Techniques to Disable Global Interrupts

This application brief discusses four methods for disabling global interrupts. The method best suited for the application may then be used. All discussion will be specific to the PIC16CXX family of products, but these concepts are also applicable to the PIC17C42, and are shown in the B examples.

To disable interrupts, either the Global Interrupt Enable (GIE) bit must be cleared or all the individual interrupt enable bits must be cleared. An issue arises when an instruction clears the GIE bit and an interrupt occurs "simultaneously". For example, when a program executes the instruction BCF INTCON, GIE (at address PC), there is a possibility that an interrupt will occur during this instruction. If an interrupt occurs during this instruction, the program would complete execution of this instruction, and then immediately branch to the user's interrupt service routine. This occurs because the GIE bit was not clear (disabled) when the interrupt occurred. Normally at the end of the interrupt service routine is the RETFIE instruction. This instruction causes the program to return to the instruction at PC + 1, but also sets the GIE bit (enabled). Therefore the GIE bit is not cleared as expected, and unintended program execution may occur.

One method to ensure that the GIE bit is cleared is shown in Example 1, as well as in the PIC16CXX data sheets. This method tests the state of the GIE bit, after clearing, to ensure that it was not accidentally set in the user's interrupt service routine by the RETFIE instruction. If the GIE bit was accidentally set, the program branches back to the instruction that clears the GIE bit.

In this method, the time to ensure that the GIE bit is cleared is indeterminate. Depending on the frequency of the enabled interrupts during this code segment, unexpected delays into the following code segment may occur. For some applications, this may be undesirable. The following three methods address this issue.

5

EXAMPLE 1A: FORCING THE GIE BIT CLEAR (METHOD 1, PIC16CXX)

```
        :
LOOP    BCF     INTCON, GIE       ; Disable Global Interrupts
        BTFSC   INTCON, GIE       ; Global Interrupts Disabled?
        GOTO    LOOP              ; NO, try again
        :                         ; YES, continue with program flow
        :
        BSF     INTCON, GIE       ; Re-enable Global Interrupts
```

EXAMPLE 1B: FORCING THE GLINTD BIT CLEAR (METHOD 1, PIC17C42)

```
        :
LOOP    BSF     CPUSTA, GLINTD    ; Disable Global Interrupts
        BTFSS   CPUSTA, GLINTD    ; Global Interrupts Disabled?
        GOTO    LOOP              ; NO, try again
        :                         ; YES, continue with program flow
        :
        BSF     INTCON, GIE       ; Re-enable Global Interrupts
```

The second method is to disable the individual interrupt enable bits. If it is known which bits are enabled at this point, it can easily be done. Example 2 shows the disabling of interrupts, where it is known which sources are enabled (some peripheral interrupts and the T0CKI pin interrupt).

This method also requires the same number of instructions for the disabling/enabling of interrupts as method 1, but requires a knowledge of which individual interrupt enable bits need to be disabled and (more importantly) re-enabled. The major advantage of this method is that it can minimizes the time delay entering the code segment which follows the point where interrupts are disabled.

EXAMPLE 2A: CLEARING KNOWN INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 2, PIC16CXX)

```
:
MOVLW  b'10011111'      ; Disable  Peripheral and T0CKI pin interrupts,
ANDWF  INTCON, F        ; All other bits unchanged
:
:
:
MOVLW  b'01100000'      ; Re-enable Peripheral and T0CKI pin interrupts,
IORWF  INTCON, F        ; All other bits unchanged
:
```

EXAMPLE 2B: CLEARING KNOWN INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 2, PIC17C42)

```
:
MOVLW b'11110011'      ; Disable Peripheral and T0CKI pin interrupts,
ANDWF INTSTA, F        ; All other bits unchanged
:
:
:
MOVLW b'00001100'      ; Re-enable Peripheral and RT pin interrupts,
IORWF INTSTA, F        ; All other bits unchanged
:
```

Method 3 can be used if the states of the individual interrupt enable bits are unknown. A temporary byte of data RAM is required to store the value of the INTCON register. This method is shown in Example 3.

This method also requires more instructions for the disabling/enabling of interrupts than in method 1 or method 2, and also a byte of data RAM to temporarily store the value of the INTCON register. The major advantage of this method is that it minimizes the time delay into the code segment which follows the point where interrupts are disabled.

EXAMPLE 3A: CLEARING THE INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 3, PIC16CXX)

```
:
MOVF   INTCON, W        ; Move the value in INTCON to
MOVWF  S_INTCON         ; a shadow register
MOVLW  b'10000111'      ; Disable all individual interrupts,
ANDWF  INTCON, F        ; All other bits unchanged
:
:
:
MOVF   S_INTCON, W      ; Restore the INTCON register
IORWF  INTCON, F        ;
:
```

EXAMPLE 3B: CLEARING THE INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 3, PIC17C42)

```
:
MOVPF INTSTA, S_INTSTA ; Move the value in INTSTA to a shadow register
MOVLW b'11110000'      ; Disable all individual interrupts,
ANDWF INTSTA, F        ; All other bits unchanged
:
:
:
MOVF  S_INTSTA, W      ; Restore the INTSTA register
IORWF INTSTA, F        ;
:
```

The final method is to use a RAM location to "shadow" the value of the GIE bit. This shadow bit can then be used in the interrupt service routine to determine which return instruction to use. That is, either one of the RETURN or the RETFIE (which enables the GIE) instructions. Example 4 shows this implementation, which require that a general purpose bit be available to hold the "shadow " GIE value. In this example, the shadow GIE (S_GIE) bit is contained in the register FLAG_REG. If an interrupt occurs during the clearing of the shadow GIE, the interrupt is responded to. At the end of the interrupt service routine, the shadow GIE bit is cleared so the RETURN instruction is executed. The GIE bit remains disabled and program execution returns to the instruction which tries to clear the GIE (disable). No interrupts can occur during this instruction since the GIE bit was not re-enabled after the interrupt service routine.

This method also requires more instructions for the disabling/enabling of interrupts than in method 1 or method 2, a single bit of data RAM to temporarily store the value of the desired GIE value, and increases the interrupt service routine execution time by one instruction cycle, for most occurrences of interrupts (two cycles worst case). The major advantage of this method is that it minimizes the time delay into the code segment which follows the point where interrupts are disabled. Also, the individual interrupt enable bits need not be modified.

EXAMPLE 4A: THE "SHADOW" GIE BIT (METHOD 4, PIC16CXX)

```
       :
              org    0x004
INT_SERVICE_ROUTINE
       :
       :
       BTFSC  FLAG_REG, S_GIE     ; Is the S_GIE bit enabled?
       RETFIE                     ; YES, the GIE should be enabled
       RETURN                     ; NO, the GIE should be disabled
END_INT_SERVICE_ROUTINE
;
MAIN   :
       :
       :
       BCF    FLAG_REG, S_GIE     ; Disable the shadow GIE bit
       BCF    INTCON, GIE         ; Disable the GIE bit
       :
       :
       :
       BSF    FLAG_REG, S_GIE     ; Enable the shadow GIE bit
       BSF    INTCON, GIE         ; Enable the GIE bit
       :
       :
       END
```

EXAMPLE 4B: THE "SHADOW" GLINTD BIT (METHOD 4, PIC17C42)

```
       :
              org    0x004
INT_SERVICE_ROUTINE
       :
       :
       BTFSS  FLAG_REG, S_GLINTD    ; Is the S_GLINTD bit enabled?
       RETFIE                       ; YES, the GLINTD should be enabled
       RETURN                       ; NO, the GLINTD should be disabled
END_INT_SERVICE_ROUTINE
;
MAIN   :
       :
       :
       BSF    FLAG_REG, S_GLINTD    ; Disable the shadow GLINTD bit
       BSF    CPUSTA, GLINTD        ; Disable the GLINTD bit
       :
       :
       :
       BCF    FLAG_REG, S_GLINTD    ; Enable the shadow GLINTD bit
       BCF    CPUSTA, GLINTD        ; Enable the GLINTD bit
       :
       :
       END
```

In conclusion, different methods exist to ensure that all interrupts are disabled. The requirement(s) of the applications determine which of the methods is the best fit. A comparison of the different methods is shown in Table 1.

TABLE 1: COMPARISON OF DIFFERENT METHODS

	Program Memory	Data Memory	Cycle Delay (Tcy) Best Case	Worst Case
Method 1	2 words * N	---	2	Indeterminate
Method 2	2 words * N	---	1	1 + Tisr
Method 3 - PIC16CXX	4 words * N	1 byte	3	3 + Tisr
- PIC17C42	3 words * N	1 byte	2	2 + Tisr
Method 4	2 words * N + 2 words	1 bit	1♣	1 + (Tisr + 2)

Legend: N - Number of occurances to disable / re-enable interrupts.

Tisr - Time to execute the interrupt service routine.

♣ This method increases the interrupt service routine time (Tisr) by 1 cycle for most occurances (2 cycles worst case).

5

Author: Mark Palmer - Logic Products Division
Contributions by Martin Burghardt - Manager Applications (Central Europe)

NOTES:

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200 Fax: 602 786-7277
Technical Support: 602 786-7627
Web: http://www.mchip.com/microhip

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770 640-0034 Fax: 770 640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508 480-9990 Fax: 508 480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 708 285-0071 Fax: 708 285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177 Fax: 214 991-8588

Dayton
Microchip Technology Inc.
35 Rockridge Road
Englewood, OH 45322
Tel: 513 832-2543 Fax: 513 832-2841

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888 Fax: 714 263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516 273-5305 Fax: 516 273-5335

AMERICAS (continued)

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408 436-7950 Fax: 408 436-7955

ASIA/PACIFIC

Hong Kong
Microchip Technology
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 2 401 1200 Fax: 852 2 401 3431

Korea
Microchip Technology
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku,
Seoul, Korea
Tel: 82 2 554 7200 Fax: 82 2 558 5934

Singapore
Microchip Technology
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65 334 8870 Fax: 65 334 8850

Taiwan
Microchip Technology
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2 717 7175 Fax: 886 2 545 0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

France
Arizona Microchip Technology SARL
2 Rue du Buisson aux Fraises
91300 Massy - France
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Muenchen, Germany
Tel: 49 89 627 144 0 Fax: 49 89 627 144 44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041
Agrate Brianza (MI) Italy
Tel: 39 039 689 9939 Fax: 39 039 689 9883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/22/95