Code for EE2361

12/5/16

Example: A/D input used to control PWM

```
/* external interrupt code for PIC24 to light led
 * for a pulse duration, uses interrupts. Duration
 * is determined by the ATD.

   T. Posbergh, 12/2/2016
   PIC24FJ64GA002                                  */

#include <xc.h>

#pragma config FNOSC=FRCPLL, POSCMOD=NONE
#pragma config FWDTEN=OFF, GCP=OFF, JTAGEN=OFF

#undef  _ISR
#define _ISR __attribute__ ((interrupt, no_auto_psv))

int main(void) {

/* configure I/O pins    */
    AD1PCFG = 0xFFFE;      // AN0 is analog
    TRISB = 0x7FFF;        // RB15 is output, RB<0:3> are input
    RPOR7bits.RP15R = 18;  // assign OC1 to RP15

/* configure Timer2      */
    T2CON = 0x0000;        // prescale 1:1,
    TMR2 =  0x0000;        // initialize to 0
    PR2 =   16000-1;       // for 1 kHz or 1 ms

/* configure ATD         */
    AD1CON1 = 0x00E0;       // SSRC is 111 for timed conversion
    AD1CHS  = 0x0000;       // AN0 is ADC input
    AD1CSSL = 0;
    AD1CON2 = 0x0000;       // no interrupts
    AD1CON3 = 0x0001;       // no auto sample, Tad = 2*Tcy

    AD1CON1bits.ADON = 1;   // turn on ADC

/* output compare        */
    OC1CON = 0x0000;       // turn off OC1
    OC1R =   0x0000;       // initialize
    OC1RS =  0x0000;
    OC1CON = 0x0006;       // PWM, no fault

/* configure interrupts */
    IPC1bits.T2IP = 4;     // Timer2 interrupt priority
    IFS0bits.T2IF = 0;     // reset flag
    IEC0bits.T2IE = 1;     // enable interrupts

    T2CONbits.TON = 1;     // enable Timer2

    while(1);              // forever loop
```

set up

main loop

```
    return 0;
}


/* Timer2 interrupt service routine  */
void _ISR _T2Interrupt(void)
{
        IFS0bits.T2IF = 0;                  // reset flag
        OC1RS  = (15999*ADC1BUF0)>>10;      // duty cycle?
        AD1CON1bits.SAMP = 1;               // manual sample start
}
```

ISR for example of A/D input to control the duty cycle of a PWM code

Note this will not compute the correct duty cycle since it will truncate 15999 * ADC1BUF0 to 16-bits before shifting.

Next look at corresponding assembly for this ISR

```
000002f6 <__T2Interrupt>:
 2f6:     80 9f be       mov.d     w0, [w15++]
 2f8:     00 00 fa       lnk       #0x0
 2fa:     84 e0 a9       bclr.b    0x84, #0x7
 2fc:     01 18 80       mov.w     0x300, w1
 2fe:     f0 e7 23       mov.w     #0x3e7f, w0
 300:     00 88 b9       mul.ss    w1, w0, w0
 302:     00 00 78       mov.w     w0, w0
 304:     4a 00 de       lsr.w     w0, #0xa, w0
 306:     00 0c 88       mov.w     w0, 0x180
 308:     20 23 a8       bset.b    0x320, #0x1
 30a:     00 80 fa       ulnk
 30c:     4f 00 be       mov.d     [--w15], w0
 30e:     00 40 06       retfie
```

From this assembly

at about 1 Tcy each instruction with additional overhead we will have plenty of time to sample and convert between interrupts

Note that mul.ss places a 32-bit result in W0:W1 but only the part in W0 appears to be used. How to fix this ??

→ fix assembly
→ fix C-code

# MUL.SS

**Integer 16x16-bit Signed Multiply**

| Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E |
|---|---|---|---|---|---|---|
| | X | X | X | X | X | X |

Syntax:

| {label:} | MUL.SS | Wb, | Ws, | Wnd |
|---|---|---|---|---|
| | | | [Ws], | |
| | | | [Ws++], | |
| | | | [Ws--], | |
| | | | [++Ws], | |
| | | | [--Ws], | |

Operands:
Wb ∈ [W0 ... W15]
Ws ∈ [W0 ... W15]
Wnd ∈ [W0, W2, W4 ... W12]

Operation: signed (Wb) * signed (Ws) →Wnd:Wnd + 1

Status Affected: None

Encoding:

| 1011 | 1001 | 1www | wddd | dppp | ssss |
|---|---|---|---|---|---|

Description:

Multiply the contents of Wb with the contents of Ws, and store the 32-bit result in two successive working registers. The least significant word of the result is stored in Wnd (which must be an even numbered working register), and the most significant word of the result is stored in Wnd + 1. Both source operands and the result Wnd are interpreted as two's complement signed integers. Register direct addressing must be used for Wb and Wnd. Register direct or register indirect addressing may be used for Ws.

The 'w' bits select the address of the base register.
The 'd' bits select the address of the lower destination register.
The 'p' bits select the source Address mode.
The 's' bits select the source register.

**Note 1:** This instruction operates in Word mode only.
**2:** Since the product of the multiplication is 32 bits, Wnd must be an even working register. See Figure 4-2 for information on how double words are aligned in memory.
**3:** Wnd may not be W14, since W15<0> is fixed to zero.
**4:** The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

Words: 1

Cycles: 1(1)

**Note 1:** In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 "Multi-Cycle Instructions"**.