AVR001: Conditional Assembly and portability macros

Features

- **Increased portability**
- **Easier Code Writing**
- **Simplified I/O Register Access**
- **Improved Assembly Status Feedback**

1 Introduction

This application note describes the Conditional Assembly feature present in the AVR Assembler® version 1.74 and later. The AVR assembler is by default installed with AVR Studio®, available from the AVR section of the Atmel web site.

Examples of how to use Conditional Assembly are presented. One of the examples is a set of macros that will enable the software writer to write a generic code that will assemble to any AVR® without other modifications than changing the device definition file.

8-bit AVR® Microcontrollers

Application Note

Rev. 2550E-AVR-04/08

2 Theory of operation

Conditional Assembly (CA) was introduced in the AVR assembler in version 1.74. CA is based on a series of directives similar to the preprocessor directives available in C.

Since version 2.0, AVR assembler also has a preprocessor modeled after the C preprocessor. This allows for, e.g., C header files for devices to be used.

The difference between the regular preprocessor and the CA directives is that the latter are evaluated at compile time, and may therefore be used as a means of overloading macros. The regular preprocessor may be used for, e.g., configuration and selecting device dependent code, although this can be done with the CA directives as well. (This application note deals with CA directives only.)

Please see AVR Studio help on Assembler for a complete list of directives.

2.1 Conditional Assembly

The assembler evaluates a CA directive expression at compile-time and determines if the code enclosed by the CA directive is to be included or not.

A list of Conditional Assembly directives is presented in Table 2-1.

Table 2-1. Conditional Assembly Directives

Directive	Description
.ifdef <symbol>	Includes the code present between the .ifdef and the corresponding .else or .endif if the symbol is defined. Only symbols declared by .EQU or .SET are evaluated by .ifdef; symbols defined by the .def directives are treated differently by the assembler and cannot be used with .ifdef.
.ifndef <symbol>	Includes the code present between the .ifndef and the corresponding .else or .endif if the symbol is not defined. Only symbols declared by .EQU or .SET are evaluated by .ifndef; symbols defined by the .def directives are treated differently by the assembler and cannot be used with .ifndef.
.if <expression>	Includes the code present between the .if and the corresponding .else, .elif, or .endif if the expression evaluates to a value different from zero (true).
.elif <expression>	A .elif must be preceded by .if. It will include the code present between the .elif and the corresponding .else, .elif, or .endif if the expression evaluates to a value different from zero (true).
.else	A .else must be preceded by a .if or .elif. Will include the code present between the .else and the corresponding .endif if the expression specified for the corresponding .if or .elif evaluates to zero (false).
.endif	A .endif must be preceded by a .ifdef, .ifndef, .if, .else or .elif. The .endif defines the end of scope for the corresponding .ifdef, .ifndef, .if, .else or .elif.

The directives can be nested in up to 5 levels. CA directives can be combined with the .MACRO directive to make macros assemble differently depending on the enclosed CA directives. This is used in the example code for this application note.

2.2 Debugging Directives

When developing code it is useful with different types of feedback at assembly-time. For this purpose two new directives have been added: one that outputs a message to the message window if encountered and one that issues an error (accompanied with an error message). The debugging directives are described in Table 2-2.

Table 2-2. Debugging Directives

Directive	Description
.message "string"	If the .message directive is encountered when code is assembled, the "string" is written to the message window.
.error "string"	If the .error directive is encountered when code is assembled, an assembly error is issued and the "string" is written to the message window.

These directives can be combined with (e.g. AC) directives to provide information about the assembling status. The use of the .message directives is included in the code example for this application note. Please note that AVR Assembler 1.77.1 stops futher assembly when encountering the .error directive.

2.3 Example 1: General usage of CA

The reuse of code modules from one project to the next can often be a great time saver. Reusage of code modules will, without pre-processor directives or in this case CA directives, easily result in several different versions of the code module as it needs to be tailored for the different targets used. Using CA one version can be used for several target devices.

As an example, consider the initialization of the I/0 pins used for UART communication:

```
.EQU    ATmega128=1         ;Declares the symbol ATmega128
;EQU    ATmega16=1          ;Declares the symbol ATmega16
.EQU    UART =0             ;UART0 or UART1

.ifdef  ATmega128
 .message "UART Module assembled for ATmega128."
 .if UART == 0
   .message "UART0 used."
   sbi  DDRE, PE1           ;Configure TxD as output
 .elif UART == 1
   .message "UART0 used."
   sbi  DDRD, PD3           ;Configure TxD as output
 .else
   .error "UART number not specified"
 .endif
.elif ATmega16
 .message "UART Module assembled for ATmega128."
```

```
.if UART == 0
  .message "UART0 used."
  sbi  DDRD, PD1          ;Configure TxD as output
.else
  .error "UART number not specified"
.endif
.endif
```

As seen from the code example, a device specific initialization of (e.g. a UART) can be contained in one file. This allows for better control of the code modules reused.

2.4 Example 2: Conditional Assembly in Macros

The code example for this application note requires the AVR Assembler version 1.77 or later to assembled correctly. The AVR assembler is included with AVR Studio, which can be downloaded from the AVR section on the Atmel web site.

The file "macros.inc" includes a number of macros to ease bit and byte access in the I/0 and data space. The macros are listed with comments in Table 2-3.

Table 2-3. Macros Defined in the File “macros.inc”

Macro name	Description
SETB [Address, Bit Mask, Register]	"Set Bit" - Set a bit in any location in the I/0 space. Registers that can be used are R16-R31.
CLRB [Address, Bit Mask, Register]	"Clear Bit" - Clear a bit in any location in the I/0 space. Registers that can be used are R16-R31.
SKBS [Address, Bit Mask, Register]	"Skip if Bit Set" - skip the instruction following the macro if the bit specified by Bit Mask in any location in the I/0 space is set.
SKBC [Address, Bit Mask, Register]	"Skip if Bit Cleared" - skip the instruction following the macro if the bit specified by Bit Mask in any location in the I/0 space is cleared.
STORE [Address, Register]	"Store register" - Stores the contents of a register in a location in any location in the I/0 space.
LOAD [Register, Address]	"Load register" - Load a register with the contents from any location in the I/0 space.

The reason for using these macros to access I/0 space (and extended I/0 space) is that the code writer need not consider where in the I/0 space the accessed registers are located. This would be required if the macros were not used as not all instructions reach all addresses in the I/0 space. The advantages are therefore numerous:

- The author doesn't need to know the I/0 map, just the names of the registers.
- The standard definition files for register and bit names can be used.
- The most code size efficient instructions are used to access a register.
- The assembly code can be ported to any device without modifying the code.

2.4.1 The Set-Bit Macro

As all the macros are similar in nature only the SETB are described in details.

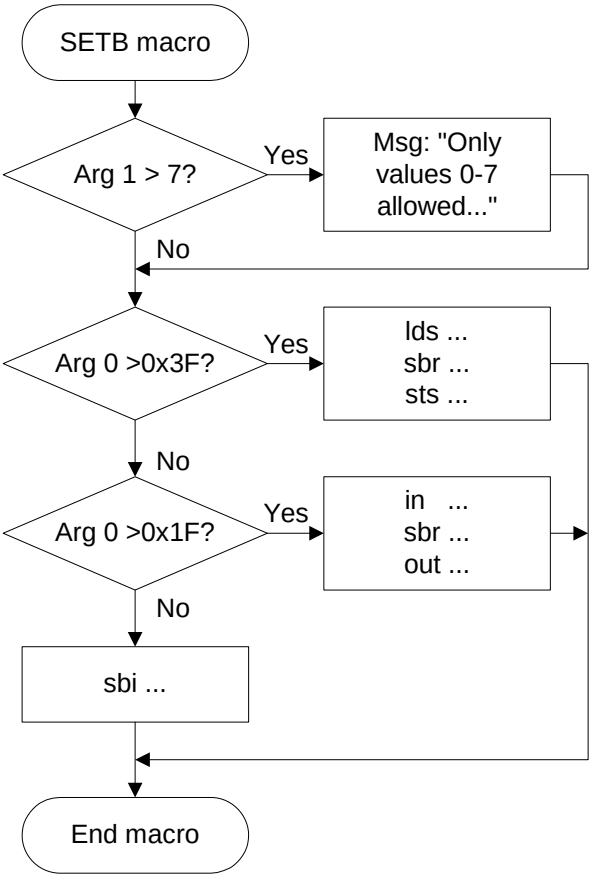
Three arguments are specified for the SETB macro: a destination address, a bit mask and a register. The register is only used if the address is higher than 0x001F, but it is recommended to specify it anyway to ensure correct assembly and best portability opportunities.

The range of the Bit mask is verified to be between 0 and 7, if this condition is violated an error is issued using the .error directive.

If the address is below 0x1F the SBI instruction is used to set the bit. If the Address is between 0x1F and 0x3F IN and OUT instruction is used to access the address. Finally, if the address is above 0x3F the LDS and STS instructions are used.

Figure 2-1 shows how the assembler handles the SETB macro using CA.

Figure 2-1. Assembling flow for CA inside SETB macro.



2.5 Improvements

At the expense of one of the registers Y or Z the LOAD and STORE macros could be improved to execute faster and be more code compact: If one of these registers are reserved for indirect access the STS and LDS instructions could be replaced by STD and LDD. As this is a constraint to the code writer this has not been added in the present implementation.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

2550E-AVR-04/08