Section 1

AVR Studio User Guide

1.1 Introduction

Welcome to *AVR* Studio from Atmel Corporation. *AVR* Studio is a Development Tool for the AT90S Series of AVR microcontrollers. This manual describes the how to install and use *AVR* Studio.

AVR Studio enables the user to fully control execution of programs on the AT90S In-Circuit Emulator or on the built-in *AVR* Instruction Set Simulator. *AVR* Studio supports source level execution of Assembly programs assembled with the Atmel Corporation's *AVR* Assembler and C programs compiled with IAR Systems' ICCA90 C Compiler for the *AVR* microcontrollers.

AVR Studio runs under Microsoft Windows95 and Microsoft Windows NT.

1.2 Installing *AVR* Studio

AVR Studio is delivered on two diskettes. Note that **in some cases, the second diskette will not be asked for** by the installation program. This is because some of the files required to run *AVR* Studio may already be present in the system.

In order to install *AVR* Studio under **Windows95** and **Windows NT 4.0**:

1. Insert the diskette labeled *AVR* Studio Diskette 1 in drive A:
2. Press the Start button on the Taskbar and select Run
3. Enter "A:SETUP" in the Open field and press the OK button
4. Follow the instructions in the Setup program

In order to install *AVR* Studio under **Windows NT 3.51**:

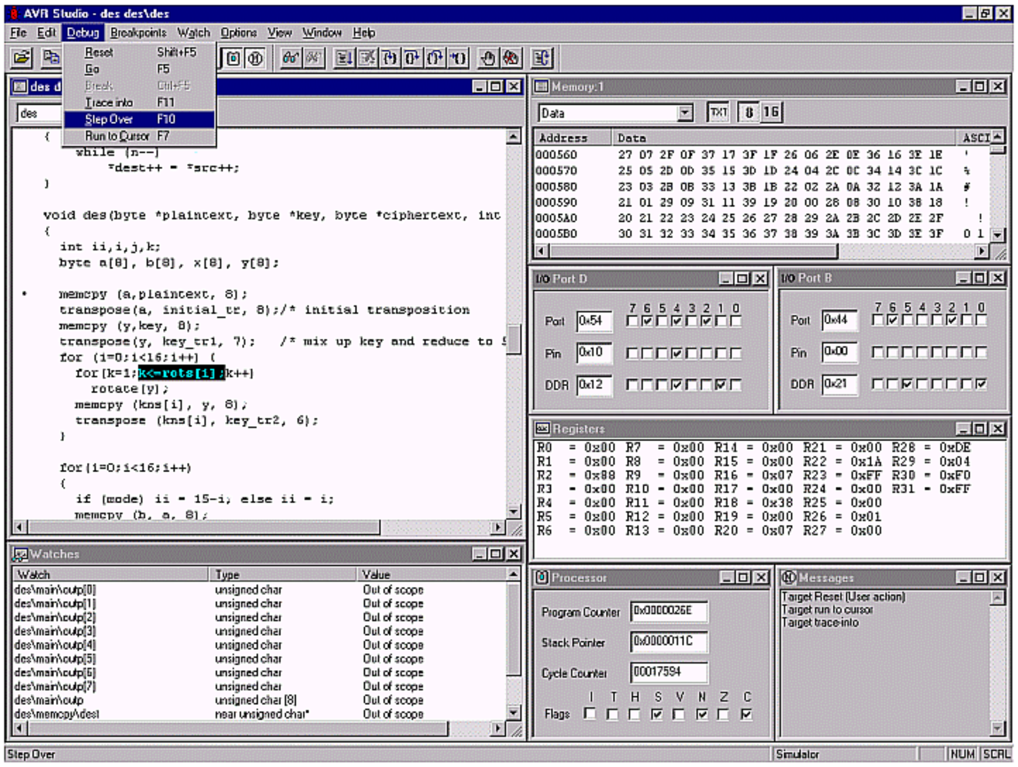
1. Insert the diskette labeled *AVR* Studio Diskette 1 in drive A:
2. Select Run from the File menu
3. Enter "A:SETUP" in the Command Line field and press the OK button
4. Follow the instructions in the Setup program

Once *AVR* Studio has been installed, it can be started by double clicking the *AVR* Studio icon. If an Emulator is the desired execution target, remember to connect the *AVR* In-Circuit Emulator before starting *AVR* Studio.

1.3 Description

This section gives a brief description of the main features of *AVR* Studio. *AVR* Studio enables execution of *AVR* programs on an *AVR* In-Circuit Emulator or the built-in *AVR* Instruction Set Simulator. In order to execute a program using *AVR* Studio, it must first be compiled with IAR Systems' C Compiler or assembled with Atmel's *AVR* Assembler to generate an object file which can be read by *AVR* Studio.

An example of what *AVR* Studio may look like during execution of a program is shown below. In addition to the Source window, *AVR* Studio defines a number of other windows which can be used for inspecting the different resources on the microcontroller.



```
    {
      while (n--)
          *dest++ = *src++;
    }

    void des(byte *plaintext, byte *key, byte *ciphertext, int
    {
      int ii,i,j,k;
      byte a[8], b[8], x[8], y[8];

•     memcpy (a,plaintext, 8);
      transpose(a, initial_tr, 8);/* initial transposition
      memcpy (y,key, 8);
      transpose(y, key_tr1, 7);   /* mix up key and reduce to 5
      for (i=0;i<16;i++) {
        for(k=1;k<=rots[i];k++)
          rotate(y);
        memcpy (kns[i], y, 8);
        transpose (kns[i], key_tr2, 6);
      }

      for(i=0;i<16;i++)
      {
        if (mode) ii = 15-i; else ii = i;
        memcpy (b, a, 8);
```

```
Address  Data                                             ASCI
000560   27 07 2F 0F 37 17 3F 1F 26 06 2E 0E 36 16 3E 1E  '
000570   25 05 2D 0D 35 15 3D 1D 24 04 2C 0C 34 14 3C 1C  %
000580   23 03 2B 0B 33 13 3B 1B 22 02 2A 0A 32 12 3A 1A  #
000590   21 01 29 09 31 11 39 19 20 00 28 08 30 10 38 18  !
0005A0   20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !
0005B0   30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F  0 1
```

I/O Port D

		7	6	5	4	3	2	1	0
Port	0x54	☐	✓	☐	✓	☐	✓	☐	☐
Pin	0x10	☐	☐	☐	✓	☐	☐	☐	☐
DDR	0x12	☐	☐	☐	✓	☐	☐	✓	☐

I/O Port B

		7	6	5	4	3	2	1	0
Port	0x44	☐	✓	☐	☐	☐	✓	☐	☐
Pin	0x00	☐	☐	☐	☐	☐	☐	☐	☐
DDR	0x21	☐	☐	✓	☐	☐	☐	☐	✓

```
R0  = 0x00 R7  = 0x00 R14 = 0x00 R21 = 0x00 R28 = 0xDE
R1  = 0x00 R8  = 0x00 R15 = 0x00 R22 = 0x1A R29 = 0x04
R2  = 0x88 R9  = 0x00 R16 = 0x07 R23 = 0xFF R30 = 0xF0
R3  = 0x00 R10 = 0x00 R17 = 0x00 R24 = 0x00 R31 = 0xFF
R4  = 0x00 R11 = 0x00 R18 = 0x38 R25 = 0x00
R5  = 0x00 R12 = 0x00 R19 = 0x00 R26 = 0x01
R6  = 0x00 R13 = 0x00 R20 = 0x07 R27 = 0x00
```

Watch	Type	Value
des\main\outp[0]	unsigned char	Out of scope
des\main\outp[1]	unsigned char	Out of scope
des\main\outp[2]	unsigned char	Out of scope
des\main\outp[3]	unsigned char	Out of scope
des\main\outp[4]	unsigned char	Out of scope
des\main\outp[5]	unsigned char	Out of scope
des\main\outp[6]	unsigned char	Out of scope
des\main\outp[7]	unsigned char	Out of scope
des\main\outp	unsigned char [8]	Out of scope
des\memcpy\dest	near unsigned char*	Out of scope

Program Counter 0x0000026E
Stack Pointer 0x0000011C
Cycle Counter 00017594

Flags	I	T	H	S	V	N	Z	C
	☐	☐	☐	✓	☐	✓	☐	✓

```
Target Reset (User action)
Target run to cursor
Target trace-into
```

The key window in *AVR* Studio is the Source window. When an object file is opened, the Source window is automatically created. The Source window displays the code currently being executed on the execution target (i.e. the Emulator or the Simulator), and the text marker is always placed on the next statement to be executed. The Status bar indicates whether the execution target is the *AVR* In-Circuit Emulator or the built-in Instruction Set Simulator.

By default, it is assumed that execution is done on source level, so if source information exists, the program will start up in source level mode. In addition to source level execution of both C and Assembly programs, *AVR* Studio can also view and execute programs on a disassembly level. The user can toggle between source and disassembly mode when execution of the program is stopped.

All necessary execution commands are available in *AVR* Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until that statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code breakpoints, and every breakpoint can be defined as enabled or disabled. The breakpoints are remembered between sessions.

The Source window gives information about the control flow of the program. In addition, *AVR* Studio offers a number of other windows which enables the user to have full control of the status of every element in the execution target. The available windows are:

1. Watch window: Displays the values of defined symbols. In the Watch window, the user can watch the values of for instance variables in a C program.
2. Register window: Displays the contents of the register file. The registers can be modified when the execution is stopped.

3. Memory windows: Displays the contents of the Program Memory, Data Memory, I/O Memory or EEPROM Memory. The memories can be viewed as hexadecimal values or as ASCII characters. The memory contents can be modified when the execution is stopped.
4. Peripheral windows: Displays the contents of the status registers associated with the different peripheral devices:
   - EEPROM Registers
   - I/O Ports
   - Timers
   - etc.
5. Message window: Displays messages from *AVR* Studio to the user
6. Processor window: Displays vital information about the execution target, including Program Counter, Stack Pointer, Status Register and Cycle Counter. These parameters can be modified when the execution is stopped.

The first time an object file is being executed, the user needs to set up the windows which are convenient for observing the execution of the program, thereby tailoring the information on the screen to the specific project. The next time that object file is loaded, the setup is automatically reconstructed

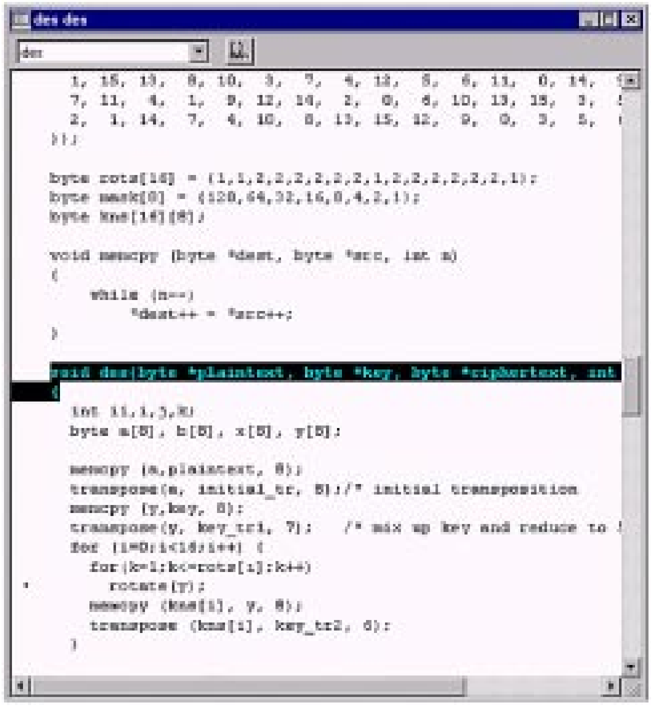
The different windows will be described more carefully in the next chapter.

1.4 *AVR* Studio Windows

1.4.1 Source window

The Source window is the main window in an *AVR* Studio session. It is created when an object file is opened, and is present throughout the session. If the Source window is closed, the session is terminated.

The Source window displays the code which is being executed. An example of a Source window is given below.



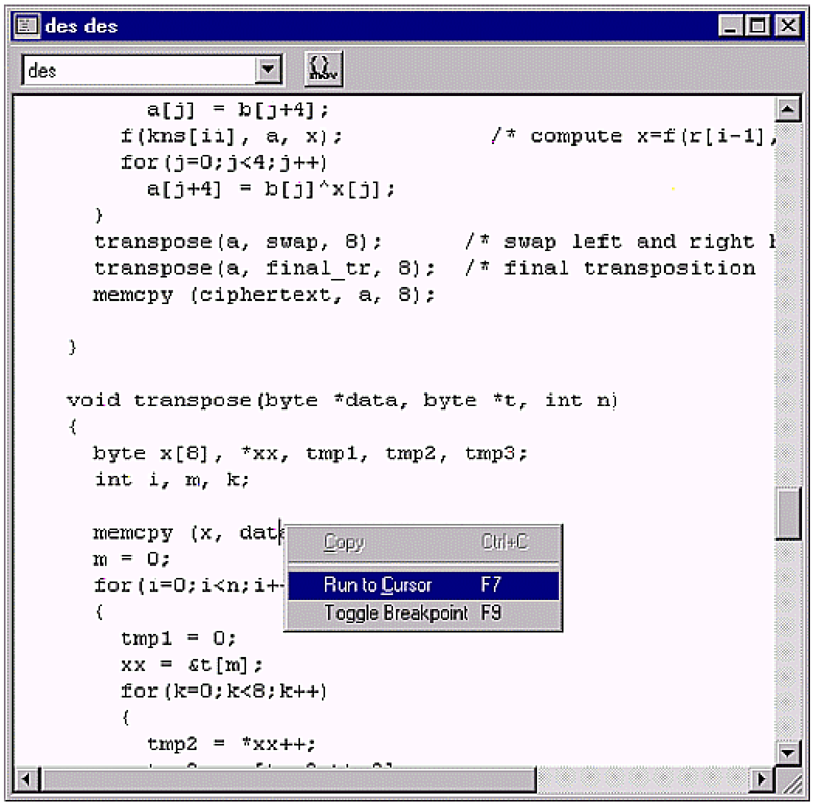
```
    1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,
    7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,
    2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,
  }};

  byte rots[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};
  byte mask[8] = {128,64,32,16,8,4,2,1};
  byte kns[16][8];

  void memcpy (byte *dest, byte *src, int n)
  {
      while (n--)
          *dest++ = *src++;
  }

  void des(byte *plaintext, byte *key, byte *ciphertext, int
  {
    int ii,i,j,k;
    byte a[8], b[8], x[8], y[8];

    memcpy (a,plaintext, 8);
    transpose(a, initial_tr, 8);/* initial transposition
    memcpy (y,key, 8);
    transpose(y, key_tr1, 7);   /* mix up key and reduce to
    for (i=0;i<16;i++) {
      for(k=1;k<=rots[i];k++)
        rotate(y);
      memcpy (kns[i], y, 8);
      transpose (kns[i], key_tr2, 6);
    }
```

The next instruction to be executed is always marked by *AVR* Studio. If the marker is moved by the user, this next statement can still be identified since the previously marked text becomes red.

A breakpoint is identified in the Source window as a dot to the left of the statement where the breakpoint is set.

If the button to the right of the module selection box is pressed, the Source window switches between source level and disassembly level execution. When *AVR* Studio is in disassembly mode, all operations, such as Single stepping, is done on disassembly level. In some cases, no source level information is available, for instance if an Intel-Hex file is selected as the object file. When no source level information is available, execution must be done on disassembly level.

The Toggle breakpoint, Run to Cursor and the Copy functions are also available by pressing the right mouse button in the Source window. When the right mouse button is pressed, a menu appears on the screen:



```
      a[j] = b[j+4];
    f(kns[ii], a, x);          /* compute x=f(r[i-1],
    for(j=0;j<4;j++)
      a[j+4] = b[j]^x[j];
  }
  transpose(a, swap, 8);      /* swap left and right b
  transpose(a, final_tr, 8);  /* final transposition
  memcpy (ciphertext, a, 8);

}

void transpose(byte *data, byte *t, int n)
{
  byte x[8], *xx, tmp1, tmp2, tmp3;
  int i, m, k;

  memcpy (x, dat
  m = 0;
  for(i=0;i<n;i+
  {
    tmp1 = 0;
    xx = &t[m];
    for(k=0;k<8;k++)
    {
      tmp2 = *xx++;
```

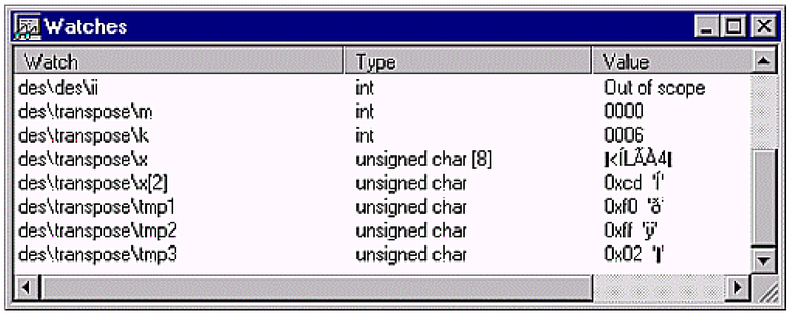
If the cursor is placed on a statement and a Run to Cursor command is issued, the program will execute until it reaches the instruction where the cursor is placed. Breakpoints are set in a similar way: the cursor is placed on a statement, and a Toggle Breakpoint command is issued. If a breakpoint was already set on the statement, the breakpoint will be removed. If no breakpoint was set on the statement, a breakpoint is inserted.

An object file can consist of several modules. Only one module is displayed at a time, but the user can change to the other modules by selecting the module of interest in the selection box on the top left of the Source window. This is a useful feature for viewing and setting breakpoints in other modules than the one currently active.

The Source window supports the Windows Clipboard. The user can select parts of (or all) the contents in the Source window and then copy it to the Windows Clipboard by selecting Copy from the Edit menu.

1.4.2 Watch window

The Watch window can display the types and values of symbols like for instance variables in a C program. Since the *AVR* Assembler does not generate any symbol information, this window can only be used in a meaningful way when executing C programs. An example of a Watch window is given below.



Watch	Type	Value
des\des\ii	int	Out of scope
des\transpose\m	int	0000
des\transpose\k	int	0006
des\transpose\x	unsigned char [8]	▌kÍLĂÅ4▌
des\transpose\x[2]	unsigned char	0xcd 'Í'
des\transpose\tmp1	unsigned char	0xf0 'ð'
des\transpose\tmp2	unsigned char	0xff 'ÿ'
des\transpose\tmp3	unsigned char	0x02 '▌'

The Watch window has three fields. The first field is the name of the symbol which is being watched. The next is the type of the symbol, and the third is the value of the symbol. By default, the Watch window is empty, i.e. all the symbols the user would like to watch have to be added to the Watch window. Once a symbol has been added, it is remembered also in subsequent executions of the programs. The added watches are also remembered if the Watch window is closed.

There are commands for adding watches, deleting watches and deleting all watches. A watch is added by giving an Add Watch command from the Watch menu or from the Debug toolbar. A watch can also be added by pressing the INS key if the Watch window is the active window. When an Add Watch command is issued, the user must enter the name of the symbol. The user can enter a symbol name with or without scope information.

AVR Studio will first search for the symbol as if it contains scope information. If no such symbol is found, *AVR* Studio appends the symbol name to the current scope, and searches for this new symbol. If no such symbol is found, the symbol is unbound, "???" appears in the type field, and the value field remains empty. If the symbol name is found, the symbol is bound, the symbol with scope information is displayed in the watch field, and the type and value fields are filled out. Every time execution stops, *AVR* Studio tries to bind unbound symbols using the current scope.

It is not possible to have floating symbols. Once a symbol is bound, it remains bound. The watches are remembered between sessions. Whether or not the symbol has been bound is a part of this information. If the program enters a scope where a bound symbol is not visible, the value field changes to "Out of scope".

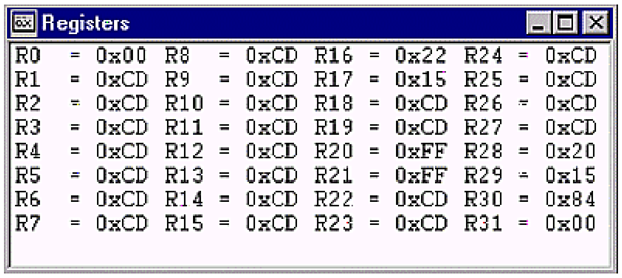
In order to delete a watch, the symbol name must first be clicked on using the left mouse button. When a symbol has been marked this way, *AVR* Studio accepts the Delete Watch command from the Watch menu. If the Watch window is the currently active window, the marked symbol can also be deleted by pressing the DEL key.

The Watch window can be used for watching C arrays and structs as well as simple variables. The syntax is the same as in C (use braces ('[' and ']') for arrays and dot ('.') for structs). Dereferencing pointers is not supported. When watching arrays, variables can be used for dynamically indexing the arrays. It is for example possible to watch "my_array[i]" if i is an integer in the same scope as the array my_array.

There can only be one Watch window active at a time. The watched symbols (with scope information) are remembered between sessions. The Watch window can also be toggled on and off, and the watches are also remembered if the Watch window is toggled on and off.

1.4.3 Register window

The Register window displays the contents of the 32 registers in the *AVR* register file. An example of the Register window is given below.



```
Registers
R0  = 0x00 R8  = 0xCD R16 = 0x22 R24 = 0xCD
R1  = 0xCD R9  = 0xCD R17 = 0x15 R25 = 0xCD
R2  = 0xCD R10 = 0xCD R18 = 0xCD R26 = 0xCD
R3  = 0xCD R11 = 0xCD R19 = 0xCD R27 = 0xCD
R4  = 0xCD R12 = 0xCD R20 = 0xFF R28 = 0x20
R5  = 0xCD R13 = 0xCD R21 = 0xFF R29 = 0x15
R6  = 0xCD R14 = 0xCD R22 = 0xCD R30 = 0x84
R7  = 0xCD R15 = 0xCD R23 = 0xCD R31 = 0x00
```

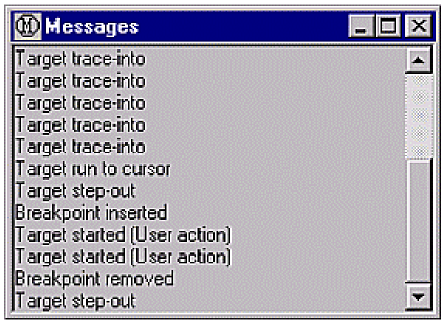
When the Register window is resized, the contents is reorganized in order to best fit the shape of the window.

The values in the Register window can be changed when the execution is stopped. In order to change the contents of a register, first make sure the execution is stopped. Then place the cursor on the register to change, press the left mouse button twice (not a double click, make sure to make a pause between the mouse button clicks). The register can then be changed. Type in the new contents in hexadecimal form. Finally, press the Enter key to confirm or the ESC key to cancel the change.

Only one Register window can be active at a time.

1.4.4 Message window

The Message window displays messages from *AVR* Studio to the user. When a Reset command is issued, the contents of the Message window is cleared. An example of a Message window is given below.



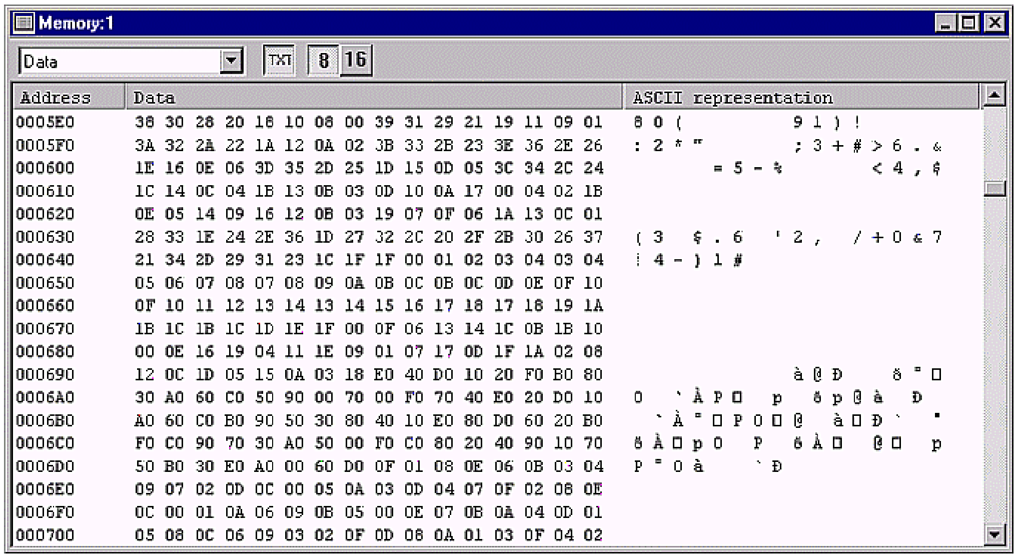
```
Messages
Target trace-into
Target trace-into
Target trace-into
Target trace-into
Target trace-into
Target run to cursor
Target step-out
Breakpoint inserted
Target started (User action)
Target started (User action)
Breakpoint removed
Target step-out
```

The contents in the Message window is remembered also when the Message window is toggled off and then on again. Only one Message window can be active at a time.

1.4.5 Memory window

The Memory window enables the user to inspect and modify the contents of the various memories present in the execution target. The same window is used to view all memory types. The Memory window can be used to view Data memory, Program memory, I/O memory and EEPROM memory.

The user can have several concurrent Memory windows. An example of a Memory window is shown below.



```
Address    Data                                               ASCII representation
0005E0     38 30 28 20 18 10 08 00 39 31 29 21 19 11 09 01    8 0 (           9 1 ) !
0005F0     3A 32 2A 22 1A 12 0A 02 3B 33 2B 23 3E 36 2E 26    : 2 * "         ; 3 + # > 6 . &
000600     1E 16 0E 06 3D 35 2D 25 1D 15 0D 05 3C 34 2C 24            = 5 - %         < 4 , $
000610     1C 14 0C 04 1B 13 0B 03 0D 10 0A 17 00 04 02 1B
000620     0E 05 14 09 16 12 0B 03 19 07 0F 06 1A 13 0C 01
000630     28 33 1E 24 2E 36 1D 27 32 2C 20 2F 2B 30 26 37    ( 3   $ . 6   ' 2 ,   / + 0 & 7
000640     21 34 2D 29 31 23 1C 1F 1F 00 01 02 03 04 03 04    ¦ 4 - ) 1 #
000650     05 06 07 08 07 08 09 0A 0B 0C 0B 0C 0D 0E 0F 10
000660     0F 10 11 12 13 14 13 14 15 16 17 18 17 18 19 1A
000670     1B 1C 1B 1C 1D 1E 1F 00 0F 06 13 14 1C 0B 1B 10
000680     00 0E 16 19 04 11 1E 09 01 07 17 0D 1F 1A 02 08
000690     12 0C 1D 05 15 0A 03 18 E0 40 D0 10 20 F0 B0 80                    à @ Ð     ð ° □
0006A0     30 A0 60 C0 50 90 00 70 00 F0 70 40 E0 20 D0 10    0   ` À P □   p   ð p @ à   Ð
0006B0     A0 60 C0 B0 90 50 30 80 40 10 E0 80 D0 60 20 B0      ` À ° □ P 0 □ @   à □ Ð `   °
0006C0     F0 C0 90 70 30 A0 50 00 F0 C0 80 20 40 90 10 70    ð À □ p 0   P   ð À □   @ □   p
0006D0     50 B0 30 E0 A0 00 60 D0 0F 01 08 0E 06 0B 03 04    P ° 0 à     ` Ð
0006E0     09 07 02 0D 0C 00 05 0A 03 0D 04 07 0F 02 08 0E
0006F0     0C 00 01 0A 06 09 0B 05 00 0E 07 0B 0A 04 0D 01
000700     05 08 0C 06 09 03 02 0F 0D 08 0A 01 03 0F 04 02
```

Which Memory type to view can be changed in the memory selection box at the top left of the Memory window. When a new Memory window is created, Data memory is the default memory type. *AVR* Studio not only keeps track over where the Memory windows are placed, but also which memory type it is displaying, and also the formatting status of the Window.

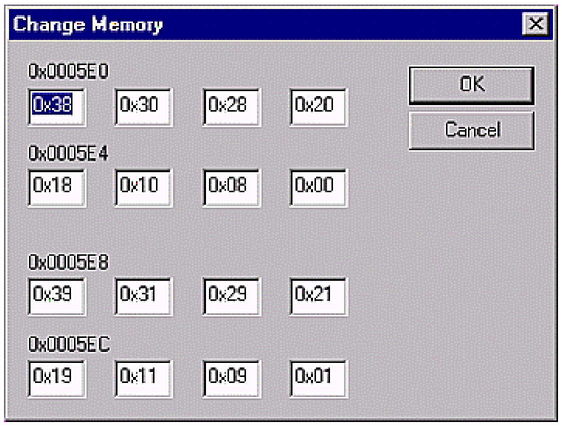
A hexadecimal representation of the addresses and the contents of the memory is always displayed. In addition, the user can view the memory contents as ASCII characters. The user also has the option to group the hexadecimal representation into 16 bit groups in stead of 8 bit groups.

When viewing Program memory, it is the Word address which is displayed in the address column, and the MSB is listed before the LSB in the data column.

1.4.5.1 Modifying memory

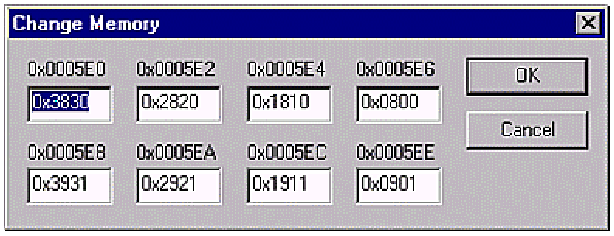
The user can modify the contents of the memories by issuing a double click on the line containing the item(s) to be changed. When a line in the Memory view is doubleclicked, a Window appears on the screen. If memory is viewed in 8 bit groups, the modifications are done on 8 bit groups and when memory is viewed as 16 bit groups, the modifications are done on 16 bit groups.

When operating on 8 bit groups, the following Window appears:



0x0005E0			
0x38	0x30	0x28	0x20
0x0005E4			
0x18	0x10	0x08	0x00
0x0005E8			
0x39	0x31	0x29	0x21
0x0005EC			
0x19	0x11	0x09	0x01

When operating on 16 bit groups, the following Window appears:

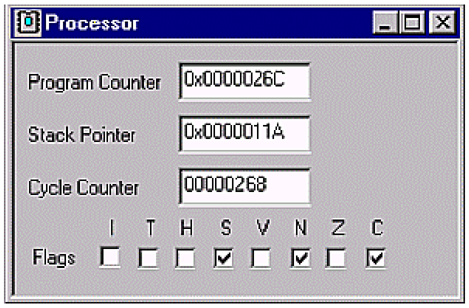


0x0005E0	0x0005E2	0x0005E4	0x0005E6
0x3830	0x2820	0x1810	0x0800
0x0005E8	0x0005EA	0x0005EC	0x0005EE
0x3931	0x2921	0x1911	0x0901

The operation is the same in the two cases. If the Cancel button is pressed, no update is done even if the user has edited one or more of the values. If the OK button is pressed, the Memory is updated if one or more of the values are changed.

1.4.6 Processor window

The Processor window contains vital information about the execution target. An example of a Processor window is shown below.



Program Counter 0x0000026C

Stack Pointer 0x0000011A

Cycle Counter 00000268

	I	T	H	S	V	N	Z	C
Flags	☐	☐	☐	✓	☐	✓	☐	✓

The Program Counter indicates the address of the next instruction to be executed. The Program Counter is displayed in hexadecimal form, and can be changed when the execution is stopped. When the Program Counter is changed, the current instruction is discarded. After the Program Counter is changed, the user must press the Single step function to jump to the desired address.

The Stack Pointer holds the current value of the Stack Pointer which is placed in the I/O area. If the Target has a Hardware stack instead of an SRAM based stack, this is indicated in the Stack Pointer field. The Stack Pointer value can be changed when the execution is stopped.

The Cycle Counter gives information about the number of clock cycles elapsed since last reset. The *AVR* In-Circuit Emulator does at time being not support a cycle counter so the Cycle Counter is always zero when the Emulator is the execution target. The Cycle Counter value is displayed as a decimal value and can be changed when the execution is stopped.

The Flags is a display of the current value of the Status register. When the execution is stopped, these bits can be changed by clicking on the flags to change. A checked flag indicates that the flag is set (the corresponding bit in the Status register has the value 1).

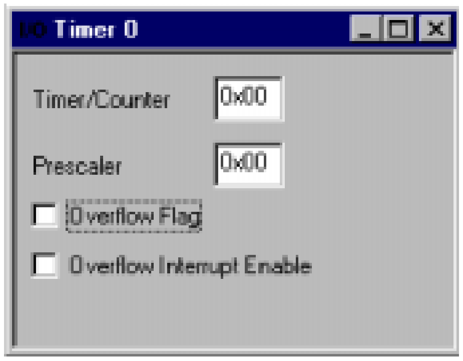
Only one Processor window can be active at a time.

1.4.7 Peripheral Device windows

The user can watch the contents of the I/O in the Memory window. Viewing the I/O area as a flat memory structure is not a very convenient way of observing the status of the many I/O devices of the microcontroller in question. Specialized Device windows have therefore been incorporated to ease the observation of I/O devices.

1.4.7.1 Timer/Counter 0

The Timer/Counter 0 window displays all essential information about Timer/Counter 0. When the Timer/Counter 0 is selected from the View → Peripherals → AT90SXXXX → Timer 0 menu, the following window appears on the screen:



Timer/Counter 0x00

Prescaler 0x00

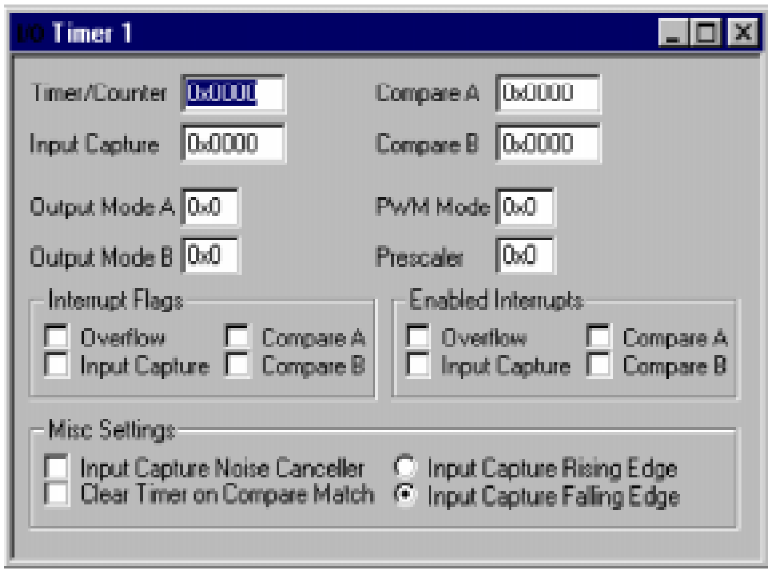
☐ Overflow Flag

☐ Overflow Interrupt Enable

The Timer/Counter field gives the value of Timer/Counter 0. The prescaler field gives the value of the corresponding prescaler. The Overflow Flag check box and the Overflow Interrupt Enable check boxes gives the status and control bits of Timer 0.

1.4.7.2 Timer/Counter 1

The Timer/Counter 1 window displays all essential information about Timer/Counter 1. When the Timer/Counter 1 is selected from the View → Peripherals → AT90SXXXX → Timer 1 menu, the following window appears on the screen:



Timer/Counter 0x0000
Input Capture 0x0000
Compare A 0x0000
Compare B 0x0000
Output Mode A 0x0
Output Mode B 0x0
PWM Mode 0x0
Prescaler 0x0

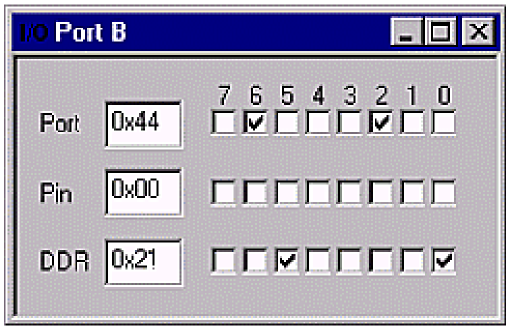
Interrupt Flags: ☐ Overflow ☐ Compare A ☐ Input Capture ☐ Compare B
Enabled Interrupts: ☐ Overflow ☐ Compare A ☐ Input Capture ☐ Compare B

Misc Settings: ☐ Input Capture Noise Canceller ☐ Clear Timer on Compare Match ○ Input Capture Rising Edge ● Input Capture Falling Edge

The Timer/Counter 1 window displays in detail all the different parameters of Timer/Counter 1. A description of the different features of Timer/Counter 1 is given in the AVR Data Book. All the values can be changed when execution is stopped.

1.4.7.3 Port window

The Port window displays the three different I/O registers usually associated with a port. WHen the user selects a port from the View → Peripherals → AT90SXXXX → Port menu, the corresponding Port window appears:

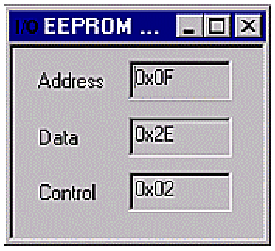


		7	6	5	4	3	2	1	0
Port	0x44	☐	✓	☐	☐	☐	✓	☐	☐
Pin	0x00	☐	☐	☐	☐	☐	☐	☐	☐
DDR	0x21	☐	☐	✓	☐	☐	☐	☐	✓

The Port window displays the setting of the Port, Pin and Data Direction Registers from the I/O area, both as hexadecimal values and as single bits. WHen execution is stopped, the values of the registers can be changed.

1.4.7.4 EEPROM Registers

When EEPROM Registers is selected from the View → Peripherals → AT90SXXXX → EEPROM Registers menu, the following window appears:



Address 0x0F

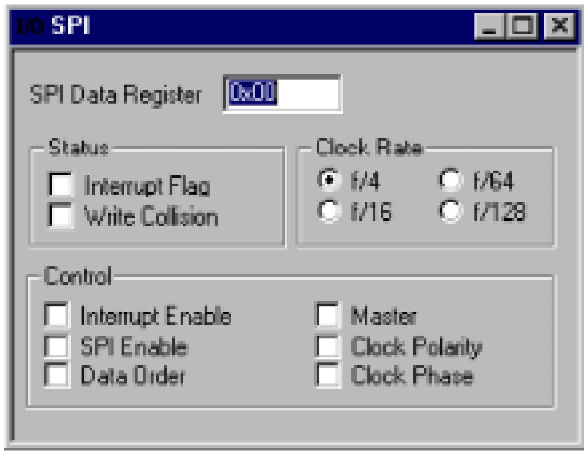
Data 0x2E

Control 0x02

AVR Studio knows how much EEPROM memory is available on the Target, so if required, the Address field contains the high byte of the address concatenated with the low byte of the address.

1.4.7.5 SPI window

The SPI window displays all essential information about the SPI. When the Timer/Counter 1 is selected from the View → Peripherals → AT90SXXXX → SPI menu, the SPI window appears on the screen:



SPI Data Register 0x00

Status
- ☐ Interrupt Flag
- ☐ Write Collision

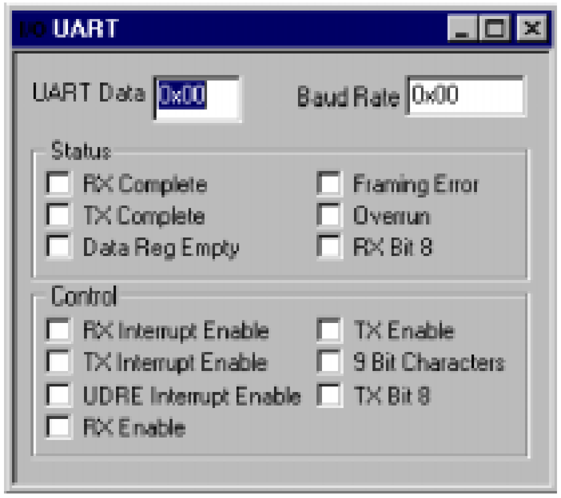
Clock Rate
- [✓ f/4]
- f/16
- f/64
- f/128

Control
- ☐ Interrupt Enable
- ☐ SPI Enable
- ☐ Data Order
- ☐ Master
- ☐ Clock Polarity
- ☐ Clock Phase

The SPI Data Register shows the SPI receive register. Editing the SPI Data Register value will not start sending data on the SPI even if the SPI is enabled. Initiating an SPI transfer can only be achieved by making the program running on the Target write to the SPI Data Register. The user can also observe and change the values of all the bits in the control and status registers.

1.4.7.6 UART window

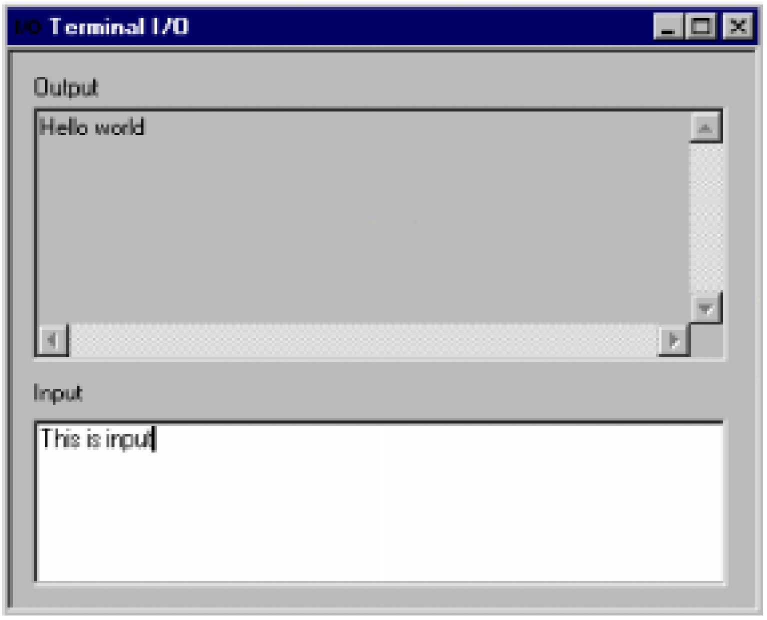
The UART window displays all essential information about the UART. When the UART is selected from the View → Peripherals → AT90SXXXX → UART menu, the UART window appears on the screen:



The UART window displays the UART Data Register, Baud Rate Register and the bits of the Control and Status Registers. Writing to the UART Data Register will not initiate a data transfer. The Data register must be written by the program executing on the Target.

1.4.7.7 Terminal I/O window

The Terminal I/O window enables simulation of Terminal I/O communication with a program executing in AVR Studio. When the Terminal I/O is selected from the View → Terminal I/O menu, the following window appears on the screen:



```
Hello world
```

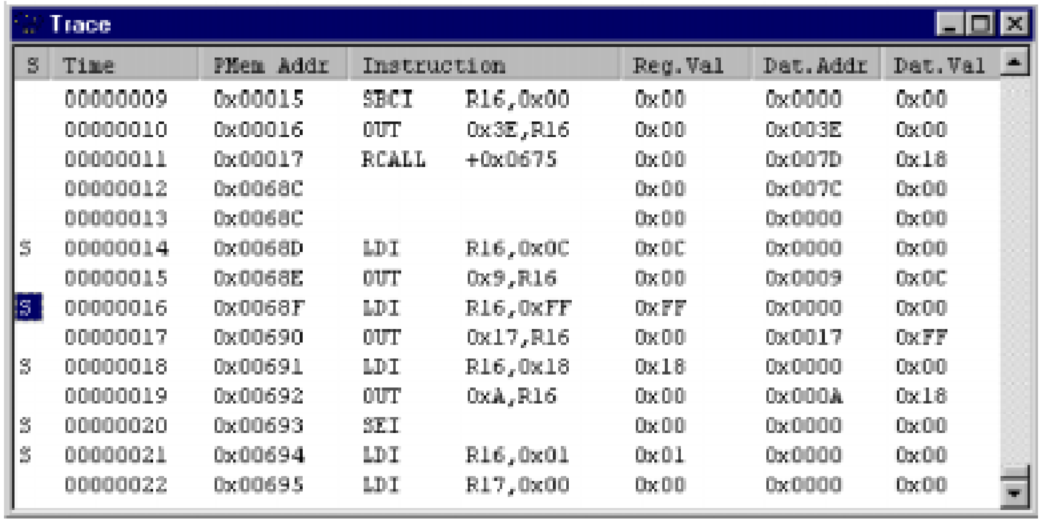
```
This is input
```

The Terminal I/O window is only available when using the Simulator and when using the IAR C Compiler to generate code. The Compiler must be set up to generate the Debug format with Terminal I/O (default). The output from the program will the be sent to the Output part of the Terminal I/O window and the input to the program will be read from the Input part of the Terminal I/O window. When a character is read from the Input part, it is removed from the input view.

The window has a fixed size.

1.4.7.8 Trace window

The Trace window keeps track of the history of the program currently being executed. When the Trace window is selected from the View → Trace menu, the Trace window appears on the screen:



S	Time	PMem Addr	Instruction		Reg.Val	Dat.Addr	Dat.Val
	00000009	0x00015	SBCI	R16,0x00	0x00	0x0000	0x00
	00000010	0x00016	OUT	0x3E,R16	0x00	0x003E	0x00
	00000011	0x00017	RCALL	+0x0675	0x00	0x007D	0x18
	00000012	0x0068C			0x00	0x007C	0x00
	00000013	0x0068C			0x00	0x0000	0x00
S	00000014	0x0068D	LDI	R16,0x0C	0x0C	0x0000	0x00
	00000015	0x0068E	OUT	0x9,R16	0x00	0x0009	0x0C
S	00000016	0x0068F	LDI	R16,0xFF	0xFF	0x0000	0x00
	00000017	0x00690	OUT	0x17,R16	0x00	0x0017	0xFF
S	00000018	0x00691	LDI	R16,0x18	0x18	0x0000	0x00
	00000019	0x00692	OUT	0xA,R16	0x00	0x000A	0x18
S	00000020	0x00693	SEI		0x00	0x0000	0x00
S	00000021	0x00694	LDI	R16,0x01	0x01	0x0000	0x00
	00000022	0x00695	LDI	R17,0x00	0x00	0x0000	0x00

The Trace window is only available when using the Simulator. The column to the left defines the synchronization points to the code. If a synchronization character is double-clicked, the marker in the source window is placed accordingly to mark the actual instruction. If the source window is in source mode, the source level synchronization points are indicated, whereas in disassembly mode, all instructions can be used as synchronization points.

The Trace buffer is 32K cycles deep.

1.5 Commands

AVR Studio incorporates a number of different commands. The commands can be given in various ways: through menu selections, toolbar buttons and by keyboard shortcuts. This section describes the available commands, and how they are invoked.

1.5.1 Administrative

1.5.1.1 Opening files

When Open is selected from the File menu, a file selection dialog appears on the screen (note that *AVR* assumes the file extension .OBJ, so by default, only files with this extension are listed). The user must then select the object file to execute. Currently, the *AVR* Studio supports the following formats:

- IAR UBROF
- *AVR* Object Files generated by the Atmel *AVR* Assembler
- Intel-Hex

AVR Studio automatically detects the format of the object file.

The four most recently used files are also available under the File menu and can be selected for loading directly.

When opening the file, *AVR* Studio looks for a file with the same filename as the file selected but with the extension AVD. This is a file *AVR* Studio generates when a file is closed, and it contains information about the project, including window placement. If the AVD project file is not found, only a Source window is created.

The AVD file also contains information regarding breakpoints. Breakpoints defined in the previous session are reinserted unless the object file is newer than the project file. In the latter case, the breakpoints are discarded.

If source level information is available, the program is executed until the first source statement is reached.

1.5.1.2 Closing files

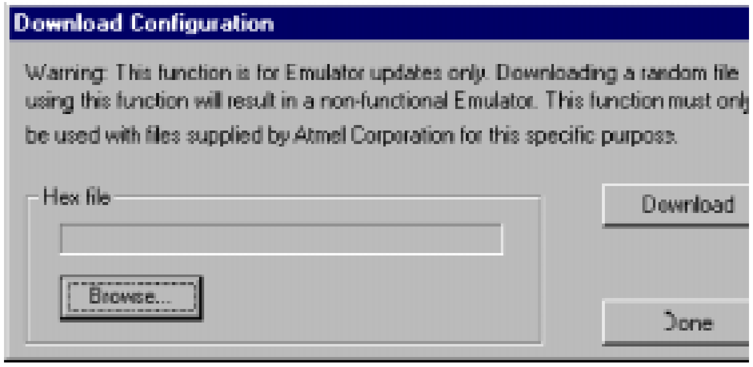
When Close is selected from the File menu, all the windows in a session are closed. *AVR* Studio also writes a file in the same directory as the object file, containing project information. The file has the same name as the object file, but has the extension AVD.

1.5.1.3 Copying text

The user can mark text in the Source window and transfer this to the Windows Clipboard by selecting Copy from the Edit menu.

1.5.1.4 Downloading configuration (Emulator only)

When no files are loaded into AVR Studio, the File menu contains the option Download Configuration. If an Emulator is connected to the computer, the following message appears.



Download Configuration

Warning: This function is for Emulator updates only. Downloading a random file using this function will result in a non-functional Emulator. This function must onl be used with files supplied by Atmel Corporation for this specific purpose.

As can be seen from the warning, downloading configurations should only be done if instructed so by Atmel. The files to be downloaded holds configuration files for the Emulator, and all other files will result in a non-functional Emulator.

1.5.2 Execution Control

Execution commands are used for controlling the execution of a program. All execution commands are available through menus, shortcuts and the Debug toolbar.

1.5.2.1 Go

The Go 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 Go command is only available when the execution is stopped.

Shortcut: F5

1.5.2.2 Break

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.

Shortcut: CTRL-F5

1.5.2.3 Trace Into

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. After the Trace Into is completed, all information in all windows are updated.

Shortcut: F11

1.5.2.4 Step Over

The Step Over command in the Debug menu executes one instruction. If the instruction contains a function call/subroutine call, the function/subroutine is executed as well. 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.

Shortcut: F10

1.5.2.5 Step Out

The Step Out command in the Debug menu executes until the current function has completed. 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.

Shortcut: SHIFT+F11

1.5.2.6 Run to Cursor

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.

Shortcut: F7

1.5.2.7 Reset

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.

Shortcut: SHIFT+F5

1.5.3 Watches

When executing at C source level, the Watch window can be used for watching symbols. When executing object files generated by the Atmel *AVR* Assembler, no symbol information is present so the Watch window can not be used for displaying any information.

1.5.3.1 Adding watches

In order to insert a new watch, the user must select Add Watch from the Watch window, or press the Add Watch button on the Debug toolbar. If the Watch window is not present when the Add Watch command is given, the Watch window is created, and already defined watches are reinserted (if any).

If the Watch window is the active window, a new watch can also be added by pressing the INS key.

1.5.3.2 Deleting watches

The user can delete a watch by first marking the symbol to be deleted in the Watch window and then give a Delete Watch command from the Watch menu or from the Debug toolbar. Selecting a watch is done by moving the mouse pointer to the name of the watch and pressing the left mouse button.

If the Watch window is the active window, a marked symbol can also be deleted by pressing the DEL key.

1.5.3.3 Deleting all watches

The Delete all watches command is available from the Watch menu. When this command is issued, all defined watches are removed from the Watch window.

1.5.4 Breakpoints

The user can set an unlimited number of code breakpoints. The breakpoints are remembered between sessions unless a new object file has been generated. If the object file is newer than the project file, the breakpoints are discarded.

When a breakpoint is set on a location, the breakpoint is indicated by a dot on the left side of the instruction.

1.5.4.1 Toggle Breakpoint

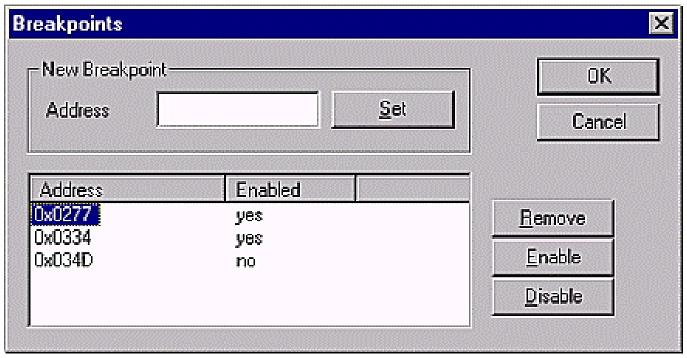
The Toggle Breakpoint command toggles the breakpoint status for the instruction where the cursor is placed. Note that this function is only available when the source view is the active view.

1.5.4.2 Clear all breakpoints

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

1.5.4.3 Show list

When Show list is selected, the following dialog appears on the screen:

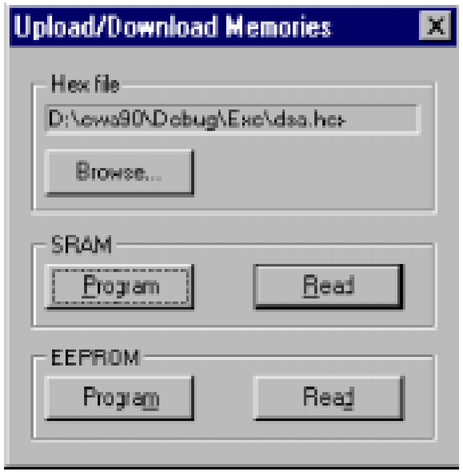


Address	Enabled
0x0277	yes
0x0334	yes
0x034D	no

In the Breakpoints dialog, the user can inspect existing breakpoints, add a new breakpoint, remove a breakpoint or enable/disable breakpoints.

1.5.5 Up/Download Memories

The user can download data to the SRAM and the EEPROM data memories. If the Up/Download Memories is selected from the File menu after a file has been loaded into AVR Studio, the following window appears on the screen:



Hex file: D:\cws90\Debug\Exe\dsa.hex

The Up/Download Memories function reads and writes Intel-Hex files. Use the Browse button to select the file to read from/write to. Then select the desired function.

1.5.6 Toolbars

AVR Studio contains three different toolbars described below. The toolbars can be individually removed and/or reinserted if desired by unchecking/checking them in the View → Toolbars menu.

1.5.6.1 The General toolbar

The General toolbar contains buttons for standard Windows commands. The General toolbar has the following buttons:

Open File	Copy	Help

1.5.6.2 The Debug toolbar

The Debug toolbar contains buttons for execution control and Watch window control. The Debug toolbar has the following buttons:

Add Watch	Delete Watch	Go
Stop	Trace Info	Step Over
Step Out	Run to Cursor	Toggle Breakpoints
Clear all Breakpoints	Reset	

1.5.6.3 The Views toolbar

The Views toolbar contains buttons for enabling and disabling the most commonly used windows and for adding Memory windows. The Views toolbar has the following buttons:

Toggle Watch window	Toggle Register window
Add Memory window	Toggle Processor window
Toggle Message window	

1.5.7 Shortcut summary

The following shortcuts are defined in *AVR* Studio:

Command	Shortcut
Toggle Register window	Alt+0
Toggle Watch window	Alt+1
Toggle Message window	Alt+2
Toggle Processor window	Alt+3
Add Memory window	Alt+4
Show Breakpoints List	Ctrl+B
Copy to Clipboard	Ctrl+C
Open File	Ctrl+O
Help	F1
Run	F5
Break	Ctrl+F5
Reset	Shift+F5
Run to Cursor	F7
Toggle Breakpoint	F9
Step Over	F10
Trace Into	F11
Step Out	Shift+F11

1.6 Execution Target

AVR Studio can be targeted towards an *AVR* In-Circuit Emulator or the built-in *AVR* Simulator. When the user opens a file, *AVR* Studio automatically detects whether an Emulator is present and available on one of the systems serial ports. If an Emulator is found, it is selected as the execution target. If no Emulator is found, execution will be done on the built-in *AVR* Simulator instead.

The Status bar will indicate whether execution is targeted at the *AVR* In-Circuit Emulator or the built-in *AVR* Simulator.

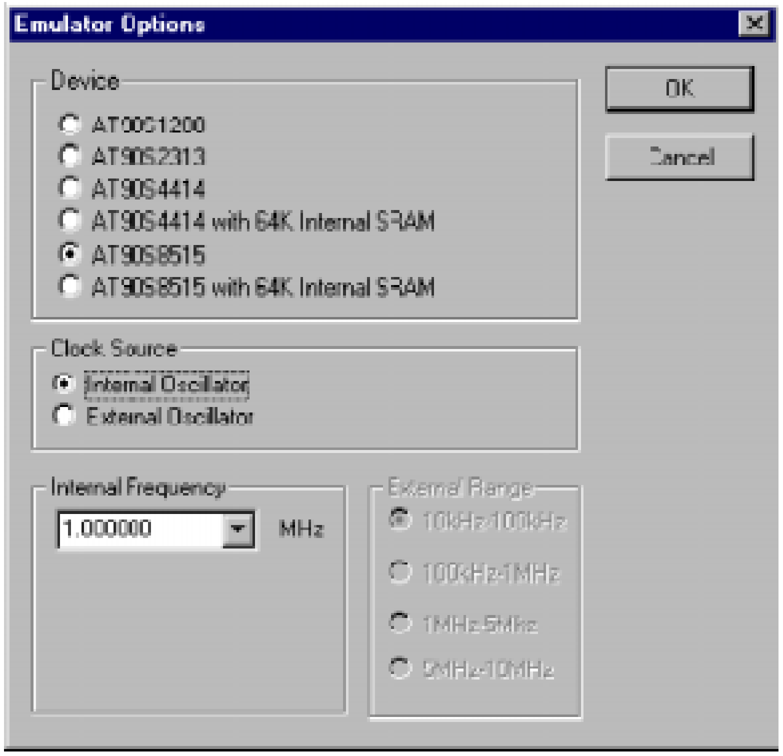
1.6.1 *AVR* In-Circuit Emulator

If an *AVR* In-Circuit Emulator is available in the system, it is automatically selected as the execution target. The Emulator must be connected through a serial port. If an Emulator is present in the system but can not be identified, close the file, reset the Emulator and try once more. For information regarding the *AVR* In-Circuit Emulator, see the AVR AT90ICE1200 User Guide or the AVR AT90ICEPRO User Guide..

If the user wants to use the Simulator, even if an Emulator is present in the system, disconnect or switch off the Emulator before opening a file.

1.6.1.1 Emulator Options

The Emulator Options dialog enables controlling the configuration and speed of the Emulator. When Options → Emulator is selected from the Options menu, the following window appears:



Device
- AT90S1200
- AT90S2313
- AT90S4414
- AT90S4414 with 64K Internal SRAM
- [✓ AT90S8515]
- AT90S8515 with 64K Internal SRAM

Clock Source
- [✓ Internal Oscillator]
- External Oscillator

Internal Frequency: 1.000000 MHz

External Range
- [✓ 10kHz-100kHz]
- 100kHz-1MHz
- 1MHz-5MHz
- 5MHz-10MHz

The window also automatically appears the first time a file is opened in AVR Studio. If the Emulator is an ICEPRO, the user can select which device the Emulator should be set up as. If the Emulator is an ICE1200, the device selection is not available.

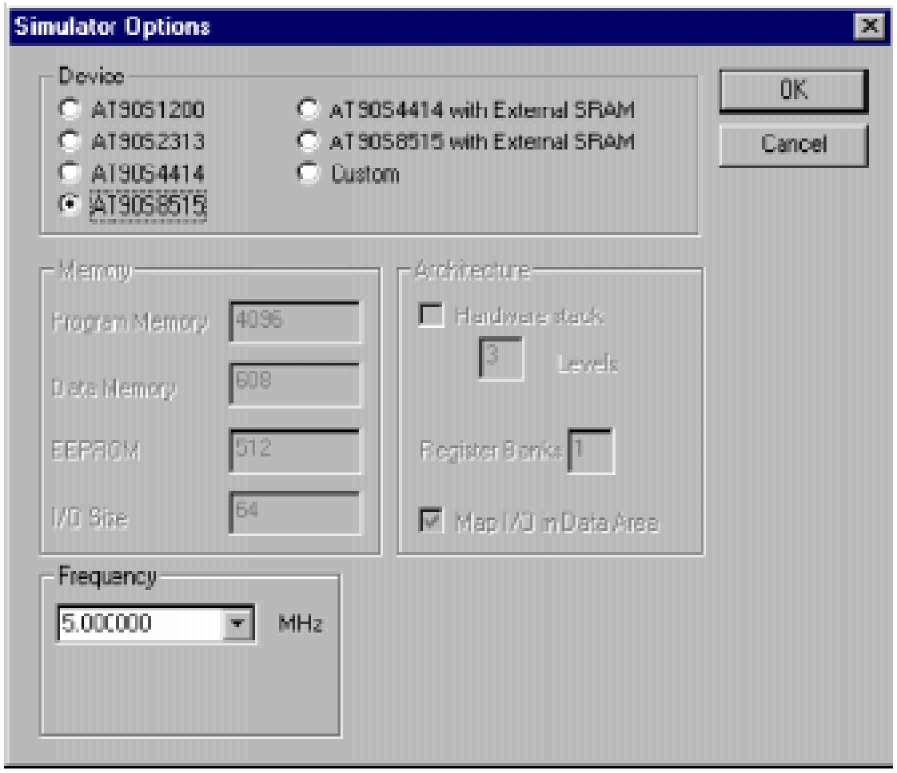
The user can select whether the Emulator should be clocked from the on-board programmable clock circuit, or if it should be clocked from an external source (see AVR In-Circuit Emulator documentation for further information).

If the Internal Oscillator is set as clock source, the user can select a frequency between 400kHz and 20MHz. The user can either select typical frequencies from the list, or enter a custom frequency. Note that not all frequencies can be exactly generated. The actual frequency is printed in the Message window. The speed of the Emulator is remembered between sessions.

If the user wants to use an external oscillator, the user must also supply information about the frequency range of the external oscillator by clicking the appropriate button in the External range box..

1.6.2 *AVR* Instruction Set Simulator

If AVR Studio does not succeed in identifying an Emulator, it selects a built-in Simulator instead. The Simulator has a number of different options. If Simulator → Options is selected from the Options menu, the following dialog appears on the screen:



Device: ( ) AT90S1200 ( ) AT90S2313 ( ) AT90S4414 [✓ AT90S8515] ( ) AT90S4414 with External SRAM ( ) AT90S8515 with External SRAM ( ) Custom

Program Memory 4096
Data Memory 608
EEPROM 512
I/O Size 64
Hardware stack ☐ 3 Levels
Register Banks 1
☑ Map I/O in Data Area
Frequency 5.000000 MHz

The window also automatically appears the first time a file is opened in AVR Studio. Changing any of the values (except the Frequency) will force AVR Studio to issue a Reset command.

1.6.2.1 Device Selection

In the device box, the user can select between 6 different standard configurations. If one of the standard configurations is selected, the memory sizes and architectural details are filled out accordingly. The user also has the possibility to define a custom configuration.

1.6.2.2 Custom Selection

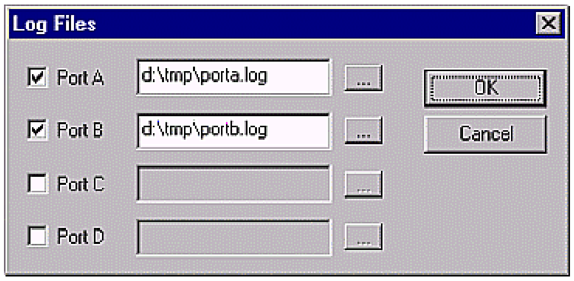
If the Custom button is pressed, the user is allowed to enter values in the Memory and the Architecture fields.

- The Program memory size is counted in Words. A maximal value of 65536 (64K Words) can be entered. If a program tries to access program memory outside the selected range, operation is undefined.
- The Data memory size is counted in Bytes. A maximal value of 65536 (64K Bytes) can be entered. If the user tries to access an SRAM location outside the selected range, operation is undefined. The value in the Data memory field is the highest address which can be used for addressing the SRAM. The size of the Register file and the I/O area (if mapped in the SRAM area) must be added to the SRAM size in order to get the correct value in this field.
- The EEPROM size is counted in Bytes. A maximal value of 65536 (64K Bytes) can be entered. The EEPROM address register only contains as many bits as is necessary for accessing the requested sized EEPROM.
- The I/O size is counted in bytes. Allowed values of I/O size are: 64, 128 and 256.

- The user can select whether the Simulator should use a hardware stack or not. If a hardware stack is selected, the user can set the number of levels in the hardware stack.
- The user can not set the Simulator to handle several register files.
- The user can select whether the I/O area locations should be addressable in the SRAM address space or not. If the I/O area is addressable in the SRAM area, it will be accessible on the addresses 0x20 and upwards.

1.6.2.3 Logging Ports

The user can log the activity on the output ports. If Simulator Port Logging is selected from the Options menu, the following dialog appears on the screen:



[✓] Port A d:\tmp\porta.log
[✓] Port B d:\tmp\portb.log
[ ] Port C
[ ] Port D

The user must select which of the Ports to log. If a Port is selected, the user must also select the file 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. The Cycle field is in decimal format and the Data field is in hexadecimal format. 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.

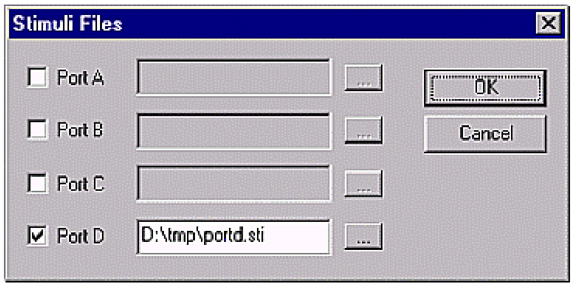
Example on a log file produced by *AVR* Studio:

```
000000000:00

000000043:FF

000000080:FE

000000090:AB

000001021:FF
```

In this example, FF was written to the port at cycle 43, then FE was written at cycle 80 etc.

1.6.2.4 External Stimuli

The user can set values on the ports. If Simulator Port Stimuli is selected from the Options menu, the following dialog appears on the screen:



[ ] Port A
[ ] Port B
[ ] Port C
[✓] Port D D:\tmp\portd.sti

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. Note that only pins set as inputs are affected.

1.6.2.5 Timer/Counter0

If a standard device is selected, the Timer/Counter0 overflow interrupt vector and the external counter pin is set accordingly. If a custom device is selected, the Timer/Counter0 overflow interrupt vector and the external counter pin is set as for AT90S1200.

1.6.2.6 Timer/Counter 1

If a standard device is selected, and the device supports the Timer/Counter 1, the Timer/Counter 1 interrupt vectors, external counter pin, input capture pin and output pins are set accordingly. For the devices AT90S8515 and AT90S4414, the ICP function is set up on PD4 and the OC1B function is set up on PD6. If a custom device is selected, the Timer/Counter 1 will not be present.

1.6.2.7 SPI

If a standard device is selected, and the device supports the SPI, the SPI interrupt vector and the SCK/MISO/MOSI/SS pins are set up accordingly. If a custom device is selected, the SPI will not be present.

1.6.2.8 UART

If a standard device is selected, and the device supports the UART, the UART interrupt vectors and the Receive/Transmit pins are set up accordingly. If a custom device is selected, the UART will not be present.

1.6.2.9 External Interrupts

If a standard device is selected, the external interrupts are set up accordingly. If a custom device is selected, there will be one external interrupt available as for AT90S1200.