

IAR Application Note 430-04

Disabling and enabling interrupts in interrupt-disabled environments for MSP430

SUMMARY

This application note describes how to disable and enable general interrupts in MSP430 applications where it is unknown whether general interrupts have been disabled earlier during execution.

KEYWORDS

interrupt disable, interrupt enable

PROBLEM

How can I be sure that general interrupts will not be enabled too early?

Example:

```
function_1 ()
{
...
    _DINT();          // GIE flag clear will disable general interrupts
...
    function_2();    // A call to function_2()
...
    _EINT();         // The final GIE flag set has nothing to enable,
                    // because interrupts are already enabled by function_2.
...
}

function_2 ()
{
...
    _DINT();         // a second GIE clear will do nothing, because
                    // interrupts are already disabled in function_1
...
    _EINT();         // a following GIE set will enable all interrupts,
                    // but the interrupts should remain disabled because of
                    // the first GIE flag clear in function_1.
...
}
```

Note: The `_DINT()` and `_EINT()` operations are intrinsic functions defined in the IAR C Compiler. `_DINT()` disables interrupts using a `DINT` instruction and `_EINT()` enables interrupts using an `EINT` instruction.

When the `_DINT()` operation is performed in `function_2` (i.e. after the `_DINT()` operation in `function_1`), the `_EINT()` operation in `function_2` will generate unpredictable results since all interrupts have to be disabled until the `_EINT()` operation in `function_1`. To avoid that, we need to find a way to keep track of the interrupt disable/enable operations so that the interrupts are not enabled before the last enabling operation.

DIFFICULTIES

Since the MSP430 family has no stack available that can be used to handle the enabling or disabling of interrupts, we have to use a user-defined solution.

SOLUTION

One solution is to count the number of times that a disable interrupt command has been executed (of course only one "disable" really takes effect). Interrupts will only be enabled when the counter decrements to zero. We must introduce a count variable "IntsDisableCount". It increments every time a general interrupt disable is executed and decrements every time a general interrupt enable is executed. The real access to the GIE bit inside the status register is only performed if the variable "IntsDisableCount" reaches zero. If the variable is not zero, no interrupts enable operation is actually performed.

With this method, no pure `_DINT()` operation or `_EINT()` operation is allowed anywhere in the firmware except for the very first time after startup, when one and only one `_EINT()` operation is performed.

Let us create two macros to perform the counter action and the real access to the status register.

```
unsigned char IntsDisableCount; // count of performed disables
#define GIEDisable             _DINT(); \
                               IntsDisableCount++;
#define GIEEnable             if(--IntsDisableCount==0) \
                               _EINT();
```

These macros handle the synchronization of general interrupt disabling and enabling in most cases. However, if the general interrupt is disabled before the first "GIEDisable" operation is performed, the next "GIEEnable" operation will enable the interrupts. To avoid this, some additional code must be inserted into the two macros to store the status at the first call of "GIEDisable".

```
unsigned char IntStatus; // status of GIE flag
unsigned char IntsDisableCount; // count of performed disables
#define GIEDisable             if(IntsDisableCount==0) \
                               IntStatus=_BIC_SR(0x0008); \
                               IntsDisableCount++;
#define GIEEnable             if(--IntsDisableCount==0&&(IntStatus&0x0008)) \
                               _EINT();
```

Note1: `_BIC_SR()` is an intrinsic function that clears bits in the status register.

Note2: If the macros are used in "if", "while" or "for" statement bodies, make sure to enclose them in "{" and "}". Otherwise faulty code could be produced.

BENEFITS

Experience shows that many undeclared faults in firmware designs are due to incorrect handling of general interrupt disables and enables. The solution described above avoids these problems completely. Furthermore, these macros are so small that they can fit into any firmware. The user does not have to worry about unsynchronized disables and enables.

Author:

Thomas Wild
Project Manager - Consulting & Services
IAR Systems AG

Contact information

USA

IAR Systems Inc.
One Maritime Plaza
San Francisco,
CA 94111
Tel: +1 415-765-5500
Fax: +1 415-765-5503
Email: info@iar.com

SWEDEN

IAR Systems AB
P.O. Box 23051
S-750 23 Uppsala
Tel: +46 18 16 78 00
Fax: +46 18 16 78 38
Email: info@iar.se

GERMANY

IAR Systems AG
Posthalterring 5
D-855 99 Parsdorf
Tel: +49 89 90 06 90 80
Fax: +49 89 90 06 90 81
Email: info@iar.de

UK

IAR Systems Ltd.
9 Spice Court,
Ivory Square
London SW11 3UE
Tel: +44 171 924 3334
Fax: +44 171 924 5341
Email: info@iarsys.co.uk

DENMARK

IAR Systems A/S
Elkjervej 30-32
DK-8230 Aabyhoj
Tel: +45 86 25 11 11
Fax: +45 86 25 11 91
Email: info@iar.dk

www.iar.com

Copyright© 2000 IAR Systems

IAR and C-SPY are registered trademarks of IAR Systems. IAR Embedded Workbench is a trademark of IAR Systems. Windows is a trademark of Microsoft Corporation. All other products are registered trademarks or trademarks of their respective owners. Product features, availability, pricing and other terms and conditions are subject to change by IAR Systems without prior notice.