AVR034: Mixing C and Assembly Code with IAR Embedded Workbench for AVR

Features

- **Passing Variables between C and Assembly Code Functions**
- **Calling Assembly Code Functions from C**
- **Calling C Functions from Assembly Code**
- **Writing Interrupt Functions in Assembly Code**
- **Accessing Global Variables in Assembly Code**

This application note describes how to use C to control the program flow and main program and assembly modules to control time critical I/O functions.

Introduction

This application note describes how to set up and use the IAR C-compiler for the AVR controller in prosjects including both C and assembly code. By mixing C and assembly designers can combine the powerful C language instructions with the effective hardware-near assembly code instructions.

Table 1. The pluses and minuses of C and Assembly

Assembly	C
+ Full control of Resource Usage	+ Efficient code in larger applications
+ Compact/fast code in small applications	+ Structured code
- Inefficient code in larger applications	+ Easy to maintain
- Cryptic code	+ Portable
- Hard to maintain	- Limited control of Resource Usage
- Non-portable	- Larger/slower code in small applications

Mixing C and Assembler

Application Note

Rev. 1234A–12/98

Passing Variables between C and Assembly Code Functions

When the IAR C-compiler is used for the AVR the register file is segmented as shown in Figure 1.

Scratch registers are not preserved across functions calls. Local registers are preserved across function calls. The Y register(R28:R29) is used as data stack pointer to SRAM. The scratch registers are used to passing parameters and return values between functions.

When a function is called the parameters to be passed to the function is placed in the register file registers R16-R23. When a function is returning a value this value is placed in the register file registers R16-R19, depending on the size of the parameters and the returned value.

Table 2 shows example placement of parameter when calling a function:

Figure 1. Segments in the register file.

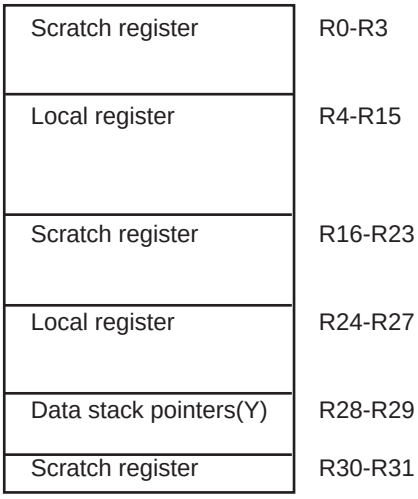

Table 2. Placement and parameters to C-functions

Function	Parameter 1 registers	Parameter 2 registers
func (char ,char)	R16	R20
func (char ,int)	R16	R20, R21
func (int ,long)	R16 ,R17	R20, R21, R22, R23
func (long ,long)	R16, R17, R18, R19	R20, R21, R22, R23

For complete reference of the supported data types and corresponding sizes, see the IAR AT90S Users Guide, Data Representation section.

Example C function call:

```
int get_port(unsigned char temp, int num)
```

When calling this C function the 1 byte parameter ***temp*** is placed in R16, the 2 byte parameter ***num*** is placed in R20:R21. The function returns a 2 byte value which is placed in R16:R17 after return from the function.

If a function is called with more than 2 parameters the first 2 parameters are passed to the function as shown above, the remaining parameters are passed to the function on the data stack. If a function is called with a ***struct*** or ***union*** as parameter a pointer to the structure is passed on to the function on the data stack.

If a function need to use any local registers it first pushes the registers on the data stack. Then return value from the function is placed at adresses R16-R19 depending on the size of the returned value.

Example 1

Calling Assembly Code Functions from a C Program

- with no parameters and no return value

Example C code for calling assembly code function

```
#include "io8515.h"
extern void get_port(void);/* Function prototype for asm function */
void main(void)
{
  DDRD = 0x00;/* Initialization of the I/O ports*/
  DDRB = 0xFF;
  while(1)/* Infinite loop*/
{
    get_port();/* Call the assembler function   */
  }
}
```

The called assembly code function:

```
NAME get_port
    #include "io8515.h"     ; The #include file must be within the module
    PUBLIC get_port         ; Declare symbols to be exported to C function
    RSEG CODE               ; This code is relocatable, RSEG

get_port;                   ; Label, start execution here
    in R16,PIND             ; Read in the pind value
    swap R16                ; Swap the upper and lower nibble
    out PORTB,R16           ; Output the data to the port register
    ret                     ; Return to the main function
END
```

Calling Assembly Code Functions from a C Function

-passing parameters and returning values.

This example C function is calling an assembler function. The 1 byte ***mask*** is passed as a parameter to the assembly function, ***mask*** is placed in R16 before the function call. The assembly function is returning a value in R16 to the C variable ***value***.

```
#include "io8515.h"
char get_port(char mask);       /*Function prototype for asm function */
void C_task main(void)
{
    DDRB=0xFF
    while(1)                    /* Infinite loop*/
    {
        char value, temp;       /* Decalre local variables*/
        temp = 0x0F;
        value = get_port(temp); /* Call the assembler function */
        if(value==0x01)
        {
         /* Do something if value is 0x01        */
    PORTB=~(PORTB);             /* Invert value on Port B  */
```

```
        }
    }
}
```

The called assembly code function:

```
NAME get_port
    #include "io8515.h"   ; The #include file must be within the module
    PUBLIC get_port       ; Symbols to be exported to C function

RSEG CODE                 ; This code is relocatable, RSEG
get_port:                 ; Label, start execution here
      in   R17,PIND       ; Read in the pinb value
      eor  R16,R17        ; XOR value with mask(in R16) from main()
      swap R16            ; Swap the upper and lower nibble
      rol  R16            ; Rotate R16 to the left
      brcc ret0           ; Jump if the carry flag is cleared
      ldi  r16,0x01       ; Load 1 into R16, return value
      ret                 ; Return
ret0: clr  R16            ; Load 0 into R16, return value
ret                       ; Return
END
```

Calling C Functions from Assembly Code

Assuming that the assembly function calls the standard C library routine rand() to get a random number to output to the port. The rand() routine returns an integer value(16 bits). This example writes only the lower byte/8bits to a port.

```
NAME get_port
    #include "io8515.h"     ; The #include file must be within the module
    EXTERN rand, max_val    ; External symbols used in the function
    PUBLIC get_port         ; Symbols to be exported to C function

    RSEG CODE               ; This code is relocatable, RSEG

get_port:                   ; Label, start execution here
    clr     R16             ; Clear R16
    sbis    PIND,0          ; Test if PIND0 is 0
    rcall   rand            ; Call RAND() if PIND0 = 0
    out     PORTB,R16       ; Output random value to PORTB
    lds     R17,max_val     ; Load the global variable max_val
    cp      R17,R16         ; Check if number higher than max_val
    brlt    nostore         ; Skip if not
    sts     max_val,R16     ; Store the new number if it is higher
nostore:
    ret                     ; Return
END
```

Writing Interrupt Functions in Assembly.

Interrupt functions can be written in assembly. Interrupt functions can not have any parameters nor returning any value. Because an interrupt can occur anywhere in the program execution it needs to store all used registers on the stack.

Care must be taken when assembler code is placed at the interrupt vector adresses to avoid problems with the interrupt functions in C.

Example code placed at interrupt vector.

```
NAME EXT_INT1
#include "io8515.h"

extern c_int1
COMMON INTVEC(1)            ; Code in interrupt vector segment
ORG INT1_vect               ; Place code at interrupt vector
    RJMP   c_int1           ; Jump to assembler interrupt function
ENDMOD

                          ;The interrupt vector code performs a jump to the function c_int1:

NAME c_int1
    #include "io8515.h"
PUBLIC c_int1               ; Symbols to be exported to C function

    RSEG CODE               ; This code is relocatable, RSEG
c_int1:
    st     -Y,R16           ; Push used registers on stack
    in     R16,SREG         ; Read status register
    st     -Y,R16           ; Push Status register

    in     R16,PIND         ; Load in value from port D

    com    R16              ; Invert it

    out    PORTB,R16        ; Output inverted value to port B

    ld     R16,Y+           ; Pop status register
    out    SREG,R16         ; Store status register
    ld     R16,Y+           ; Pop Register R16
    reti
END
```

Accessing Global Variables in Assembly

The main program introduces a global variable called **max_val**. To access this variable in assembly the variable must be declared as **EXTERN max_val**. To access the variable the assembly function uses LDS (Load Direct from SRAM) and STS (STore Direct to SRAM) intructions.

```
#include "io8515.h"

char max_val;
void get_port(void);        /* Function prototype for assembler function */

void C_task main(void)
{
    DDRB = 0xFF;            /* Set port B as output */
    while(1)                /* Infinite loop */
    {
        get_port();         /* Call assembly code function */
    }
}
```

```
NAME get_port
    #include "io8515.h"     ; The #include file must be within the module
    EXTERN rand, max_val    ; External symbols used in the function
    PUBLIC get_port         ; Symbols to be exported to C function

    RSEG CODE               ; This code is relocatable, RSEG

get_port:                   ; Label, start execution here
    clr     R16             ; Clear R16
    sbis    PIND,0          ; Test if PIND0 is 0
    rcall   rand            ; Call RAND() if PIND0 = 0
    out     PORTB,R16       ; Output random value to PORTB
    lds     R17,max_val     ; Load the global variable max_val
    cp      R17,R16         ; Check if number higher than max_val
    brlt    nostore         ; Skip if not
    sts     max_val,R16     ; Store the new number if it is higher
nostore:
    ret                     ; Return
END
```

References

IAR Systems AT09S USER GUIDE