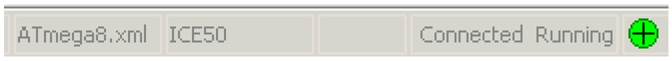
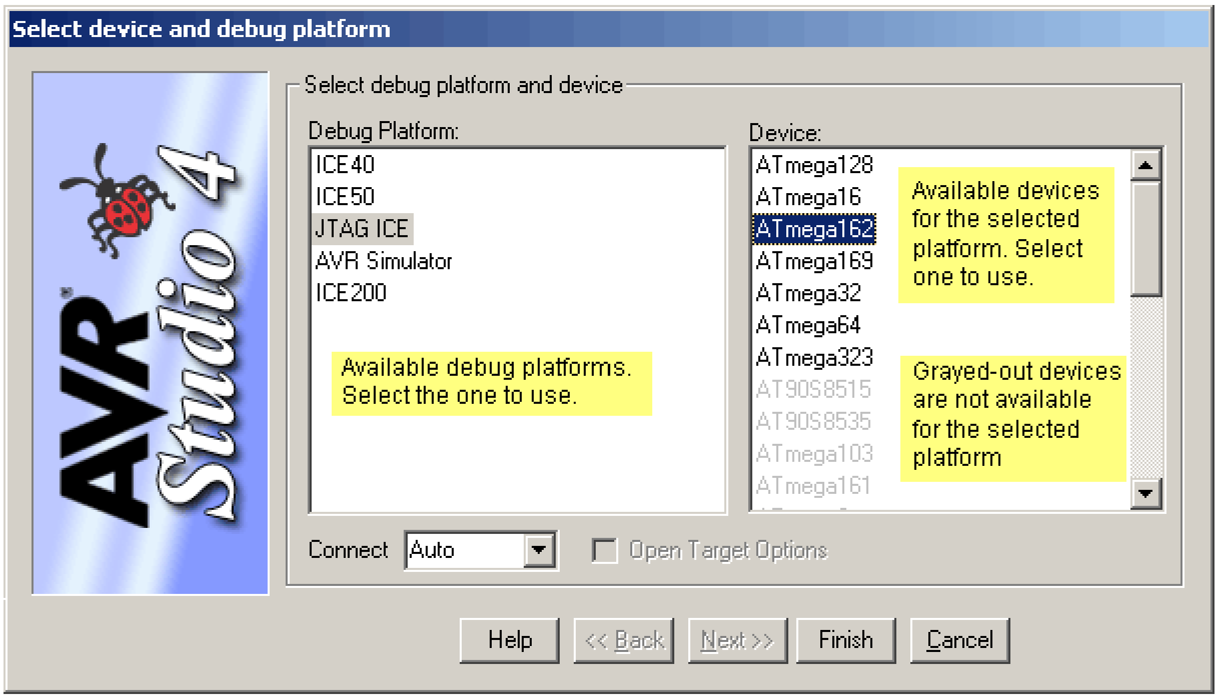
DESCRIPTION OF THE DEBUGGING FACILITIES

1) Status Bar

The Status bar always indicate whether debugging is targeted at the AVR In-Circuit Emulator or the built-in AVR Simulator. The name of the actual device and debug platform is output on the lower status bar:



Debug platform and device selection can be done by selecting *debug->Select debug platform and device.* All on-system debug platforms and devices are listed. When selecting a platform name, all supported devices for that platform are listed in black color, and unsupported are listed in gray. Click finish when your selection is done.


Select device and debug platform
AVR Studio 4
Select debug platform and device
Debug Platform:
ICE40
ICE50
JTAG ICE
AVR Simulator
ICE200
Available debug platforms. Select the one to use.
Device:
ATmega128
ATmega16
ATmega162
ATmega169
ATmega32
ATmega64
ATmega323
AT90S8515
AT90S8535
ATmega103
ATmega161
Available devices for the selected platform. Select one to use.
Grayed-out devices are not available for the selected platform
Connect
Auto
Open Target Options
Help
<< Back
Next >>
Finish
Cancel

2) Object files format and support

All debug sessions require that you load a debug object file that is supported by AVR Studio. Usually a debug file contains symbolic information which is not included in a "thinner" release file. The debug information enables AVR Studio to give extended possibilities when debugging, e.g. source file stepping and symbolic watches.

Below is a brief description of the object file formats that is supported by AVR studio

Object file format	**File**	**Description**
Extended Intel hex	.hex	This file is needed to program the device.This format is usually produced by most of the developer packages and is targeted for release testing. No additional debug information is included, and therefore not a recommended format for debugging. The file contains only program memory data. Source file stepping and symbolic watches are not available.
EEPROM data	.eep	Holds the contenet that should be written to the EEPROM.
Text file	.lst	Text file. The Program with all its addresses, commands and error messages can be displayed with a simple text editor. It might be useful in some cases to debug errors.
AVR Assembler format	.obj	This file is needed to simulate the AVR. Not compatible with the programmer software.The AVR assembler output file format contains source file info for source stepping and is an Atmel internal format only. The .map file are automatically parsed to get some watch information.

All projects are saved with your selected name with the ending .aps. When the user wants to reopen a project, this can be done under the file menu and the recently used file list, or under the project menu, open project.

A project file is created, an *.asm file is available in the editor window and you are set for writing your first instructions.

If you develop for a specific device in mind you should include the *.def.inc file for the part. Each part has it's own *.inc file that defines all internal registers, bits and a lot of other stuff that makes it simpler for you to write code for the part. In addition the *.inc file sets the device directive for the assembler, letting the assembler know which part you are developing for.

Before debugging, make sure you have set up your compiler/assembler to generate a debug file like one of the formats above.

3) Debug Control

Debug Control is used for controlling the execution of a program. All debug controls are available through menus, shortcuts and the Debug toolbar.

Note! If source level information is available in the object file, all debug operations will continue to execute until the first source statement is reached. If no source line is encountered, the program will keep executing. To stop execution, switch to disassembly mode before issuing a Break command.

Start Debugging

This command start the debugging process, and all debug control commands will be available. Normally in debug mode no code editing can be done. It connects to the debug platform, load the object file and performs a reset.

Stop Debugging

This command stop the debugging process. It disconnect the debug platform, and edit mode will be available.

Reset (SHIFT+F5)

The Reset command performs a Reset of the execution target. If a program is executing when the command is issued, execution will be stopped. If the user is in source level mode, the program will, after the Reset is completed, execute until it reaches the first source statement. After the Reset is completed, all information in all windows are updated.

Run (F5)

The Run command in the Debug menu starts (or resumes) execution of the program. The program will be executed until it is stopped (user action) or a breakpoint is encountered. The Run command is only available when the execution is stopped. Successive runs advance to the following Breakpoints.

Break (CTRL-F5)

The Break command in the Debug menu stops the execution of the program. When the execution is stopped, all information in all windows are updated. The Break command is only available when a program is executing.

Single step, Trace Into (F11) Steps into the code

The Trace Into command in the Debug menu executes one instruction. When AVR Studio is in source mode, one source level instruction is executed, and when in disassembly level, one assembly level instruction is executed. When running a subroutine call, the Simulator jumps to the subroutine and runs it line by line.After the Trace Into is completed, all information in all windows are updated.

Step Over (F10) Steps over routine calls

The Step Over command in the Debug menu executes one instruction. If the instruction contains a function call/subroutine call, it will treat it as a single instruction and not jump to the routine instructions. If a user breakpoint is encountered during Step Over, execution is halted. After the Step Over is completed, all information in all windows are updated.

Step Out (SHIFT+F11) Steps out of routine calls

The Step Out command in the Debug menu executes until the current function has completed. Temporarily puts the simulation into run mode for the remainder of the routine and will pause at the next instruction after the routine call.If a user breakpoint is encountered during Step Over, execution is halted. If a Step Out command is issued when the program is on the top level, the program will continue executing until it reaches a breakpoint or it is stopped by the user. After the Step Out command is completed, all information in all windows are updated.

Run To Cursor (F7)

The Run to Cursor command in the Debug menu executes until the program has reached the instruction indicated by the cursor in the Source window. If a user breakpoint is encountered during a Run to Cursor command, execution is not halted. If the instruction indicated by the cursor is never reached, the program executes until it is stopped by the user. After the Run to Cursor command is completed, all information in all windows are updated. As the Run To Cursor instruction is dependent on a cursor position, it is only available when the Source Window is active.

Auto Step

The Auto Trace Into command in the Debug menu repeatedly executes Trace Into instructions. When AVR Studio is in source mode, one source level instruction is executed, and when in disassembly level, one assembly level instruction is executed. After each Trace Into is completed, all information in all windows are updated. Auto Trace Into run until it is stopped (user action) or a breakpoint is encountered (optional).

Set next statement

With this command (mouse right click) you can set the yellow marker anywhere in the code. Point to the actual program source and chose the command. The next debug run command will execute from this point.

Show next statement

Set yellow marker at the actual program counter location. Focus the window.

4) Breakpoint control

Advanced debugging is achieved by the use of Breakpoints. Breakpoints are user defined locations in the code where the program halts when running at full speed.
The user can set an unlimited number of code breakpoints. The breakpoints are remembered between sessions.
When a breakpoint is set on a location, the breakpoint is indicated by a mark on the left hand bar in the Source window. By clicking the right mouse button on a source line with a breakpoint, the user can toggle the breakpoint on and off, or disable it.
Breakpoints can also be controlled from the toolbars and the breakpoint list in the output view. Breakpoints must be listed in the output window, before the build and run procedure.

Toggle breakpoint (F9)

The Toggle breakpoint command toggles the breakpoint status for the instruction where the cursor is placed. Note that this function is only available when the source window or disassembly window is the active view.

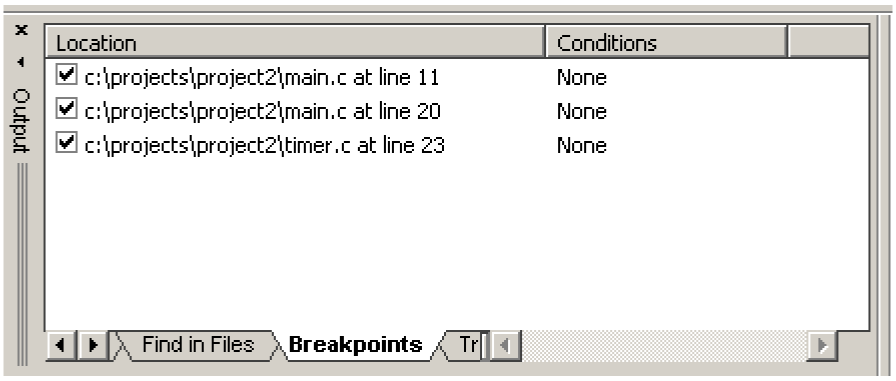
Clear all break points

This function clears all set breakpoints, including breakpoints which have been disabled.

Disable/enable breakpoints

Breakpoints can be disabled. This means they are not used, but a white circle shows where they were. This can be useful for toggling breaking on/off without losing the position of the breakpoints.

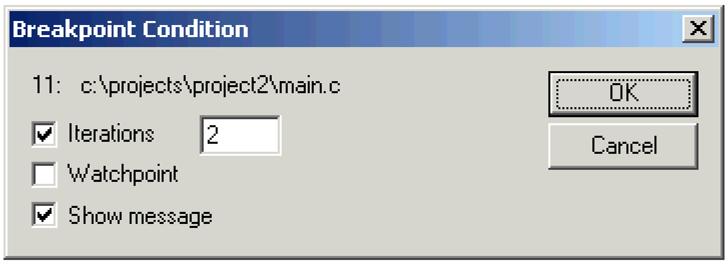
In the output view, by toggling the check box next to a listed breakpoint, the user can enable or disable it.



Location	Conditions
✓ c:\projects\project2\main.c at line 11	None
✓ c:\projects\project2\main.c at line 20	None
✓ c:\projects\project2\timer.c at line 23	None

5) Setting conditions from the breakpoint list in the output view

This list contains all user breakpoints. By clicking the right mouse button, the user can set some conditions for the break point.



Breakpoint Condition

11: c:\projects\project2\main.c

✓ Iterations 2

Watchpoint

✓ Show message

Iterations
Checking this box means that break should only occur after a number of passings of the breakpoint. Note that if this feature is used, emulation will not be real time. The execution is stopped when hitting the breakpoint, but execution is resumed automatically until the number of iterations are passed. The views in AVR Studio are not updated during the temporary break.

Watch point
Checking this box means that execution is halted and all views are updated when the breakpoint is passed. When the views have been updated, the program execution resumes automatically. This condition can be combined with the *iterations* condition above.

Show Message
Checking this box will output a message in the in the output window when the condition are met. This can be informative if any of the conditions above are used.

6) Reloading

Note that reloading a modified object file may cause problems in restoring the breakpoints automatically.

When a object file is loaded and it is changed, currently saved program breakpoints are invalid. You can decide what AVR Studio should do then:

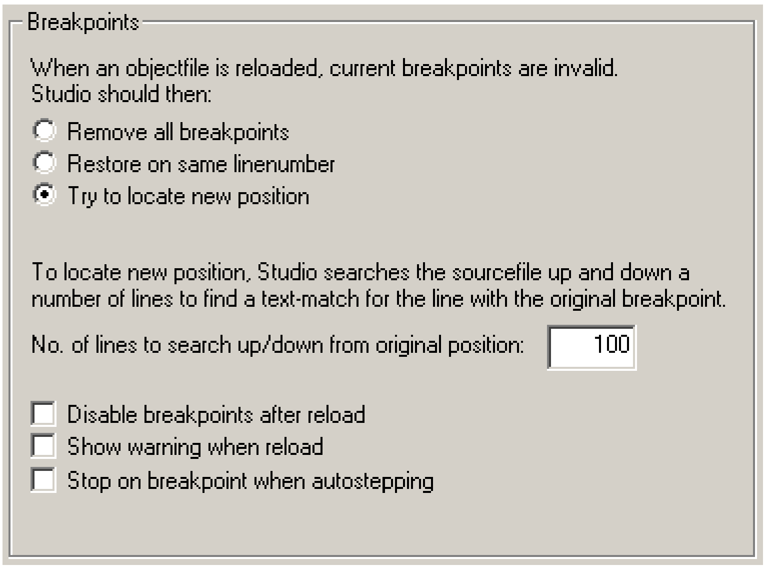
Remove all breakpoints
This will remove all breakpoints if the object file is changed.

Restore on same line number
This will set the breakpoint on the same line number.

Try to locate new position
AVR Studio save the line number, address and current text for the actual line for all breakpoints after each session. With this option set, AVR Studio will search the lines after the actual text and then set the breakpoint. You can also set the number of lines to search.

Disable breakpoints after reload
This will disable all breakpoints after you reload the project.

Show warning when reload
This will give you a warning if the object file is changed and breakpoints are saved.


Breakpoints

When an objectfile is reloaded, current breakpoints are invalid.
Studio should then:

( ) Remove all breakpoints
( ) Restore on same linenumber
(●) Try to locate new position

To locate new position, Studio searches the sourcefile up and down a number of lines to find a text-match for the line with the original breakpoint.

No. of lines to search up/down from original position: 100

☐ Disable breakpoints after reload
☐ Show warning when reload
☐ Stop on breakpoint when autostepping

7) Debugging strategies και συχνά λάθη

- Διαχωρισμός του κώδικα σε μικρού μεγέθους modules.Ετσι διευκολύνεται το στάδιο της αποσφαλμάτωσης.
- Σχόλια σε ομάδες εντολών για την περιγραφή της διαδικασίας που υλοποιείται.
- Συμβατότητα του επιλεχθέντος μοντέλου AVR, με τις επιλογές σε εντολές, τρόπους προσπέλασης, κλπ. Χρήση των συμβατών I/O ports με το συγκεκριμένο μοντέλο AVR. Χρησιμοποιείστε θέσεις υπάρχουσας (εγκατεστημένης για το συγκεκριμένο μοντέλο AVR) SRAM .
- Για δεικτοδότηση χρησιμοποιούνται οι X, Y Z και όχι ο SP.
- Μην ξεχνάτε την ψευδοεντολή include “8515def.inc”.
- Μη ξεχνάτε τις αρχικές συνθήκες (ιδιαίτερα στο περιβάλλον του simulator) για τους καταχωρητές R0- R31 και SP, καθώς και την εισαγωγή της εντολής sei.
- Input από PINx και όχι από PORTx.
- Το πρόγραμμα σας να τερματίζει με κάποιο τρόπο, π.χ. Loop: rjmp loop.
- Οταν χρησιμοποιείτε τη στοίβα, θα πρέπει να δίνετε αρχικές συνθήκες στον SP . Συνήθως η αρχή της στοίβας ορίζεται στην ramend (γνωστή από την include).
- Προσοχή στη χρήση αρνητικής λογικής για τον καθορισμό δυαδικών μεταβλητών.
- Προσοχή στη χρήση των ψευδοεντολών για την ονομασία καταχωρητών.

- **Ξεκινήστε με Open Project, για τον καθορισμό περιβάλλοντος και επεξεργαστή.**
- **Αποθηκεύστε τα αρχεία σας στο συγκεκριμένο χώρο users. Αρχεία από την Flash δεν διαβάζονται. Κατά τον προγραμματισμό του 8515, προσοχή στην χρήση του επιθυμητού αρχείου (αυτού που πρόκειται να εκτελέσετε) με το κατάλληλο pathname.**
- Οι κάρτες **STK500** διαθέτουν διαφορετικό επεξεργαστή. Ελέγξτε του τύπο του, για να τον χρησιμοποιήσετε στην εντολή include και στο παράθυρο προγραμματισμού της STK500.
- Μετά από κάθε αλλαγή στον κώδικα, εκτελέστε Build and run και στην συνέχεια προγραμματίστε την κάρτα STK500.
- Στο στάδιο της αποσφαλμάτωσης. βρόχοι καθυστέρησης καλύτερα να ορίζονται σαν σχόλια (comment out).
- Προσεκτική επιλογή των Breakpoints.
- Προσεκτική παρακολούθηση των I/O window και Memory windows.
- Η χρήση του watch view είναι χρήσιμη για την παρακολούθηση των τιμών των μεταβλητών που ορίζονται με κωδικό όνομα.
- **Οταν το πρόγραμμα εκτελείται στον target AVR στην STK500, για να βεβαιωθείτε οτι το πρόγραμμα σας διέρχεται από συγκεκριμένη περιοχή χρησιμοποιείτε κατάλληλες ενδείξεις στα LEDs και προσωρινές στάσεις στην εκτέλεση του προγράμματος με αέναο βρόχο ή με πίεση κατάλληλου πλήκτρου.**
- **Η ορθή λειτουργία του προγράμματος στο περιβάλλον του simulator, δεν σας εξασφαλίζει και ορθή λειτουργία στην κάρτα STK500. Ιδιαίτερη προσοχή στη χρήση διακοπτών και LEDs (αρνητική λογική).**