changes to the design use Lazarus and the source code files.

In the ATmegs 48/88/168/328 the number of available AD channels is incorrect for some sub-types (P, PA, PB) and for certain package types. Only the base types (w/o Px) are implemented correct.