Defining Interrupt Functions

This technical note details how to
- Implement interrupt functions
- Initialize the vector table.

Defining an Interrupt

Interrupt functions can be defined using:
- The pragma "**TRAP_PROC**"
- The keyword "**interrupt**"

Using the pragma TRAP_PROC:

The pragma **TRAP_PROC** tells the compiler that a function is an interrupt function (i.e. it should be terminated with RTI instead of RTS).
The pragma **TRAP_PROC** is only valid for the next function implemented, so you have to specify it in front of each interrupt functions.

Example of interrupt function definition in ANSI C with pragmas:

```
unsigned int intCount = 1;
#pragma TRAP_PROC
void IntFunc(void)
{
  intCount++;
}

#pragma TRAP_PROC
void IntFunc2(void)
{
  intCount--;
}

#pragma TRAP_PROC
void IntFunc3(void)
{
  intCount=intCount*5;
}

void main (void)
{
....
}
```

In your prm file:

```
/* add the following lines at the end of the PRM file */
…
VECTOR ADDRESS 0xFFF0 IntFunc1   /* 0xFFF0 contains the address of IntFunc1 */
VECTOR ADDRESS 0xFFF2 IntFunc2   /* 0xFFF2 contains the address of IntFunc2 */
VECTOR ADDRESS 0xFFF4 IntFunc3   /* 0xFFF4 contains the address of IntFunc3 */
```

Example of interrupt function definition in C++ with pragmas:

If you use C++, the compiler performs 'name mangling' on the name of interrupt functions too. In this case the mangled name has to be specified in the linker parameter file (e.g.MyIntFkt_Fv). To prevent name mangling, the function can be declared with 'extern "C"'

In your C++ file

```
unsigned int intCount = 1;
#pragma TRAP_PROC
extern "C" void IntFunc1(void)
{
  intCount++;
}

#pragma TRAP_PROC
void IntFunc2(void)
{
  intCount--;
}

#pragma TRAP_PROC
extern "C" void IntFunc3(void)
{
  intCount=intCount*5;
}

#pragma DATA_SEG DEFAULT
void main (void)
{
....
}
```

In your prm file:

```
/* add the following lines at the end of the PRM file */
…
VECTOR ADDRESS 0xFFF0 IntFunc1   /* 0xFFF0 contains the address of IntFunc1 */
VECTOR ADDRESS 0xFFF2 IntFunc2__Fv /* 0xFFF2 contains the address of
                                      IntFunc2 */
VECTOR ADDRESS 0xFFF4 IntFunc3   /* 0xFFF4 contains the address of IntFunc3 */
```

Using the keyword 'interrupt':

The keyword "**interrupt**" can be used to specify that a function is an interrupt function.

Syntax:

```
interrupt  <function_name>
/* interrupt number is specified in the PRM file */
{
   …/*code*/
}
```

or

```
interrupt <vector_number> <function_name>
{
   …/*code*/
}
```

The first syntax tells the compiler that the function is an interrupt function (i.e. it should be terminated with RTI rather than RTS). The vector table must be initialized in the PRM file.
The second syntax tells the compiler that the function is an interrupt function (i.e. it should be terminated with RTI rather than RTS) and initializes the corresponding entry in the vector table. The location of the entry associated with vector_number depends on the CPU you are using.

- For HC05, ST7, HC08, HC11 and HC11, the reset vector is vector number 0 (at address 0xFFFE), vector number 1 is located just before the vector 0 (at address 0xFFFC), and so on.
- For HC16 the reset vector is vector number 0 (located at address 0), vector number 1 is located just after the vector 0 (at address 0x2) and so on.

Note:

When you are using HC05 or ST7, if the reset vector is not located at address 0xFFFE, if you want to use vector numbers while defining interrupts, you have to tell the compiler where the reset vector is located. This can be performed using the environment variable RESET_VECTOR.

Example 1:

```
interrupt 3  IntFunc1()
{
   …/*code*/
}
```

Means that the third entry in the vector table is initialized with the address of IntFunc1().

Example 2:

In the C file:

```
interrupt   IntFunc2()
{
   ….  .
}
```

In the PRM file:

```
VECTOR ADRESS 0xFFF2 IntFunc2
```

Means that the address of IntFunc2 is written at address 0xFFF2.

Note:

Not all Back Ends may support this keyword. See section "Non-ANSI Key-words" in the Back End of your compiler manual.
Refer to your compiler manual for more details about the interrupt management with the interrupt keyword.

Saving Registers

According to the CPU you are using an additional parameter can be specified after the pragma TRAP_PROC: SAVE_REGS. The parameter SAVE_REGS ensures that all the CPU registers or compiler pseudo registers are saved when entering an interrupt function and retored when existing it. Please refer to the Back End chapter in your compiler manual for more information about saving/restoring interrupts.