AVR Development Tools

This section describes some of the development tools that are available for the 8-bit *AVR* family.

- **Atmel AVR Assembler**
- **Atmel AVR Simulator**
- **IAR ANSI C-Compiler, Assembler, Linker, Librarian & Debugger**
- **Atmel In-Circuit Emulator (ICE)**

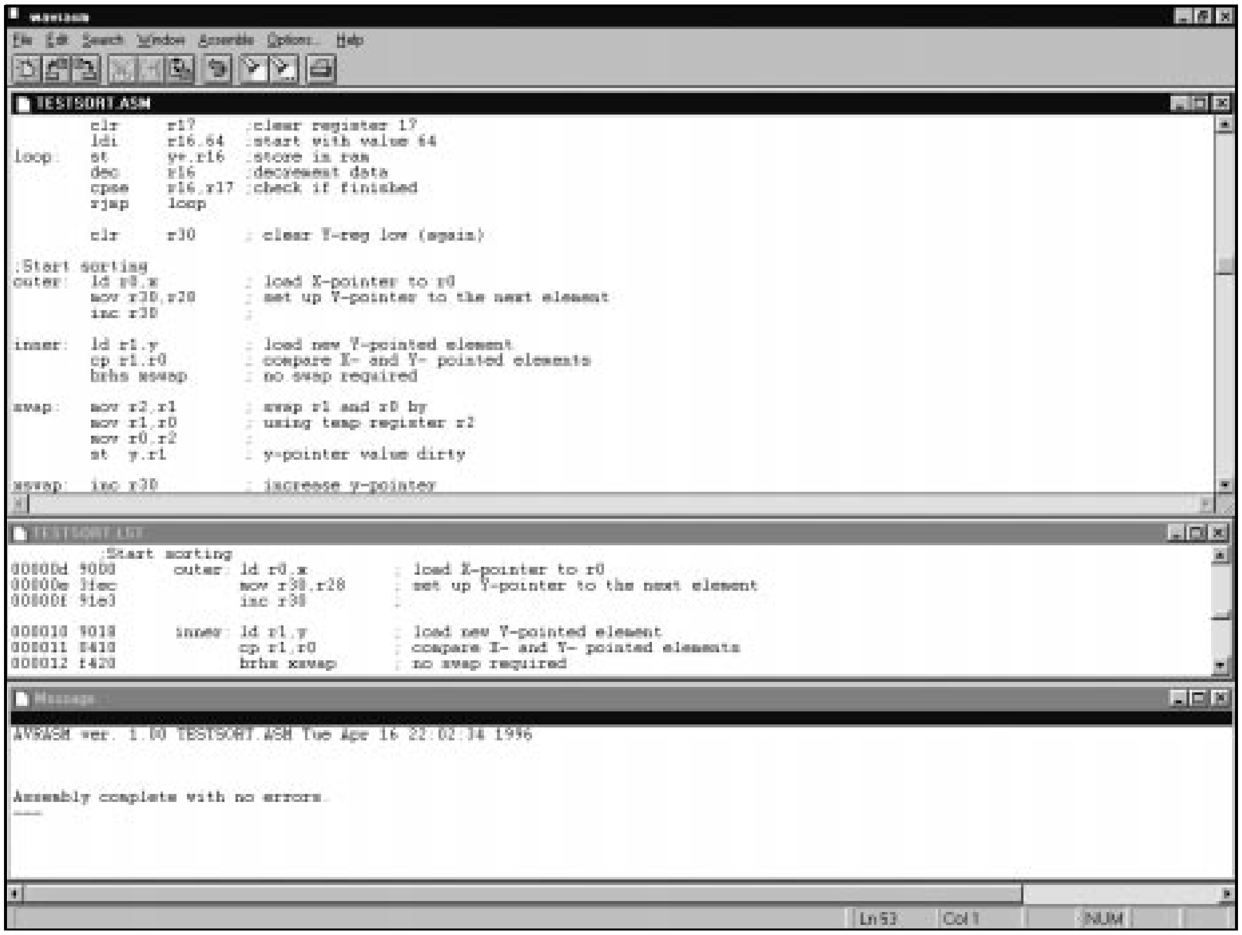
There are a lot of development tools under development, please contact Atmel for more details.

8-Bit AVR® Development Tools

Atmel *AVR* Assembler

Features:

- Translates Assembler source programs into object code
- Extremely fast assembling
- Supports all the microcontrollers in the AT90S family
- Powerful Macro capabilities
- Supports all standard output formats
- Easy to use MS-Windows interface
- Editor included in MS-Windows version
- Jump to next/previous error
- Also available in MS-DOS command line version



```
        clr     r17     ;clear register 17
        ldi     r16,64  ;start with value 64
loop:   st      y+,r16  ;store in ram
        dec     r16     ;decrement data
        cpse    r16,r17 ;check if finished
        rjmp    loop

        clr     r30     ; clear Y-reg low (again)

;Start sorting
outer:  ld r0,x         ; load X-pointer to r0
        mov r30,r28     ; set up Y-pointer to the next element
        inc r30         ;

inner:  ld r1,y         ; load new Y-pointed element
        cp r1,r0        ; compare X- and Y- pointed elements
        brhs xswap      ; no swap required

swap:   mov r2,r1       ; swap r1 and r0 by
        mov r1,r0       ; using temp register r2
        mov r0,r2       ;
        st  y,r1        ; y-pointer value dirty

xswap:  inc r30         ; increase y-pointer
```

```
           ;Start sorting
00000d 9000      outer: ld r0,x         ; load X-pointer to r0
00000e 2fec             mov r30,r28     ; set up Y-pointer to the next element
00000f 95e3             inc r30         ;

000010 9018      inner: ld r1,y         ; load new Y-pointed element
000011 1410             cp r1,r0        ; compare X- and Y- pointed elements
000012 f420             brhs xswap      ; no swap required
```

```
AVRASM ver. 1.00 TESTSORT.ASM Tue Apr 16 22:02:34 1996


Assembly complete with no errors.
```

Powerful Macro Capabilities

The Assembler contains powerful macro capabilities, enabling the user to build a virtual instruction set which is structures of ordinary *AVR* instructions. For example, this macro does a 16 bit subtraction:

```
;
; SUB16 macro definition
; The macro subtracts a 16 bit constant from a register
; pair. A call to the macro is done by
; SUB16 Regh,Regl,Const
;
.MACRO SUB16            ; Macro name
  subi$1,low($2)        ; subtract low byte
  sbci $0,high($2)      ; subtract high byte
.ENDMACRO

; ...

; Call the macro
  ldi   r16,low(0x3400)  ; set values in registers
  ldi   r17,low(0x3400)  ;
  SUB16 r17,r16,0x23a0   ; compute 0x3400-0x23a0
```

Assembly Directives

The assembler supports a number of directives making the application development easier. In addition to the directives for macro generation and control, the assembler contains directives for:

- Including files. Included files can be nested.
- Set program origin.
- Symbol usage. The user can define symbols and labels and refer to these throughout the assembly program.
- Constant data initialization. The user can do constant initialization. Constants will be placed in the Flash program memory.
- List file control.
- Support of expressions in a C-like syntax.

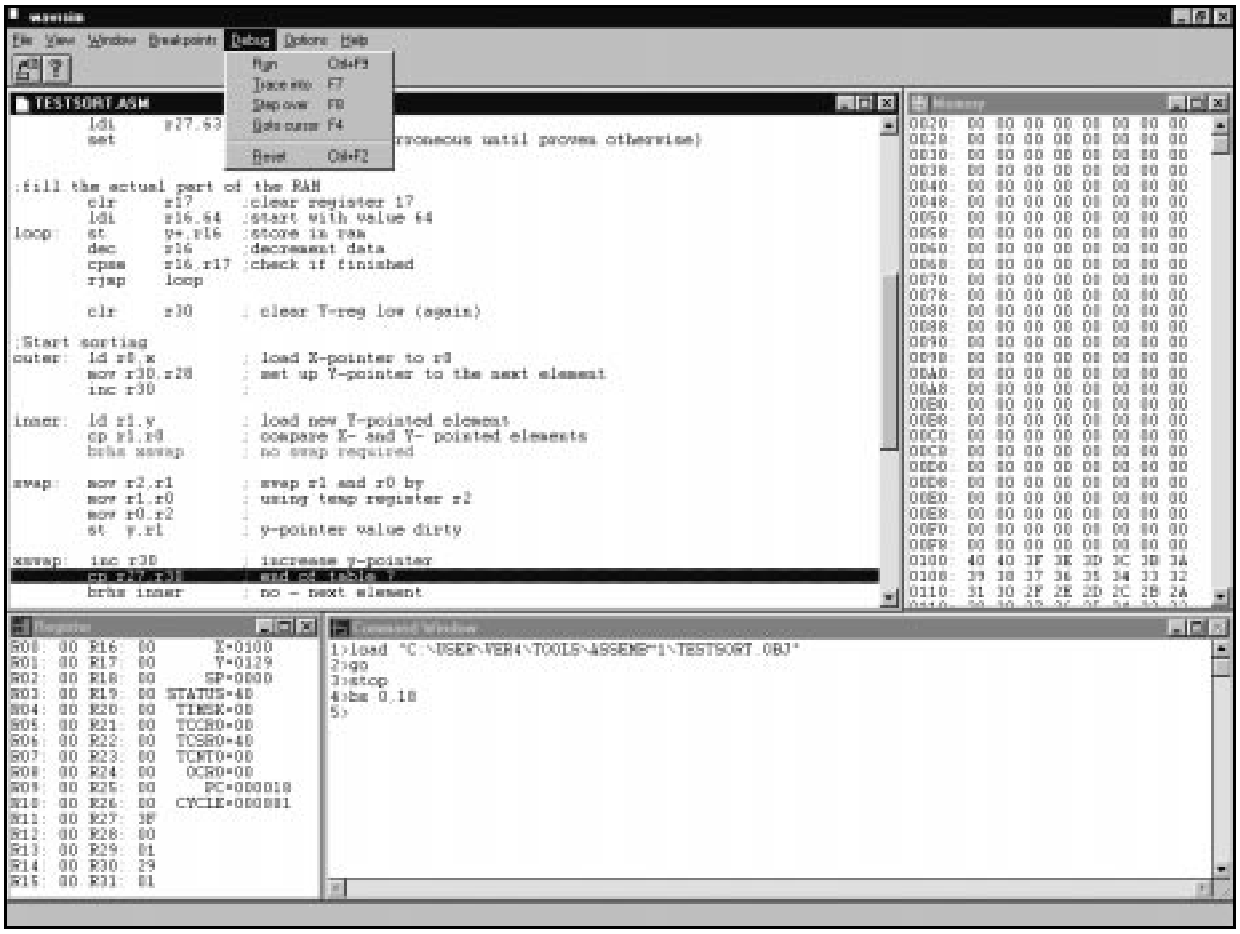
MS Windows Application

The assembler executes under the Windows environment. The Windows version can be executed under Windows 3.11, Windows 95 and Windows NT. The Windows version includes a full editor for writing assembly programs. An MS-DOS command line version is also available.

Atmel *AVR* Simulator

Features:

- Supports the whole range of AT90S microcontrollers
- Assembly source level simulation
- Powerful debugging facilities
- Full support of *AVR* peripheral devices
- Easy to use MS-Windows interface



```
        ldi     r27,63
        set                      rroneous until proven otherwise)

;fill the actual part of the RAM
        clr     r17     ;clear register 17
        ldi     r16,64  ;start with value 64
loop:   st      y+,r16  ;store in ram
        dec     r16     ;decrement data
        cpse    r16,r17 ;check if finished
        rjmp    loop

        clr     r30      ; clear Y-reg low (again)

;Start sorting
outer:  ld r0,x          ; load X-pointer to r0
        mov r30,r28      ; set up Y-pointer to the next element
        inc r30          ;

inner:  ld r1,y          ; load new Y-pointed element
        cp r1,r0         ; compare X- and Y- pointed elements
        brhs xswap       ; no swap required

swap:   mov r2,r1        ; swap r1 and r0 by
        mov r1,r0        ; using temp register r2
        mov r0,r2        ;
        st  y,r1         ; y-pointer value dirty

xswap:  inc r30          ; increase y-pointer
        cp r27,r30       ; end of table ?
        brhs inner       ; no - next element
```

```
0020:  00 00 00 00 00 00 00 00
0028:  00 00 00 00 00 00 00 00
0030:  00 00 00 00 00 00 00 00
0038:  00 00 00 00 00 00 00 00
0040:  00 00 00 00 00 00 00 00
0048:  00 00 00 00 00 00 00 00
0050:  00 00 00 00 00 00 00 00
0058:  00 00 00 00 00 00 00 00
0060:  00 00 00 00 00 00 00 00
0068:  00 00 00 00 00 00 00 00
0070:  00 00 00 00 00 00 00 00
0078:  00 00 00 00 00 00 00 00
0080:  00 00 00 00 00 00 00 00
0088:  00 00 00 00 00 00 00 00
0090:  00 00 00 00 00 00 00 00
0098:  00 00 00 00 00 00 00 00
00A0:  00 00 00 00 00 00 00 00
00A8:  00 00 00 00 00 00 00 00
00B0:  00 00 00 00 00 00 00 00
00B8:  00 00 00 00 00 00 00 00
00C0:  00 00 00 00 00 00 00 00
00C8:  00 00 00 00 00 00 00 00
00D0:  00 00 00 00 00 00 00 00
00D8:  00 00 00 00 00 00 00 00
00E0:  00 00 00 00 00 00 00 00
00E8:  00 00 00 00 00 00 00 00
00F0:  00 00 00 00 00 00 00 00
00F8:  00 00 00 00 00 00 00 00
0100:  40 40 3F 3E 3D 3C 3B 3A
0108:  39 38 37 36 35 34 33 32
0110:  31 30 2F 2E 2D 2C 2B 2A
```

```
R00: 00 R16: 00       X=0100
R01: 00 R17: 00       Y=0129
R02: 00 R18: 00      SP=0000
R03: 00 R19: 00  STATUS=40
R04: 00 R20: 00   TIMSK=00
R05: 00 R21: 00   TCCR0=00
R06: 00 R22: 00   TCSR0=40
R07: 00 R23: 00   TCNT0=00
R08: 00 R24: 00    OCR0=00
R09: 00 R25: 00      PC=000018
R10: 00 R26: 00   CYCLE=000081
R11: 00 R27: 3F
R12: 00 R28: 00
R13: 00 R29: 01
R14: 00 R30: 29
R15: 00 R31: 01
```

```
1>load "C:\USER\VER4\TOOLS\ASSEMB~1\TESTSORT.OBJ"
2>go
3>stop
4>bs 0,18
5>
```

Debugging Facilities

The simulator has a number of functions to help the programmer to debug programs including:

- Breakpoints: Set up to 256 breakpoints in the source window, and program execution will halt upon reaching one of the breakpoints.
- Single stepping: Step through the code instruction by instruction and watch the execution.
- Step into/Trace over: Select whether calls should be traced, or if these simulation details should be omitted.
- Goto cursor: Place the cursor on an instruction, and the simulator will execute until the marked instruction is reached.
- Run from file. The user can write scripts consisting of simulator commands.

- Display of registers and memory. The user can view all memory spaces, all general purpose working registers and the registers in the I/O map. The user also has the ability to write values in these memories and registers.
- Logging. All information, including register contents, memory contents and I/O accesses can be logged for each instruction.
- The simulator holds control on the number of clock cycles elapsed.

All commands are available through a command window and through menus.

Simulation of Peripherals

- The simulator supports the peripheral devices of the *AVR* microcontrollers, including:
- Interrupts. Each interrupt can be set in each cycle enabling complex combinations including nested interrupts.
- Timer/Counters. The timer/counters can also be simulated. This of course includes generating interrupts on overflows and compare matches. Free running mode is also supported.
- I/O ports. The I/O ports are implemented, giving the user the ability to communicate with the simulated programs through the ports.

Together with the powerful control mechanisms present, this makes the simulator a complete debugging tool for the *AVR* family of Enhanced RISC Microcontrollers.

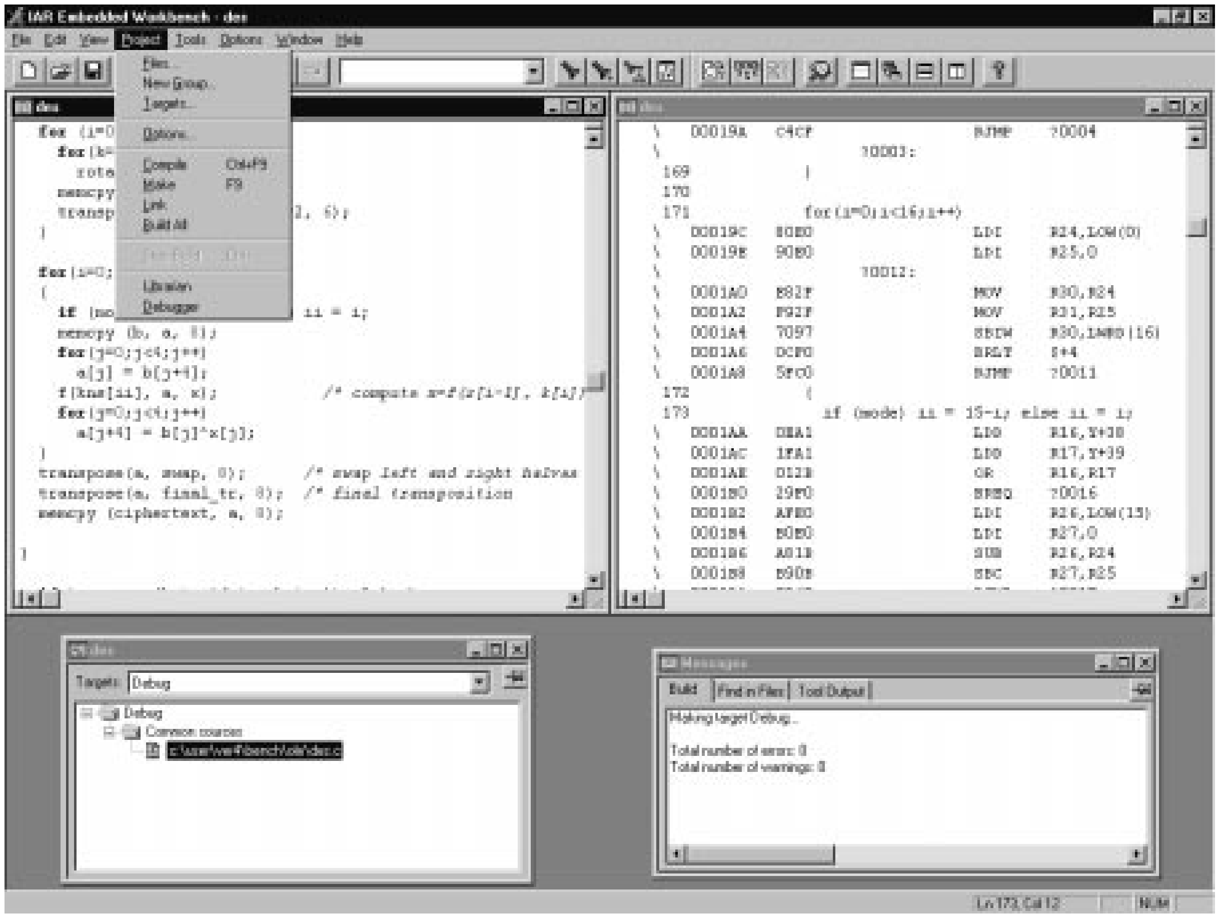
MS Windows Application

The simulator is developed for execution under the Windows environment. The Windows version can be executed under Windows 3.11, Windows 95 and Windows NT.

IAR Development Tools for Atmel *AVR*

Features:

- Fully ANSI Compatible C Compiler
- Includes Embedded Workbench
- C and Assembler level language debugger
- Runs under DOS, Windows 3.11, Windows 95 and Windows NT



```
  for (i=0
    for (b=
      rota
    memcpy
    transp                    2, 6);
  }

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

}
```

```
  \    00019A    C4CF                  RJMP    ?0004
  \                      ?0003:
  169              }
  170
  171              for(i=0;i<16;i++)
  \    00019C    80E0                  LDI     R24,LOW(0)
  \    00019E    90E0                  LDI     R25,0
  \                      ?0012:
  \    0001A0    E82F                  MOV     R30,R24
  \    0001A2    F92F                  MOV     R31,R25
  \    0001A4    7097                  SBIW    R30,LWRD(16)
  \    0001A6    DCF0                  BRLT    $+4
  \    0001A8    5Fc0                  RJMP    ?0011
  172              {
  173                if (mode) ii = 15-i; else ii = i;
  \    0001AA    D8A1                  LDD     R16,Y+38
  \    0001AC    1FA1                  LDD     R17,Y+39
  \    0001AE    D12B                  OR      R16,R17
  \    0001B0    29F0                  BREQ    ?0016
  \    0001B2    AFE0                  LDI     R26,LOW(15)
  \    0001B4    B0E0                  LDI     R27,0
  \    0001B6    A81B                  SUB     R26,R24
  \    0001B8    B90B                  SBC     R27,R25
```

```
Making target Debug...

Total number of errors: 0
Total number of warnings: 0
```

Embedded Workbench

The Embedded Workbench offers a total integration of C compiler, editor, linker, librarian, assembler, editor and debugger in a seamless environment. The Embedded Workbench includes the following features:

- Flexible editor. The editor offers flexibility in terms of customizable toolbar and user defined shortcuts. The editor implements the basic Windows editing commands as well as extensions for C programming, such as C syntax coloring and direct jump to context from error listing.
- The hierarchical project maintenance facility makes it possible to have several targets, such as a release target and a debug target with different option settings. Each target is built from one or several groups which in turn are built from one or several files. Compiler options can be set on each level.

- The Make system automatically generates a dependency list of output files, source files and include files. This allows the Make system to only compile, recompile or reassemble the updated parts of the source code, which speeds up the building process.
- Extensive on-line help function makes it easy to quickly find specific help about the tool without leaving the Embedded Workbench.
- The compiler is also available under MS-DOS in a mouse controlled menu driven user interface, allowing all development steps to be performed in an integrated DOS environment.

C Compiler

The C Compiler is an optimizing ANSI C compatible C compiler. The C compiler is the core product in the Embedded Workbench. All data types required by ANSI are supported. Full ANSI C compatibility also means that the compiler conforms to all requirements placed by ANSI on run-time behavior. The C Compiler includes the following features:

- Fully compatible with the ANSI C standard
- Fully reentrant code
- Absolute read/write of I/O locations at C level
- Built-in AT90S specific Optimizer
- AT90S specific extensions
- Full floating point support
- Four different memory models to allow best fit selection

Assembler

The C compiler kit comes with a relocatable structured assembler. This provides the option of coding time critical sections of the application in assembly without loosing the advantages of the C language. The preprocessor of the C language is incorporated in the assembler, thus allowing use of the full ANSI C macro language, with conditional assembly, macro definitions, if statements and more. C include files can also be used in an assembly program. All modules written in assembly can easily be accessed from C and vice-versa, making the interface between C and assembly a straightforward process.

Linker

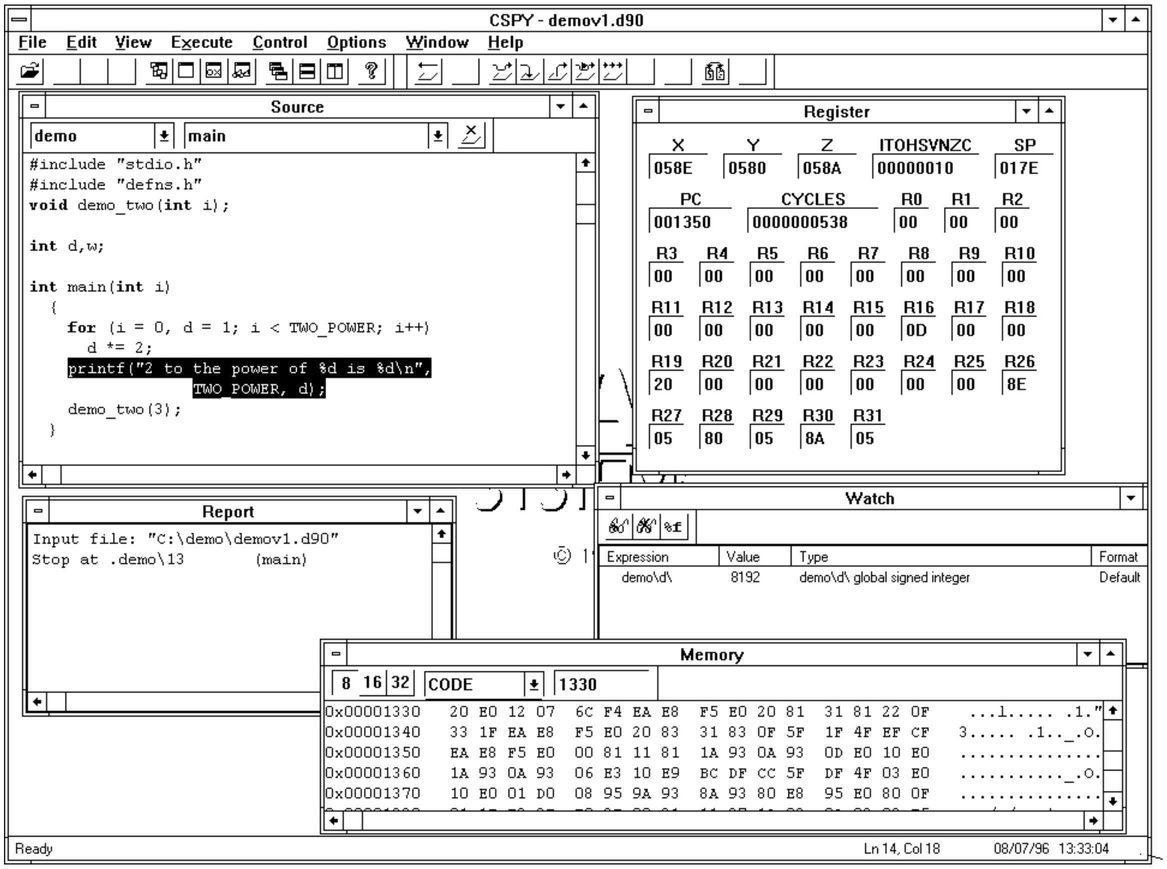
The linker XLINK supports complete linking, relocation and format generation to produce AT90S code. Over 30 output formats are supported. Detailed cross reference and map listing with segments, symbol information, variable locations and function addresses are easily generated.

Librarian

The librarian XLIB creates and maintains libraries and library modules. Listings of modules, entry points and symbolic information contained in every library are easily generated. XLIB can also give the library attribute for conditional loading or the program attribute for unconditional loading.

C-SPY High Level Language Debugger and Simulator

The C-SPY high level language debugger/simulator combines the detailed control code execution needed for embedded development debugging with the flexibility and power of the C language. The source window can display C source and mix it with assembly source. No extra hardware is needed since instruction execution is simulated.



```
#include "stdio.h"
#include "defns.h"
void demo_two(int i);

int d,w;

int main(int i)
  {
    for (i = 0, d = 1; i < TWO_POWER; i++)
      d *= 2;
    printf("2 to the power of %d is %d\n",
                 TWO_POWER, d);
    demo_two(3);
  }
```

X	Y	Z	ITOHSVNZC	SP			
058E	0580	058A	00000010	017E			
PC	CYCLES	R0	R1	R2			
001350	0000000538	00	00	00			
R3	R4	R5	R6	R7	R8	R9	R10
00	00	00	00	00	00	00	00
R11	R12	R13	R14	R15	R16	R17	R18
00	00	00	00	00	0D	00	00
R19	R20	R21	R22	R23	R24	R25	R26
20	00	00	00	00	00	00	8E
R27	R28	R29	R30	R31			
05	80	05	8A	05			

```
Input file: "C:\demo\demov1.d90"
Stop at .demo\13        (main)
```

Expression	Value	Type	Format
demo\d\	8192	demo\d\ global signed integer	Default

8 16 32 CODE 1330

```
0x00001330   20 E0 12 07  6C F4 EA E8  F5 E0 20 81  31 81 22 0F   ...l..... .1."
0x00001340   33 1F EA E8  F5 E0 20 83  31 83 0F 5F  1F 4F EF CF  3..... .1.._.O.
0x00001350   EA E8 F5 E0  00 81 11 81  1A 93 0A 93  0D E0 10 E0  ..............
0x00001360   1A 93 0A 93  06 E3 10 E9  BC DF CC 5F  DF 4F 03 E0  ..........._.O.
0x00001370   10 E0 01 D0  08 95 9A 93  8A 93 80 E8  95 E0 80 0F  ..............
```

Ready   Ln 14, Col 18   08/07/96 13:33:04

C-SPY includes the following features:

- Powerful breakpoint setting. C-SPY has an unlimited number of breakpoints. Breakpoints can be set on C statements, assembler instructions, and on any address with access type of read, write and opcode fetch. It may also be set as a combination of these. After triggering a breakpoint, a macro command can be executed.
- C-like macro language. A powerful C-like macro language can tailor the environment used for debugging in the C-SPY, including system macros for host file I/O simulation, reset, start up and shut down, as well as statements such as for loops, while loops, if and return statements.
- Interrupt simulation implements commands to launch specific interrupts at a specific cycle count or periodically.
- I/O simulation. C-SPY terminal I/O emulation offers a console window for target system I/O. This unique feature is useful for debugging embedded applications when logical flows are of interest or the target is not yet ready.
- The watch points window makes it possible to watch any expression. The window itself will be updated whenever a breakpoint is gritted or a step is finished. Any variable can be modified during the execution by using specific C expressions.
- The source window for the assembler debugger displays the assembler instructions. It has a built-in assembler and disassembler function, menu and register window, and can evaluate assembler expressions.

Atmel *AVR* In-Circuit Emulator

Features:

- Supports the whole range of AT90S microcontrollers
- Full visibility of all MCU resources
- Powerful breakpoint facilities
- Extensive execution control
- 32K x 96 bit wide Trace Buffer for real-time data collection
- 32-bit Time Stamp generator
- 8-Bit Event memory for event generation
- Supports 3 download modes
- Real time emulation
- Software adjustable clock speed
- Supports assembler and C source level debugging
- Supports all on-chip peripherals
- Serial- and parallel port interfacing
- Includes simple programmer
- Integrated with other *AVR* development tools

Full visibility

Using the emulator, the status of all resources can be monitored, and most of them can also be modified:

- The register file (R/W)
- SRAM (R/W)
- Program memory (R/W)
- EEPROM (R/W)
- Program Counter (R/W)
- I/O locations (R/W)

Powerful breakpoint facilities

The emulator incorporates powerful breakpoint facilities including:

- SRAM address breakpoint: Break when a specified address in the SRAM is read or written.
- SRAM data breakpoint: Break when a specified value is written to or read from SRAM.
- SRAM address and data breakpoint: An SRAM address breakpoint can be combined with an SRAM data breakpoint.
- Program memory address breakpoint: Break when a specified program memory address is accessed.
- Program memory data breakpoint: Break when a specified value is read from the program memory.
- Register match breakpoint: Break when a specified value is read/written from/to one of the 32 registers.
- External trigger breakpoint: Break when an external signal is rising or falling.

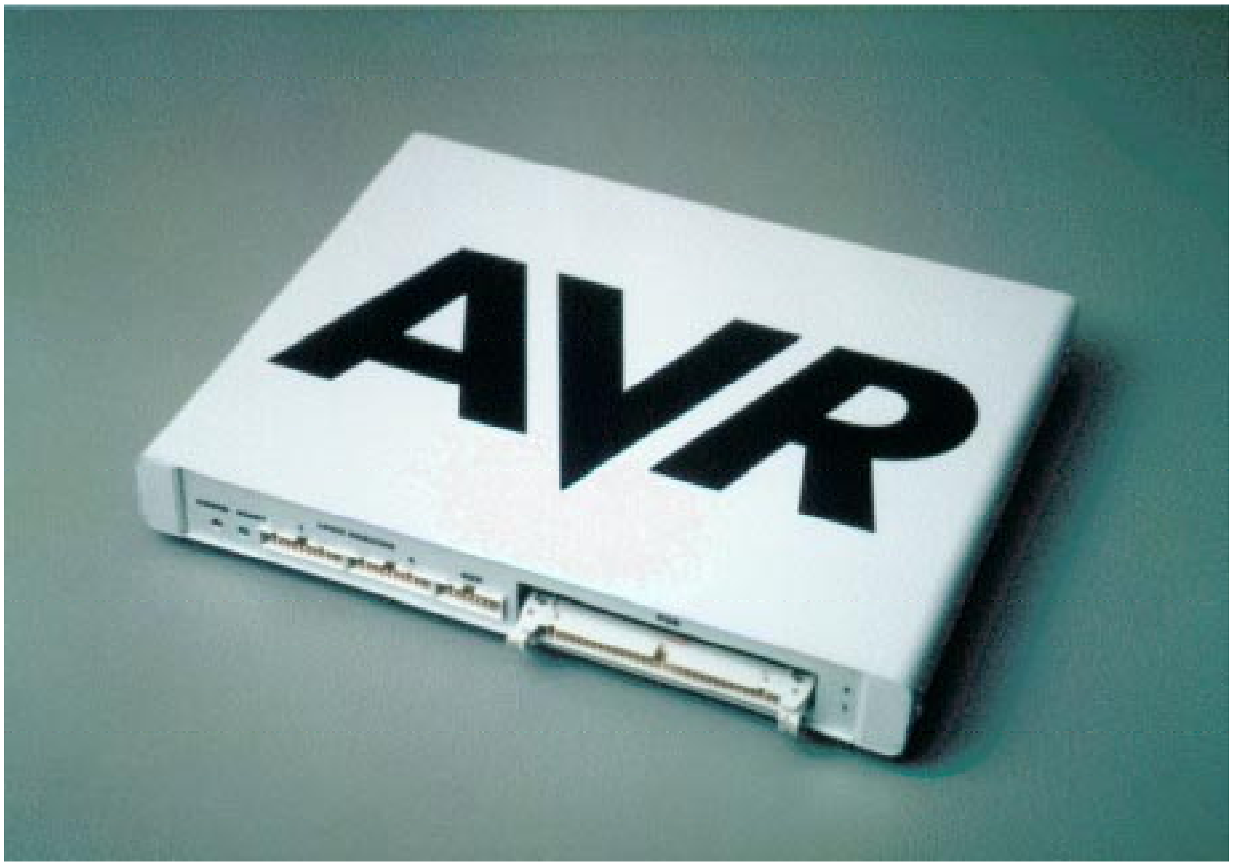
Extensive execution control

The emulator features extensive execution control:

- Single step execution: The emulator executes one instruction and then stops.
- Multiple step execution: The emulator executes a specified number of instructions and then stops.
- Software controlled Trace into/Step over.
- Start/Resume/Stop execution.
- Reset emulator.

Miscellaneous

- 5 trigger outputs are provided for connection to a DSO or a Logic Analyzer.
- Serial- and parallel port interface. The emulator can be connected to the PC through a standard serial- or parallel port.
- Simple programmer. A device present in the socket can be programmed from the PC or from the emulators program memory.
- In-circuit programming capabilities.
- Supports a wide range of download file formats like Intel Hex and Motorola S-Records.
- Fast download time.



Technical Specifications

System Unit

Physical Dimensions	(H x W x D) 32.4 x 277.1 x 218.6 mm / 1.3 x 10.8 x 8.5 in
Weight	400g / 0.88 lbs
Power Voltage Requirements	9 - 15 VDC
Power Consumption	< 20W
ICE Power Consumption	10W
Max. Application Power Consumption	5W
Ambient Temperature	-40 - +85°C (Operating) -55 - +85°C (Non-Operating)
Relative Humidity (Non-condensing)	10 - 90 % (Operating) 5 - 95 % (Non-Operating)
Shock	20 g, 11ms half sine
Vibration	5g

Connections

Power

Connector	5.5 mm OD/ 2.1mm ID Center Negative

Host

Serial Connector (RS-232)	9-pin D-SUB Female
Serial Communications Speed	9600 - 115,200 bits/s
Parallel Connector (LPT)	25-pin D-SUB Male
Parallel Communications Speed (Max)	80 kbyte/s

Pod

Emulating ≤ 40 pins	one 2 x 32 Male Header

External Trigger Inputs / Outputs

Connector	2 x 7 Male Header

Logic Analyzer Interface

Connectors	two 2 x 10 Male Headers

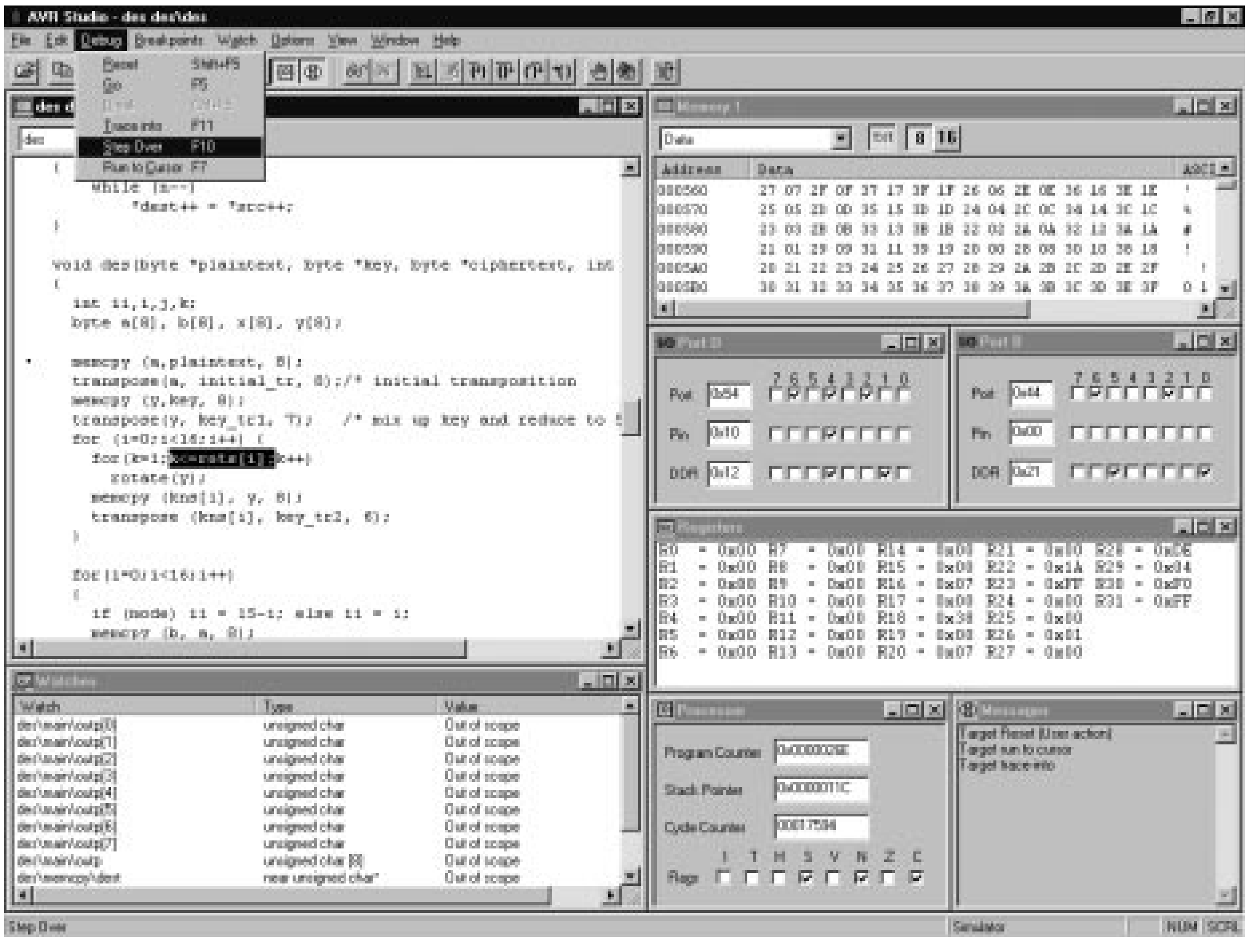
AVR Programmer Module Interface

Connector	2 x 5 Male Header

Atmel *AVR* Studio

Features:

- Supports the full range of AT90S microcontrollers
- Enables source level debugging of C and Assembly source code
- Allows for Assembly level debugging of C source code
- Interfaces the AVR In-Circuit Emulator for Real Time execution
- Built-in AVR instruction set Simulator allowing for debugging when no Emulator connected
- Watch for viewing and modifying symbols such as plain variables, structures and unions
- Full visibility of all MCU memories: Register file, SRAM, Program memory and EEPROM
- Exploits all features of the AVR In-Circuit Emulator
- Powerful breakpoint facilities
- Extensive execution control
- Supports IAR's ICCA90 C Compiler, IAR's AA90 Assembler and Atmel's AVR Assembler



```
  {
    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                                                ASCII
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
```

Port D

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

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  = 0x00 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 = 0x08 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

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

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