

Δίνεται σύντομη περιγραφή για :

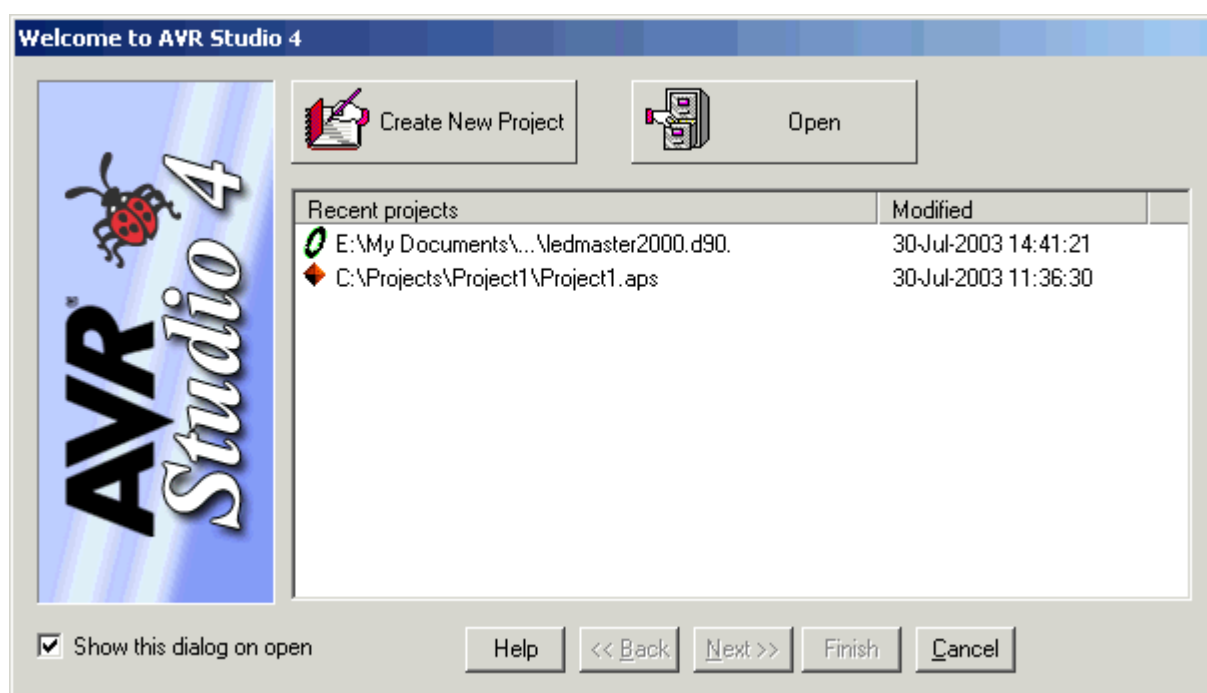
- 1) **Startup wizard**
- 2) **ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ PROJECT**
- 3) **SELECT PLATFORM AND DEVICE**
- 4) **ΤΑ ΔΙΑΦΟΡΑ ΕΡΓΑΛΕΙΑ** (κυρίως τα διάφορα παράθυρα, ο assembler και ο simulator. Ο Debugger περιγράφεται σε ξεχωριστό αρχείο).
- 5) **The STK500 development card** (επιπλέον πληροφορία στο αντίστοιχο doc)
- 6) **PROGRAMMING**

ΣΥΝΤΟΜΗ ΠΕΡΙΓΡΑΦΗ

AVR Studio is an Integrated Development Environment (IDE) for writing and debugging AVR applications in Windows 9x/Me/NT/2000/XP environments. AVR Studio provides a **project management tool, source file editor, chip simulator and In-circuit emulator interface** for the AVR 8-bit RISC family of microcontrollers. In addition, AVR Studio supports the STK500 development board, which allows programming of all AVR devices, and the new JTAG on-chip emulator.

1) Startup wizard

The startup wizard is displayed every time you start AVR Studio 4. From within this dialog you can quickly reopen the latest used projects, change debug platform/device setup or create a new project. Just double-click on the wanted project and it will automatically open and restore to its last settings.



Other functions are described in more detail below.

Show this dialog on open

Uncheck this checkbox if you don't want to display this dialog on startup.

Create new project

If you want to create a new project, use this function.

Open

If you want to load an existing project or a single debug object file, press this button.

Next Step

This button is highlighted when a project is selected. Press next to select platform and device to eventually change the debug platform or device setup for the selected project. This is described in (3).

Load

Load the selected project.

2) ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ PROJECT

Καθορισμός ονομασίας και pathname.

Select *File->new project* from the menu, and the dialog below will appear. The startup dialog will also have this option. Project name and project type must at least be selected before continuing.

Project type

All available project types are listed in the project type list box. Project type must be selected before next step or finish can be pressed.

Project name and initial file

Type your project name. Default the initial file will have the same name + .asm and will be created, but this can be changed. A folder with the project name can be created, but this is not default selected.

Next Step

If project name and project type are ok, press next to select platform and device to simulate/emulate. This is described in (3). You can also finish now, but then the debug platform and device must be selected when a debug session is started.

Create new Project

Project Type: Atmel AVR Assembler

Project Name: Project1

☒ Create initial File ☒ Create Folder

Initial File: Project1 .asm

Location: C:\Projects\ ...

3) SELECT 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

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

4) ΤΑ ΔΙΑΦΟΡΑ ΕΡΓΑΛΕΙΑ

AVR Studio 4 consists of many views and sub-modules. Each of them supports parts of the work you try to undertake. The major views are described in the following.

4.1) Project Manager

All code creation within AVR Studio is done as programming projects. All code projects has a project file that maintains information about the project, which files are included, assembler set-up, customized views and so forth. The project file ensures that the project stays the same every time you come back to it, and that everything is set up properly and will assembler / compile.

4.2) Editor

This is the place you do the actual work of coding. AVR Studio 4 uses the Stingray editor from Bsquare corporation. It is a full fledged programming editor with all the functionality that you are used to, including syntax color coding (that can be altered and extended by the user). The editor window is also used when you debug your code, break and tracepoints can be set at the left margin of the view. The usage is straight forward.

4.3) DESCRIPTION OF THE VIEWS

The following views are available:

Output tabs

Build window – Message window – Find in files function – Breakpoints

Workspace tabs

Project view

Register – Processor core register – Stack – I/O register

Info view

Interrupt vector – AVR package – I/O register

Watch view

Memory view

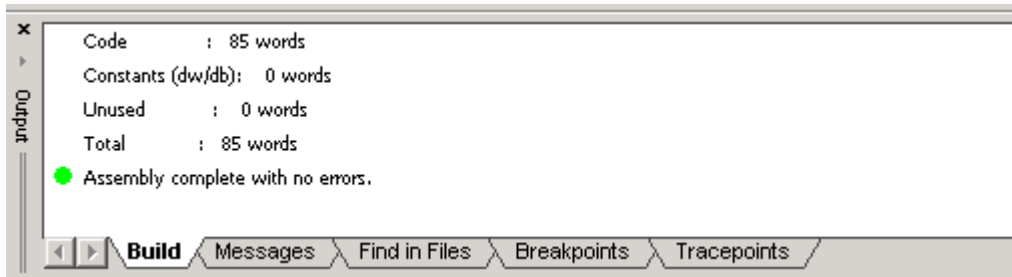
SRAM – Flash Memory – Register – I/O register – EEPROM

Register window

Dissassembler window

4.3.1) The output tabs

Is a collection of several views integrated into one tabbed frame. The tabs on the frame select the view you want.



You can select from:

- **The Build view.** Output from the compiler / assembler is routed to this window. The result of the compilation / assembly can be read here.
When compiling or building projects output messages and warnings are printed in this view. Double click on an error or warning and a blue marker will point you to the source code location. The key F4 can also be used, and will go to the next error.
- **The Messages view.** The Messages window is the common window for all modules of the software to present messages to the user. Messages are color coded. Most messages are plain messages of no significant priority. They have no color. Warnings signalling potential problems are yellow. Errors are red. All messages can be time stamped (right click your mouse over the window and select the time stamp option). There is even a filter function enabling you to turn messages on and off.
- **Find in files view.** The AVR Studio 4 has an advanced "find in files" function. The result of the search is routed to this window.
- **Breakpoints.** Lists all active breakpoints in all modules. Breakpoints can be enabled, disabled and removed in the view. More details in section 4 of the debugging document.

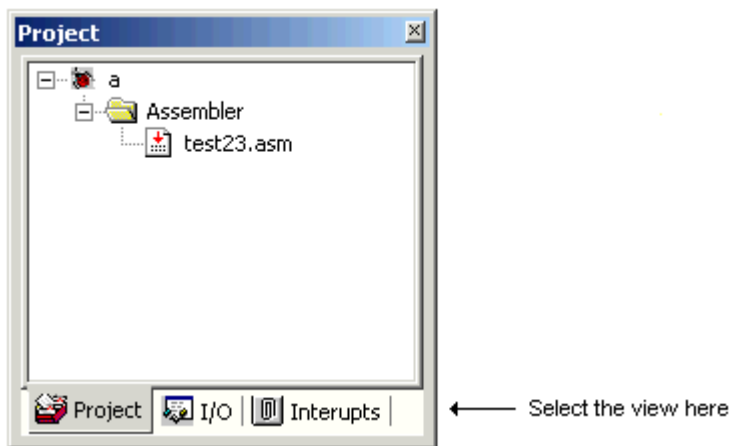
4.3.2) The workspace tabs

AVR Studio 4 workspace view consists of a number of views that are intended to help the developer to debug the code he has written in a systematic way:

4.3.2.1) The Project View

The Project view lists the files that are included in your code project, if you have created a code project. If you have opened an object file for debugging, the project

view shows the name of the currently loaded object file as well as the source files that the object file refers to.



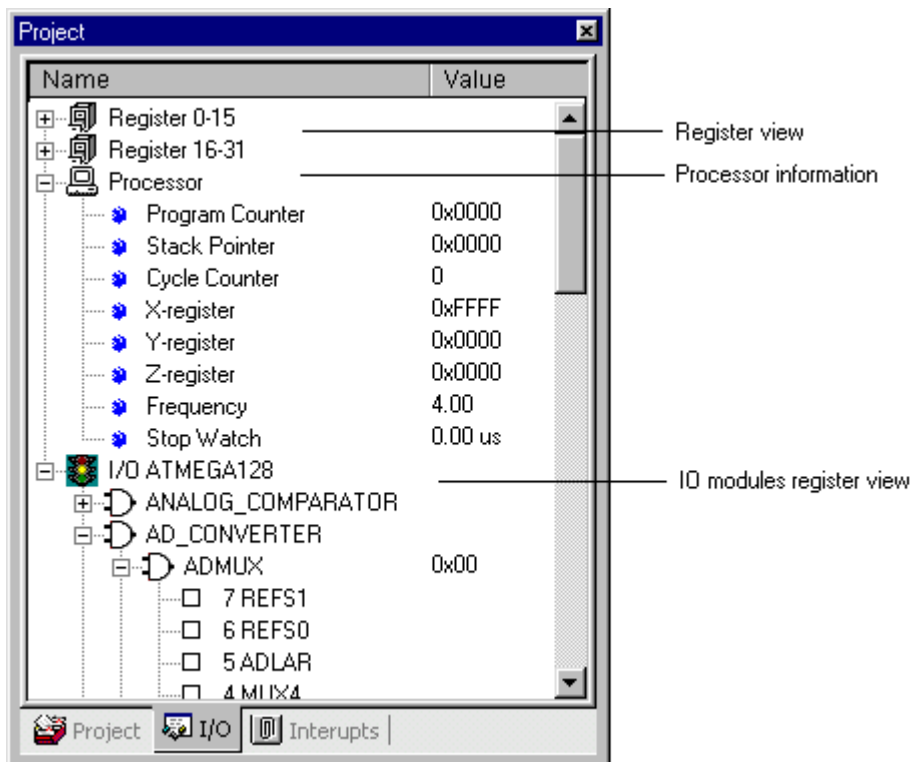
4.3.2.2) The I/O View

When studio opens a saved project the state of the tree is restored, branches left open are displayed opened.

The view has 4 fields of information for the I/O registers, name, value, bits and address.

The width of the fields can be adjusted by sliding the separators in the header back and forth. These settings are preserved between projects.

The information view can be used to get valuable information about I/O registers and bits. The information window is floating and not dockable. Just point on the actual I/O name, and the information will pop up. The information window is activated from the right menu.



The I/O view is divided into:

- **The register file section.** All AVR controllers have a set of 32 general purpose registers that freely can be used by the programmer or the compiler. Registers with values that is updated with the latest debug command are colored red. Double click on value to edit. There is also another register window available. If the value has changed since the last break, the register appears color coded.
- **The processor core register section.** As for the register view, the processor registers are updated when the execution breaks. Here you can view the program counter, the stack counter and so forth. This part of the view show the following information

Program counter (PC)

Program execution pointer. Location is marked with a yellow marker in the source window.

Cycle counter

Number of cycles executed after reset.

Stack pointer

Stack pointer points to a SRAM/Data location. The 16 bit value of I/O register SPH and SPL.

X Register

R26 and R27 16 bit value.

Y Register

R28 and R29 16 bit value.

Z Register

R30 and R31 16 bit value.

Frequency

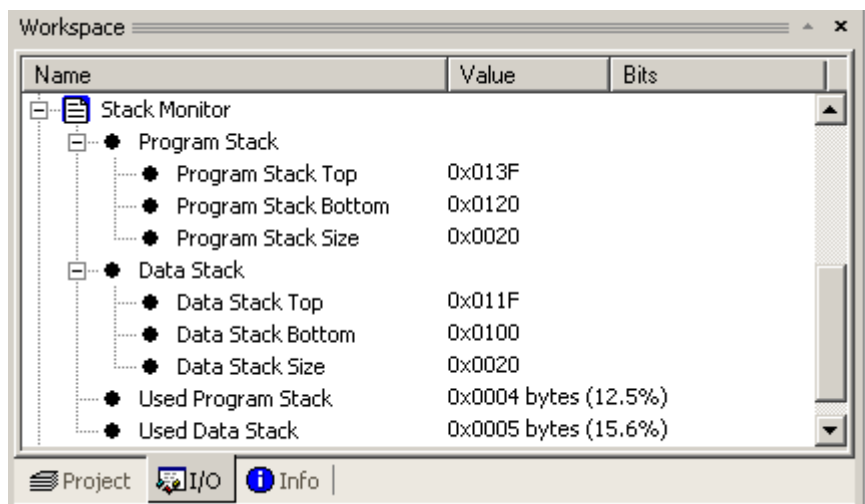
The frequency in megahertz read from the debug platform. Usually selected by the user.

Stopwatch

The stopwatch is showing time used. This is a product of the cycle counter and the frequency. Right click on the value to reset the stopwatch and toggle between microseconds and milliseconds.

- **The stack monitor section**

The ICE50 (not the simulator) has built-in stack monitors for both the stack pointer (SP) used as a program stack and the Y-register used as a data stack. The purpose of the stack monitor is first of all to notice the user of any over- or underflow of the program or data stack, which will corrupt the program flow or operation of the application code. An additional feature is that the maximum used stack size is logged for both the program stack and the data stack. This gives the user the possibility to optimize the memory space used for program and data stack.



There is one branch for the program stack, and one branch for the data stack. Each of the two branches shows the start address (top of stack), stop address (bottom of stack) and size of the respective stack. Below the two branches is a separate field for the maximum used stack size for both the program stack and the data stack. The maximum percentage used of the available stack is also shown beside these values.

- **The I/O register section.** Each different AVR device is defined through the

peripherals the device supports. An ATmega128 has a completely different set of peripherals compared to an AT90S8515, but the core and the instruction sets of the two chips are quite equal. All peripherals are controlled through 8 or 16 bit registers that can be read and/or written. The I/O view is configured for the part you have selected and shows all registers and bits logically grouped. Bits and registers can be both read and written when the emulation is in break mode. This view gives you total control over the device subjected to debugging. Registers with values that is updated with the latest debug command are colored red. Double click on value to edit.

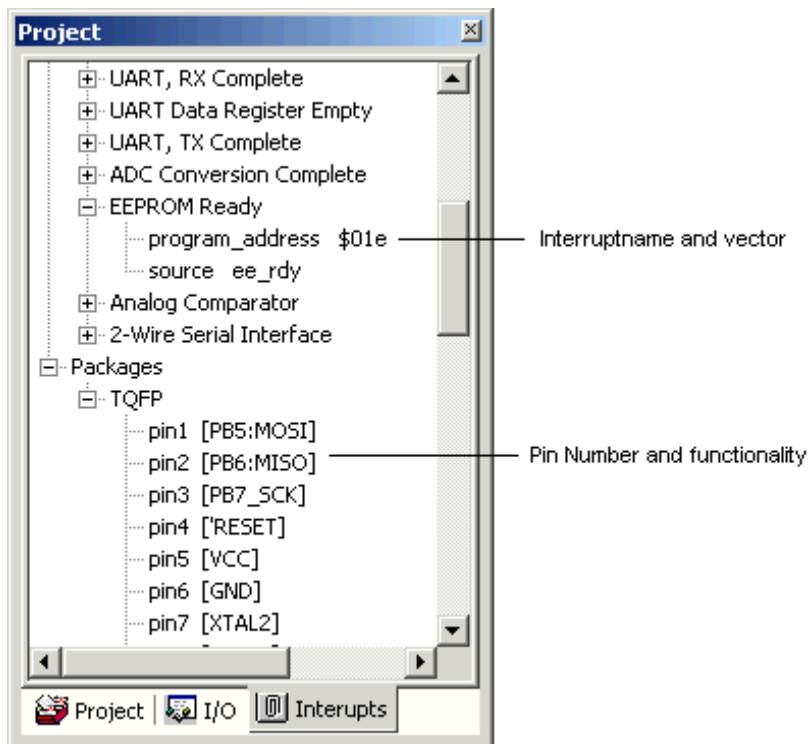
4.3.2.3) The Info View

This view is static and show all interrupts, pin configuration and available IO addresses for the selected device.



Use the information window which is selectable from the right menu to get more detailed information about interrupts, pins or IO addresses. Just point with the mouse to the actual text and information will be available in the information window.

The info view is divided into:

- **The interrupt vector section.** The interrupt vectors for a device usually reflects the peripherals. Thus they are different for each AVR device. This view lists all interrupts and their corresponding interrupt vector addresses. Useful stuff if you are to program interrupt service routines (and most microcontroller programmers are!)
- **The available packages section.** Lists the available packages of the device, and the corresponding pinout of each package. The tooltip help will contain information on the use of most pins. Again, this is a useful feature if you are debugging both HW and SW, you have the necessary pin information in view at all times.
- **Registers,** this is a complement to the I/O view. The I/O view displays all registers grouped logically into peripheral functions. The register view lists all registers from high to low address



4.3.3)The Watch View

Function	Toolbar	Menu	Shortcut
Toggle Watch window		View->watch	Alt-I
Quick watch			

Description

With the watch window you can view and edit all defined symbols when debugging. Functionality depends on the type of object file you have loaded. There are 4 different tabs (watch #1-#4) and all can contain different variables and settings. The settings and contents will automatically be saved when project are closed. It will restore to its last state when project are reloaded.

Watching many items at once can slow down debugging speed, especially on ICE's which uses the serial communication line. A good idea is to make the actual variables visible during any debug operation as only the visible items are updated.

Name, type, value and location are displayed for each item.

Name	Value	Type	Location
ms	43	int	R16:R17
MCUCR	0	volatile unsigned char	0x0035 [IO]
a	Not in scope		

Watch 1 Watch 2 Watch 3 Watch 4

Edit

Double click on an empty line to type a variable name. Variables can be dragged into the view from the source/editor window. Remove any variable by selecting it and select *remove variable* from the right click menu or pressing the DEL key. The INS key inserts a new entry. Use space to select an entry (name or value) for editing.

When editing values the prefix 0x or 0X indicates a hexadecimal value. Starting with 0 interpret as octal.

Quick watch

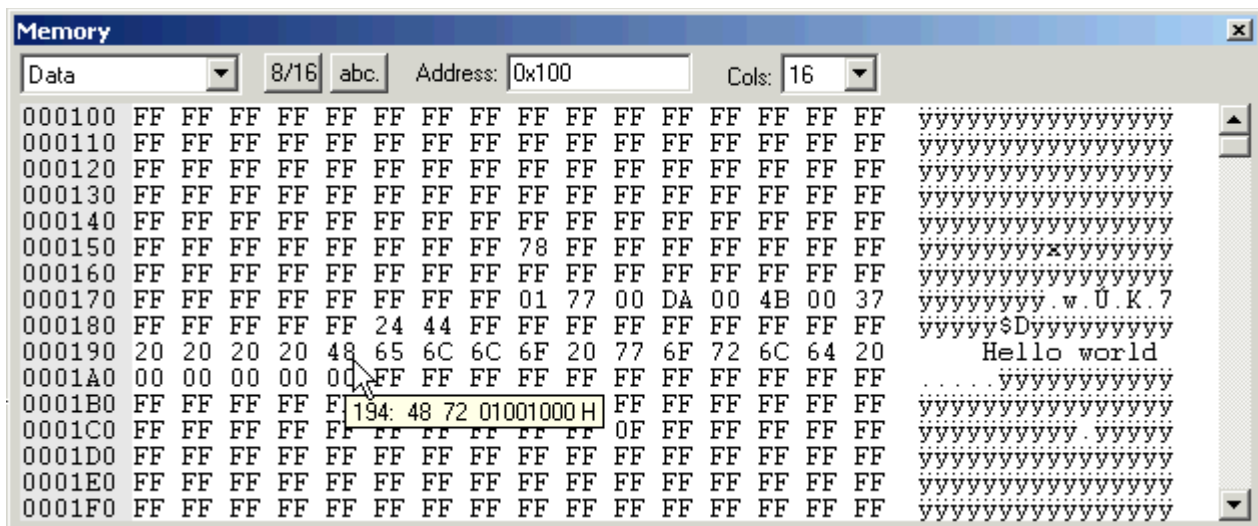
Highlight the variable you want to watch in the editor and select quickwatch. You can look at the variable and its contents and alternatively add it to the watch view.

Debugging high level languages

Debugging high level languages as C/C++ you need to watch variables. With tools like AVR Studio 4 this is an easy task, click on the variable you want to watch and drag-and-drop it into the watch window. If the variable you selected is a struct or an array a [+] symbol will occur in front of the variable, indicating that it can be expanded in the view. When the execution breaks, the variable automatically will be updated - if it is in SCOPE. Different languages define their scopes differently, to be in scope simple means that the variable actually exist in memory at the location where the program stopped it's execution.

4.3.4) The Memory View

A microcontroller like the AVR can do nothing without memory. The code executed lies in the program memory, the variables are located in SRAM and registers, the I/O registers are mapped into I/O memory area, and the EEPROM is yet another memory area. The Memory View is able to display all the different memory types associated with the AVR devices. And as for the watch and the I/O view, the area in display on the screen is automatically updated when an execution break occurs. All visible locations which are changed with the latest debug command (e.g. single step) will be marked by red.



Memory type

Select type of memory from the drop down list on the top left.

8/16 bit view

Press the "8/16" button to group the data into 16-bit words.

ASCII view

Press the "abc" button to toggle the ASCII pane on the right side of the window.

Address

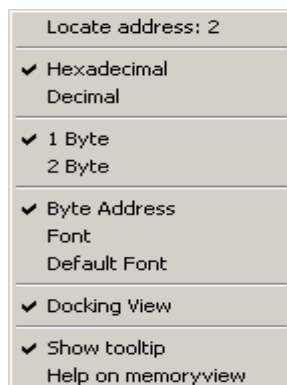
The address field shows the address of the first data location shown in the memory window. You can quickly jump to a new location by entering a new value in the address field.

Columns

You can select how many columns of data the memory window should display, from 2 to 32 columns. The Auto selection causes the contents to fit the window size.

Right mouse click menu

If you right-click inside the memory window, this menu is shown. Here you can control the behaviour of the memory window.



Editing

Change the value by clicking on the location and type a new value. This can also be done with the ASCII values in the right border (if shown). If you double-click on a

value, a dialog box opens, which lets you enter values in hexadecimal, decimal, octal or binary.

Navigating

Scroll up/down in the memory with the arrows, scrollbar or PageUp/PageDown, Home/End-buttons. Jump to a specific location by typing it in the address field. This is always treated as hex-value.

Docking

This is a docking view. Docking can be toggled ON/OFF with the right menu command Docking view

☐ **TIP:** You can display up to three memory windows at the same time. They are located on the View menu.

4.3.5) The Register window

All the registers R0-R31 can be viewed and edited in the register view. All visible locations that are changed with the latest debug command (e.g. single step) will be marked by red.

Edit Value

To edit just click at the wanted location and type the value. Navigate with the cursor keys.

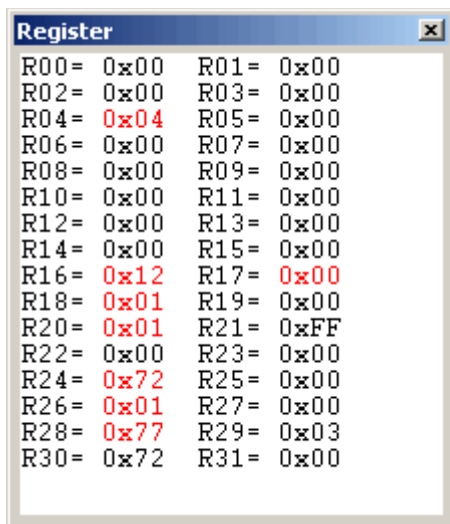
Alternatively double click on the actual value and an edit box will appear.

Edit label

Double click on any register/label, and you can enter any custom name.

Hexadecimal/decimal switch

Accessed by the right click menu, the displayed values can be toggled between hexadecimal and decimal output.



4.3.6) The Disassembler window

The disassembler window shows your program code disassembled. Program execution and AVR instructions can be followed in this view. When any supported high level language is used, the source window is automatically displayed and the disassembly window is OFF. When no high level language is available (e.g. using .hex files) the disassembly window will prompt at startup.

Program execution is controlled as in the source view.

Example of disassembly code cut from the disassembly window

```
@000000B9: main
---- c:\user\test programs\core io test.c (alternate location) -----
16:      MCUCR = 0;    //disable XRAM in order to use PORTA and PORTC as ordinary IO ports
+000000B9: 2700      CLR  R16          Exclusive OR
+000000BA: BF05      OUT  0x35,R16      Out to I/O location
17:      MCUCSR = 0x80;
+000000BB: E800      LDI  R16,0x80      Load immediate
+000000BC: BF04      OUT  0x34,R16      Out to I/O location
```

All lines starting with the + sign is the disassembled code. The first field is the actual program address. The next is the 2 byte value of the instruction. Then follows the instruction and a textual description.

The line starting with @ and address presents the high level label, in this case the c main label.

Configuring output

The window can be configured to toggle ON/OFF the following sections; program code, instruction help, source, source file header and labels.

4.4) ΑΝΑΠΤΥΞΗ ΚΩΔΙΚΑ

AVR Assembler

To get started is simple, when AVR Studio loads, select the AVR Assembler from the project dialog box, select a project name directory where the project shall reside, and click finish. A project file is created, an *.asm file is available in the editor window and you are set for writing your first instructions. Check out the online help. The AVR assembler has it's own book where all instructions and directives are explained. In addition, there is context sensitive help in the editor window, just write an instruction, place the cursor on top of the instruction and press F1, and you will get help on the syntax on the selected instruction.

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.

The part files are found in the \Program Files\Atmel\AVRTools\AVRAssembler\Appnotes folder on your computer. A include file for ATmega8 will typically be named "m8def.inc". You do not have to give a path with the *.inc file as long as it is found in the default directory.

Press F7 in order to compile. The result of the compilation will show in the previously described Build view in the output window frame.

4.5)SIMULATOR

This section describes how some of the functionality of AVR Studio's built in simulator varies from default behavior as described in the AVR Studio User's Guide.

The debugging facilities are given in the document “debugging”.

The simulator supports all existing new AVR devices, look at the tools and device support for an overview. It simulates not only the CPU, but nearly all the on-chip I/O modules and memory, as well as the I/O ports. Special care has been taken to ensure proper simulation of the device, and there are only small differences between simulated and actual behaviour. The simulator does not connect to outside hardware and has to be stimulated from pre-calculated stimuli files. But as the device is simulated entirely inside the PC memory, the user has extended visibility of all the on-chip functions.

Some I/O modules are not fully supported. See below for a link to an overview of the supported modules and known issues.

When AVR Studio is launched, the simulator will reset all program memories,

SRAM and EEprom to 0xFF. IO locations are set to 0x00.

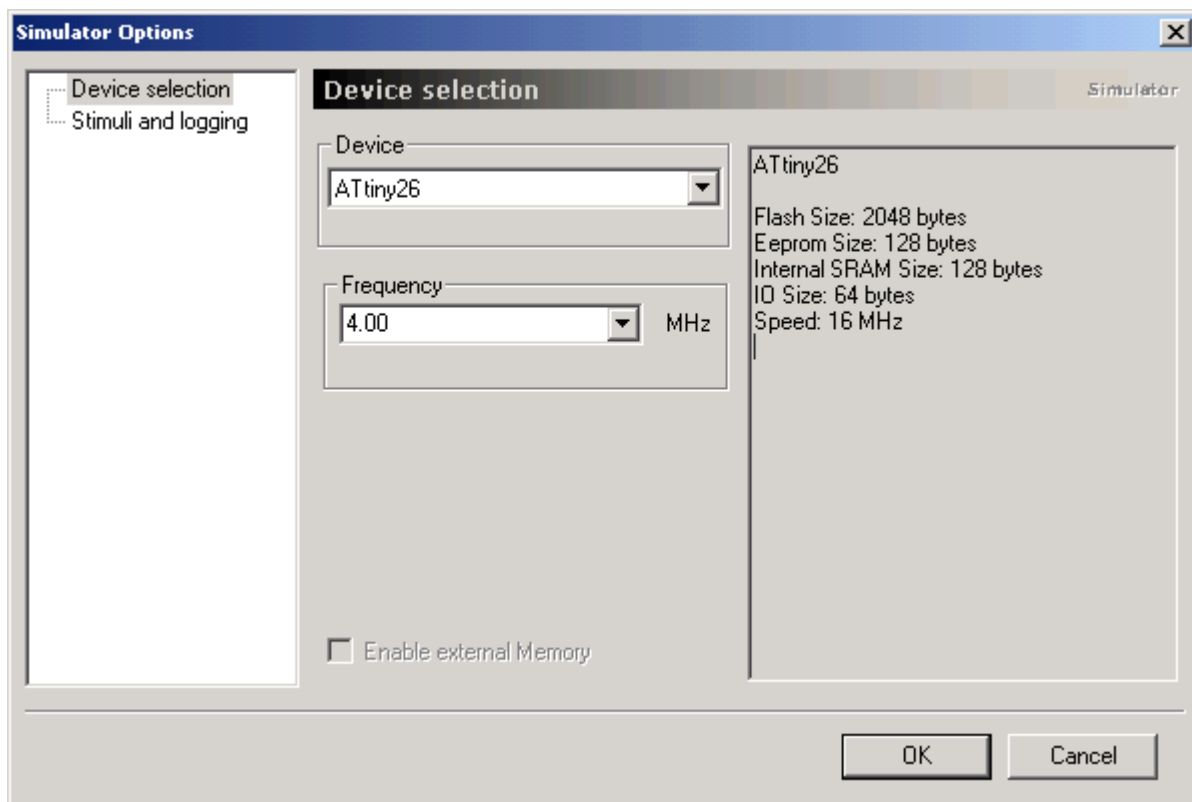
The full functionality of the memory/IO windows is supported by the simulator. But the following should be noted:

The memory contents can not be modified when the simulator is running. The simulator must break before new values can be written into the data area.

When simulating Target Devices supporting External Data Memories, the entire external memory is regular SRAM. If the object file contains initial data for the external memories, this memory will be written to external memory when the project is opened. A target Reset will NOT restore these initial memory settings.

4.5.1) Simulator options

With the simulator options the user can configure the simulator and some device specific parameters.



Device selection

The selected device is highlighted in a combo box. Some main parameters are output in the right column describing the part. Click on the combo box dropdown arrow and select the actual part. All simulator supported devices are listed in the box.

Frequency

Here the user can select any frequency in MHz. This parameter has no meaning regarding simulation speed, but is used as a product to calculate the stopwatch time.

Enable External memory

All devices that support external SRAM memory can toggle this switch on to get access to external SRAM. The internal SRAM are is expanded to cover the external area.

Boot loader

For all devices with a boot loader and the instruction spm, reset vector can be decided. If you enable boot reset, the combo box below will be enabled and reset vector can be selected. The actual reset vectors are read out from the part description .xml file.

4.5.2) Logging Ports

The user can log the activity on the output ports to a file or to the output window.

The user must select which of the Ports to log. If a Port is selected, the user must also select the file (or screen) to place the logged data on. It is the contents of the Port's PORT register which is placed on the file. Each line of the log file has the following format:

Cycle:Data

IMPORTANT: The Cycle field is in decimal format and the Data field is in hexadecimal format. Maximum 9 digits for the cycle value. If the contents of the port register is unchanged in a cycle, no output is produced.

The log files are deleted each time the program is reset. Logging must be activated manually each time a program is loaded into AVR Studio, settings are not saved between project sessions.

4.5.3) External Stimuli

The user can set values on any of the available ports from a file.

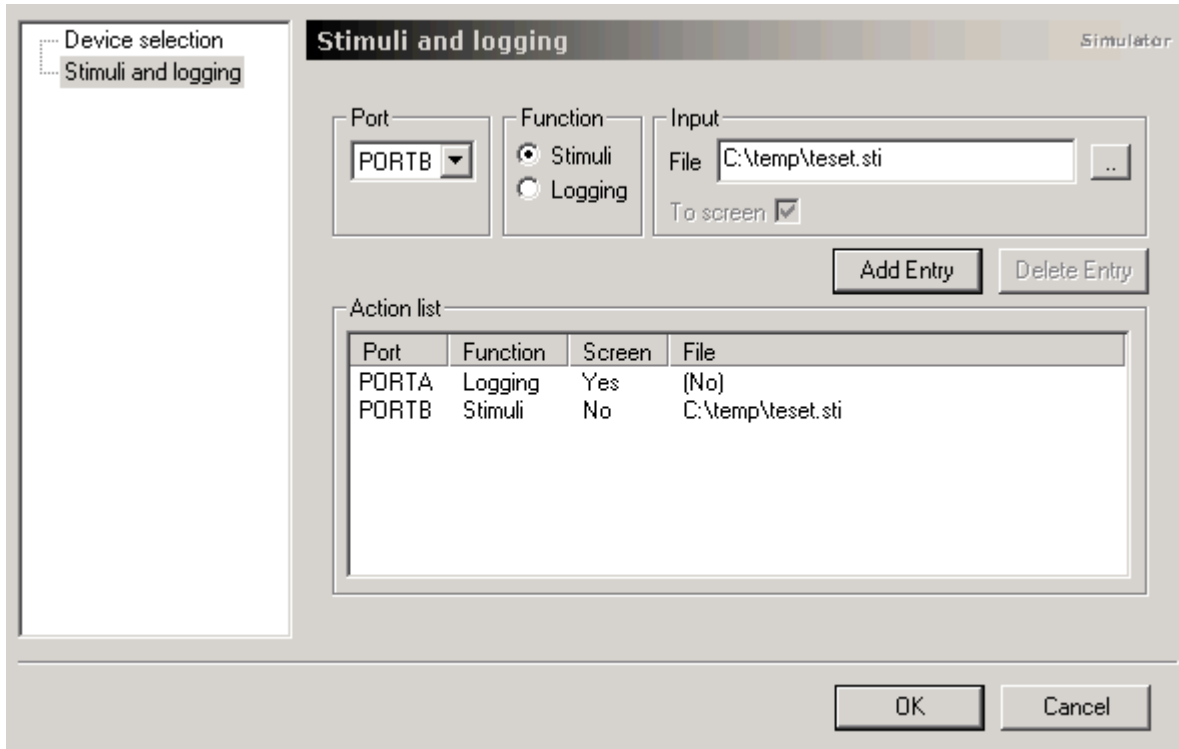
If a Port is selected, the user must inform where the stimuli file is placed. The values in the stimuli file are placed in the PIN register on the specified port on the specified cycle. The format of the stimuli file is the same as for the Port logging. Example:

```
000000000:00  
000000009:AB  
000000014:AC
```

Note that only pins set as inputs are affected.

The last line of the stimuli file must be like this:

```
999999999:FF
```



4.5.4) SIMULATOR MODULES

The following modules are supported :

4.5.4.1) Instruction Set

All instructions are simulated including spm and sleep. Sleep does only support IDLE mode.

4.5.4.2) Interrupts

All interrupts are supported and setup as described in the part description .xml file.

4.5.4.3) Ports

Ports are simulated as on the real device. This includes the 1.5 clock cycle debouncing delay found in the standard port logic hardware in actual AVR parts. When an I/O module take control over a pin, the value read from the PINx register is the value generated by that I/O module.

4.5.4.4) Timers/Counters 0/1/2/3

Timers/Counters are supported by the Simulator. The Timer/Counter interrupt vectors and the external counter(s) pin are set as described in the part description .xml file.

Asynchronous operation is not supported.

4.5.4.5) UART/USARTS

The UART/USART is supported by the Simulator. The UART/USART interrupt vectors and the Receive/Transmit pins are set up as described in the part description .xml file. Writing to the UART Data Register (UDR) will not initiate a data transfer. The Data Register must be written by the target application.

4.5.4.6) SPI

The SPI is supported by the Simulator. The SPI Data Register shows the SPI receive Register. Editing the SPI Data Register will not initiate a data transfer, even if the SPI is enabled in Master mode. The Data Register must be written by the target application.

4.5.4.7) External Interrupts, Pin change interrupts

All external and pin change interrupts are supported by the Simulator.

4.5.4.8) EEPROM

The EEPROM is supported by the Simulator, including WE and WEE interrupts. For simplicity, the write timeout has been set to 22 clock cycles, which is significantly shorter than the actual device.

4.5.4.9) ADC/AC/TWI/USI

These modules are not supported by the simulator at the moment, but the interrupts are initiated.

4.5.4.10) Watchdog Timer (WDT)

The Watchdog Timer is supported for the ATmega48/88/168 parts, and will be included with other parts in a future release.

5) The STK500 development card

- STK500 / 501 supports all AVR devices, it has LED's and switches so you can write programs for your device that reacts to real life events, and all programming modes of the device is supported. You can also use it as a test target for ICE50 or JTAGICE. STK501 is a mezzanine board that supports the ATmega128 TQFP package. It includes adapter for the JTAGICE.

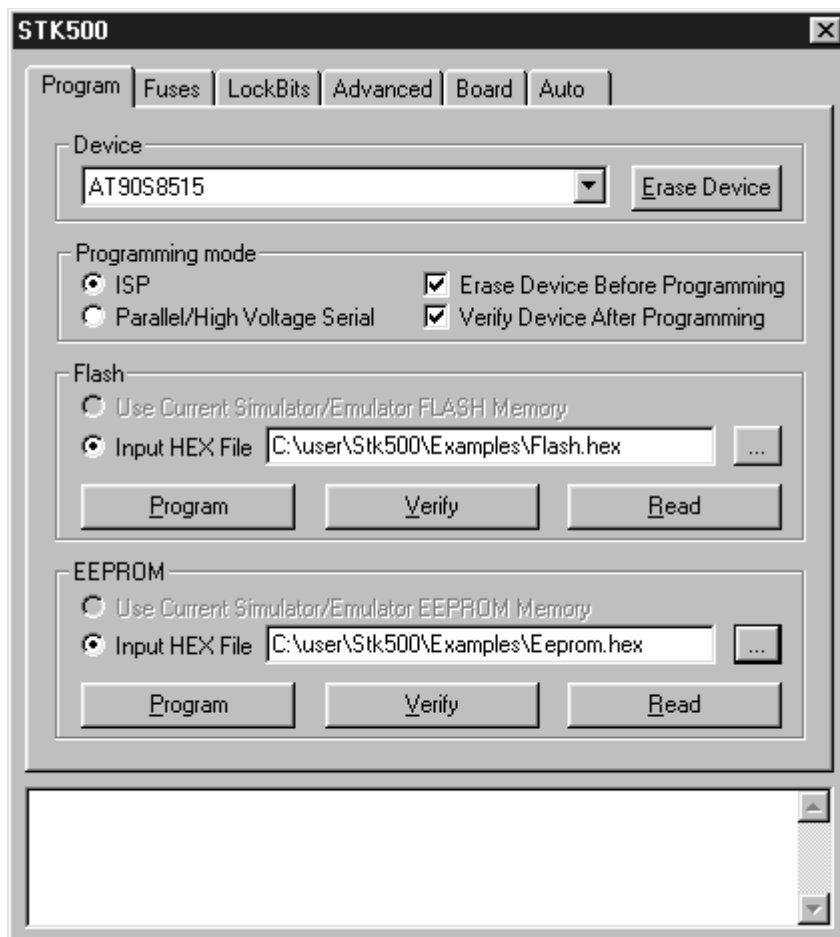
6) Programming the Target AVR Device

STK500 is controlled from AVR Studio, version 3.2 and higher.

To program a hex file into the target AVR device, select "STK500" from the "tools" menu in AVR Studio.

Select the AVR target device from the pull-down menu on the "Program" tab and locate the intel-hex file to download.


Press the "erase" button, followed by the "program" button. The Status LED will now turn yellow while the part is programmed, and when programming succeeds the LED will turn green. If programming fails, the LED will turn red after programming, see the ["Trouble-shooting guide" \(Help in STK500 user's guide\)](#).

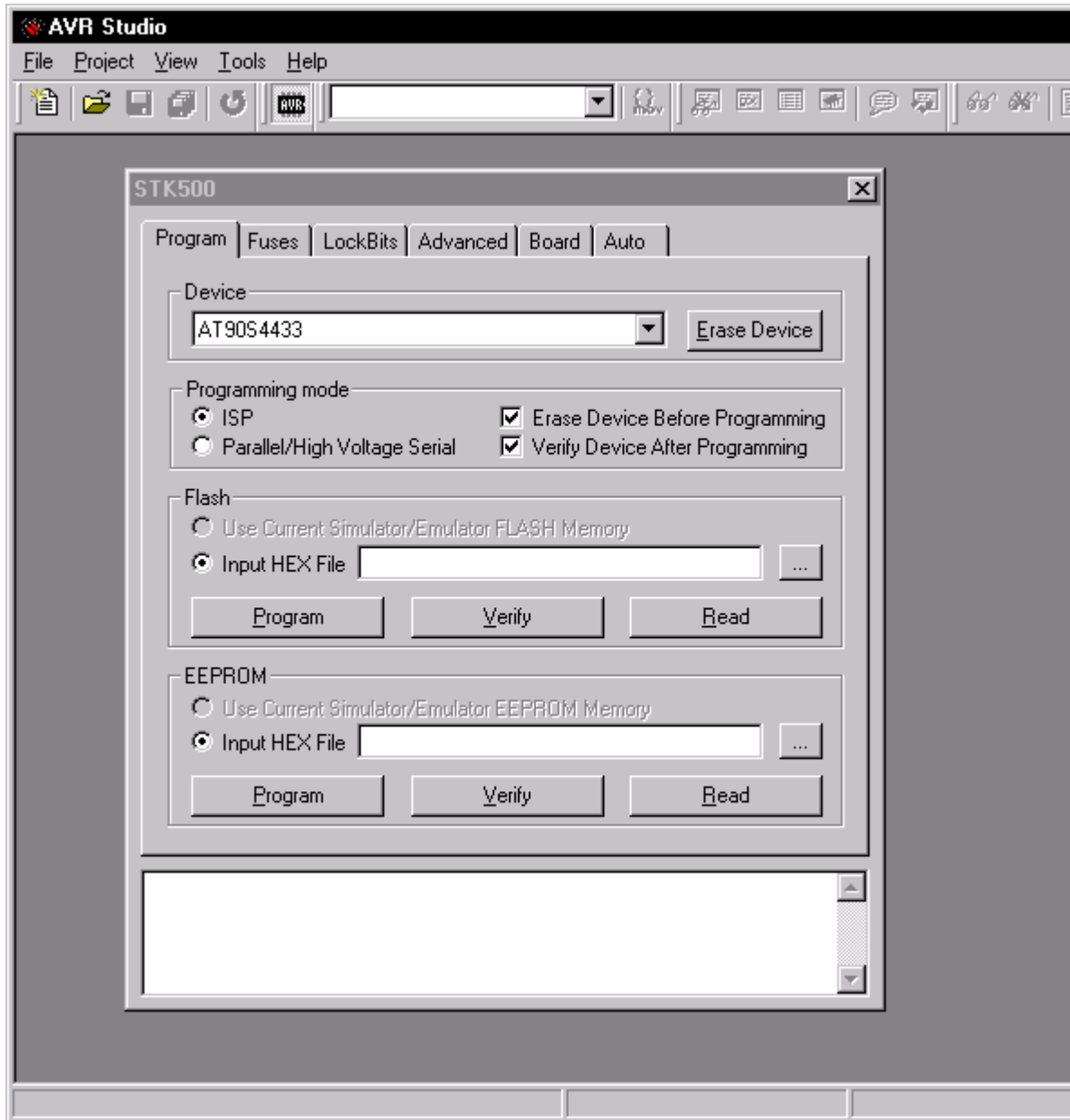


6.1) Using AVR Studio with STK500

In this section the supporting software for STK500 will be presented, and an in-depth description of the available programming options is given.

Starting STK500

Pressing the  button on the AVR Studio toolbar will start the STK500 user interface as shown in the figure below.



6.2) Program settings

The program settings are divided into 4 different sub groups.

Device


Device is selected by selecting the correct device from the pull-down menu. This group also includes a button that performs a chip-erase on the selected device, erasing both the FLASH and EEPROM memories.

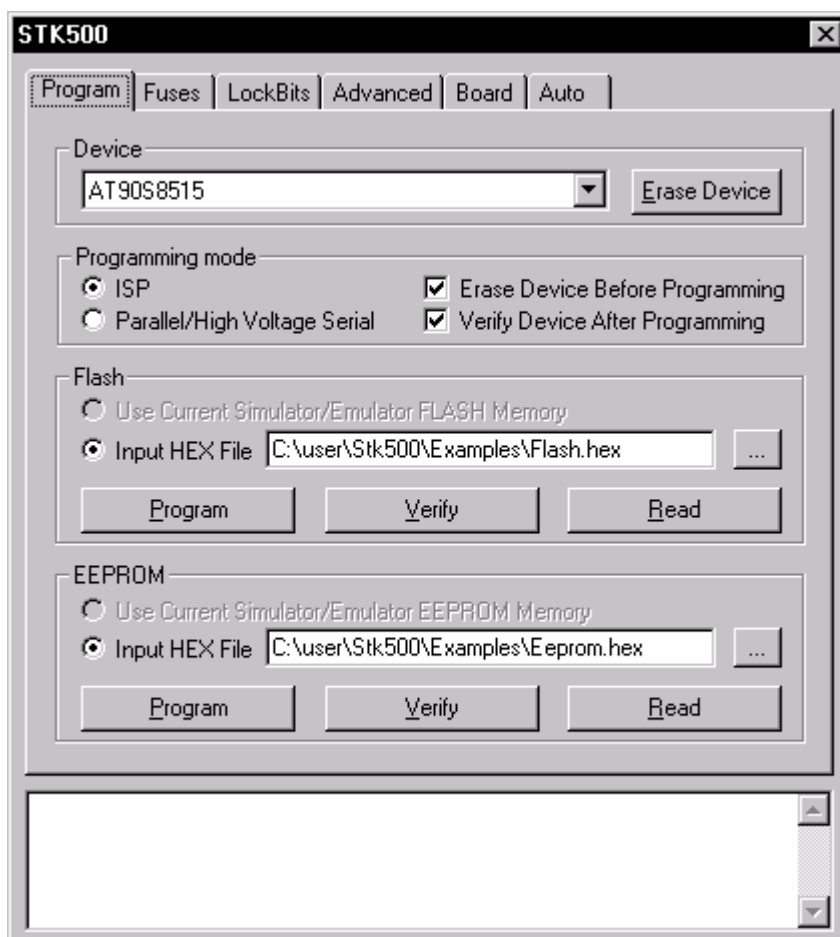
Programming Mode

This group selects programming mode. For devices only supporting High-Voltage programming, the ISP option will be grayed out. If both modes are available, select mode by clicking on the correct method. Checking the "Erase Device Before Programming" will force STK500 to perform a chip-erase before programming code to the program memory (Flash). Checking the "Verify Device After Programming" will force STK500 to perform a verification of the memory after programming it (Both Flash and EEPROM).

Flash

If the STK500 User Interface is opened without a project loaded in AVR Studio, the "Use Current Simulator/Emulator FLASH Memory" option will be grayed out. When a project is open this option allows programming of the Flash memory content currently present in the Flash Memory view of AVR Studio.


If no project is running, or the source code is stored in a separate HEX file, select the "Input HEX File" option. Browse to the correct file by pressing the  button, or writing the complete path and filename in the text field. The selected file must be in "Intel-hex" format or "extended Intel-hex" format.



EEPROM

If STK500 User Interface is opened without a project loaded in AVR Studio, the "Use Current Simulator/Emulator EEPROM Memory" option will be grayed out. When a project is open this

option allows programming of the EEPROM memory content currently present in the EEPROM Memory view.

If no project is running, or the source code is stored in a separate HEX file, select the "Input HEX File" option. Browse to the correct file by pressing the  button, or writing the complete path and filename in the text field. The selected file must be in "Intel-hex" format or "extended Intel-hex" format.

See Also

- "Fuse" Settings
- "Lock Bits" Settings
- "Advanced" Settings
- "Board" Settings
- "Auto" Settings
- History Window

Example Application: Using LEDs and Switches

Connect PORTB to LEDS and PORTD to SWITCHES. LEDS will operate differently depending on what switch is pressed.

Tip: Copy the code from this document into AVR Studio

```
;***** STK500 LEDS and SWITCH demonstration
#include "8515def.inc"
.def Temp =r16 ; Temporary register
.def Delay =r17 ; Delay variable 1
.def Delay2 =r18 ; Delay variable 2
;***** Initialization
RESET:
    ser        Temp
    out        DDRB,Temp                ; Set PORTB to output
;**** Test input/output
LOOP:
    out        PORTB,temp                ; Update LEDS
    sbis       PIND,0x00                ; If (Port D, pin0 == 0)
    inc        Temp                    ; then count LEDS one down
    sbis       PIND,0x01                ; If (Port D, pin1 == 0)
    dec        Temp                    ; then count LEDS one up
    sbis       PIND,0x02                ; If (Port D, pin2 == 0)
    ror        Temp                    ; then rotate LEDS one right
    sbis       PIND,0x03                ; If (Port D, pin3 == 0)
    rol        Temp                    ; then rotate LEDS one left
    sbis       PIND,0x04                ; If (Port D, pin4 == 0)
    com        Temp                    ; then invert all LEDS
    sbis       PIND,0x05                ; If (Port D, pin5 == 0)
    neg        Temp                    ; then invert all LEDS and add 1
    sbis       PIND,0x06                ; If (Port D, pin6 == 0)
    swap       Temp                    ; then swap nibbles of LEDS
;**** Now wait a while to make LED changes visible.
DLY:
    dec        Delay
    brne       DLY
    dec        Delay2
    brne       DLY
    rjmp       LOOP                    ; Repeat loop forever
```