```
//*******************************************************************************************
//  MSP430 ULP Advisor Demo - Good Example
//
//  Description: This code demonstrates how a program can be optimized for
//  low power consumption. The MSP430 low power modes and peripherals are used.
//
//  Comment: This code addresses the largest improvements for power consumption
//           and there are more optimizations possible.
//
//               MSP430G2553
//            -----------------
//        /|\|              XIN|-
//         | |                 | 32kHz
//         --|RST          XOUT|-
//  Button_  |                 |
//       - -->|P1.3         P1.0|-->LED1
//            |             P1.6|-->LED2
//            |             P1.1|<--RXD    Serial UART
//            |             P1.2|-->TXD    Interface
//
//  This software has been placed into the Public Domain by Texas Instruments
//  Incorporated and is example code only  THIS SOFTWARE IS PROVIDED BY TEXAS
//  INSTRUMENTS INCORPORATED"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
//  INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
//  AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TEXAS
//  INSTRUMENTS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  SPECIAL,
//  EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  LIMITED TO,
//  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
//  OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
//  WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
//  OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
//  ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
//  Roger Neumair
//  Texas Instruments, Inc
//  May 2012
//  Built with Code Composer Studio v5
//*******************************************************************************************


#include <msp430g2553.h>
#include <string.h>


#define DEBOUNCE_DELAY 6250               // Debounce delay. 1 Cycle = 8us @ 1MHz SMCLK. => 50ms


// Definitions for LEDs and button on the MPS430 Launchpad
#define BUTTON BIT3
#define LED1 BIT0
#define LED2 BIT6


#define APPSTATE_IDLE    0
#define APPSTATE_MEASURE 1


#define MEASUREMENT_INFO_SIZE 14
union MEASUREMENT_INFO {
        unsigned char b[MEASUREMENT_INFO_SIZE];
        struct {
                unsigned short A;
                unsigned short aA;
                unsigned short B;
                unsigned short aB;
                unsigned short F;
                unsigned short aF;
                unsigned short x;
        } rs;
};
typedef union MEASUREMENT_INFO MEASUREMENT_INFO_T;


//  Globals
//-------------------------------------
MEASUREMENT_INFO_T tr;           // transmit measurement record

unsigned long temp;
unsigned long TmpDegF;
unsigned long TmpDegC;

//unsigned char ButtonFlag = 0;
unsigned char ADC10ReadyFlag = 0;
unsigned char Port1IRQHappened = 0;
unsigned char OneSecondFlag = 0;
unsigned char DebounceFlag = 0;


//  Function Prototypes
//-------------------------------------
void Setup_Ports(void);
void Setup_ADC10(void);
void Setup_UART(void);
void Setup_Timer(void);
void ProcessCurrentSample(unsigned short smpl, MEASUREMENT_INFO_T* x);
```

```
//=============================================================================
//  main program
//=============================================================================
void main(void)
{
        unsigned short app_state = APPSTATE_IDLE; //APPSTATE_MEASURE;  //APPSTATE_IDLE;
        unsigned short TA0R1, TA0R2;
        MEASUREMENT_INFO_T cr;                  // current measurement record

        WDTCTL = WDTPW + WDTHOLD;                       // Stop watchdog timer

        // MSP430G2553 setup
        Setup_Ports();
        Setup_UART();
        Setup_Timer();
        __enable_interrupt();                                           // Enable interrupts
        Setup_ADC10();


        // Main processing loop
        while(1)
        {

                LPM3;                   // Wait for wakeup event (Timer, ADC10, or button)

                // Start debouncing if a port 1 interrupt occurred
                if(Port1IRQHappened)
                {
                        Port1IRQHappened = 0;

                        // Start debounce of Button. Flag is set by ISR
                        DebounceFlag = 0;
                        if(app_state == APPSTATE_IDLE)
                        {
                                // Start timer when system is in IDLE mode
                                TA0CTL = TASSEL_1 + ID_0 + MC_2 + TACLR + TAIE;
                        }

                        // Synchronize readout of TAR
                        TA0R1 = TA0R;
                        TA0R2 = TA0R;
                        if(TA0R1 != TA0R2)
                                TA0R2 = TA0R;

                        // Set CCR1 count and enable CCR1 interrupt
                        TA0CCR1 = TA0R2 + DEBOUNCE_DELAY;
                        TACCTL1 |= CCIE;
                }       // if(Port1IRQHappened)


                if(DebounceFlag)
                {
                        // Button debounce by software.
                        // Check if button is still pressed after debounce delay expired
                        DebounceFlag = 0;

                        // Assumption: button is low active
                        if(!(P1IN & BUTTON))            // if Button is still pressed
                        {
                                // Toggle app_state
                                if(app_state == APPSTATE_MEASURE)
                                {
//                                      P1OUT &= ~LED2;
                                        app_state = APPSTATE_IDLE;
                                        // Stop the timer.
                                        TA0CTL = 0;
                                }       // if(app_state == APPSTATE_MEASURE)
                                else
                                {
//                                      P1OUT |= LED2;
                                        app_state = APPSTATE_MEASURE;
                                        TA0CTL = TASSEL_1 + MC_2 + TACLR + TAIE;  // Restart the timer
                                }       // else .. if(app_state == APPSTATE_MEASURE)
                        }       // if(!(P1IN & BUTTON))

                        P1IFG &= ~BUTTON;                               // Clear interrupt flag
                        P1IE |= BUTTON;                                 // Enable button interrupt again
                }       // if(DebounceFlag)

                if(OneSecondFlag)       // One second interval reached
                {
                        OneSecondFlag = 0;
                        if(ADC10ReadyFlag)
                        {
                                unsigned short Sample;
                                // ADC10 completed sampling and conversion
                                ADC10ReadyFlag = 0;

                                Sample = ADC10MEM;
                                ProcessCurrentSample(Sample,&cr);       // Do the calculation

                                // Do the data transmit
                                IE2 |= UCA0TXIE;                                                // Enable USCI_A0 TX interrupt
                        }       // if(ADC10ReadyFlag)

                        if(app_state == APPSTATE_MEASURE)
```

```
                        {
                                //Start sampling and conversion
                                ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
//                              P1OUT ^= LED1;
//                              P1OUT |= LED2;
                        }       //      if(app_state == APPSTATE_MEASURE)

                }       // if(OneSecondFlag)

                if(app_state == APPSTATE_IDLE)
                {
//                      P1OUT &= ~LED2;
                }       //      if(app_state == APPSTATE_MEASURE)


        }       // while(1)

}       // main



// =======================================================
//  Functions
// =======================================================


// =======================================================
// Setup timer to generate 1s intervals
// Uses continuous mode to allow easy use for other timings
// =======================================================
void Setup_Timer(void)
{
        // Combining TA0 and TA1-TAIFG interrupts gives 1s timing
        // TA1 IRQ when overflow (every 2s)
        // TA0 (CCR0) (every 2s)

        // ACLK /1, Continuous mode, Clear TAR,
        // Generate TAIFG interrupt on overflow (65535 -> 0)
        TA0CTL = TASSEL_1 + ID_0 + MC_2 + TACLR + TAIE;

        // Generate a CCR0 interrupt when 32768 is reached
        TA0CCR0 = 32767;
        TACCTL0 |= CCIE;

        // Generate a CCR1 interrupt if debounce delay expired.
        TACCTL1 |= CCIE;
}       // Setup_Timer


// =======================================================
//  Setup MSP430 ports
// =======================================================
void Setup_Ports(void)
{
        // P1.3 input with falling edge interrupt, others are output
        P1DIR = ~BUTTON;          // Button (P1.3) is input for Button
        P1SEL = BIT1 + BIT2;      // P1.1 and P1.2 used for UART
        P1SEL2 = BIT1 + BIT2;     // P1.1 and P1.2 used for UART
        P1OUT = BUTTON;                   // Select pullup for button
        P1REN = BUTTON;                   // Enable pullup for button
        P1IE  = BUTTON;                   // Button (P1.3)  interrupt enabled
        P1IES = BUTTON;                   // Falling edge sensitive
        P1IFG = 0x00;                     // All port 1 IFGs are cleared

        P2DIR = 0xFF;
        P2OUT = 0x00;

        P3DIR = 0xFF;
        P3OUT = 0x00;

}       // Setup_Ports


// =======================================================
//  Setup ADC10
// =======================================================
void Setup_ADC10(void)
{
        // Setup ADC10 to sample internal temperature sensor
        // Use ADC10 clock /4 for sample clock
        ADC10CTL1 = INCH_10 + ADC10DIV_3;

        // Enable ADC10 and reference and use it, 64 sample clocks, enable IRQ
        ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE;
}       // SetupADC10


// =======================================================
//  Setup UART for Host Communication
// =======================================================
void Setup_UART(void)
{
        // Setup USCIA as UART with 9600 bd
        UCA0CTL1 |= UCSSEL_1;                                           // CLK = ACLK
        UCA0BR0 = 0x03;                                                         // 32kHz/9600 = 3.41
        UCA0BR1 = 0x00;                                                         //     - " -
        UCA0MCTL = UCBRS1 + UCBRS0;                                     // Modulation UCBRSx = 3
```

```
        UCA0CTL1 &= ~UCSWRST;                                           // **Initialize USCI state machine**
}       // Setup_UART


// ================================================
//  Process samples, calculate the temperatures
// ================================================
void ProcessCurrentSample(unsigned short smpl, MEASUREMENT_INFO_T* x)
{
        // Ringbuffer definition to hold values for average calculation
        #define RINGBUFFERSIZE 16
        static unsigned short RingBuffer[RINGBUFFERSIZE];
        static unsigned char Idx = 0;
        unsigned short i;

        RingBuffer[Idx++] = smpl;    // add new value to ring buffer, increment buffer index

        if(Idx >= RINGBUFFERSIZE)    // handle buffer overflow
        {
                Idx = 0;                // roll over index when buffer is full
        }

        x->rs.A = smpl;

        // Temperature in Celsius
        // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
        x->rs.B = ((smpl - 673) * 423) >> 10;

        // Temperature in Fahrenheit
        // oF = ((A10/1024)*1500mV)-923mV)*1/1.97mV = A10*761/1024 - 468
        x->rs.F = ((smpl - 630) * 761) >> 10;

        // calculate average value
        x->rs.aA = 0;                                                   // set to 0

        for(i = RINGBUFFERSIZE; i > 0; i--)
        {
                x->rs.aA += RingBuffer[i];                      // accumulate stored values
        }
        x->rs.aA = x->rs.aA >> 4;                       // divide by 16

        // finally copy current record to transmit buffer
        memcpy(tr.b, x->b, MEASUREMENT_INFO_SIZE);
}       // ProcessCurrentSample



// ================================================
//    ISRs
// ================================================


// ================================================
//  Timer A0 CCR0 Interrupt Service Routine
// ================================================
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TA0_ISR(void)
{
        // Generate OneSecondFlag every 2 s
        OneSecondFlag = 1;
        LPM3_EXIT;                                                                              // Exit LPM3 on reti
}       // TA0_ISR


// ================================================
//  Timer A1 Interrupt Vector (TAIV) handler
// ================================================
#pragma vector=TIMER0_A1_VECTOR
__interrupt void TA0_IV_ISR(void)
{
        switch( TA0IV )
        {
                case  2:                // CCR1 interrupt for debounce
                        DebounceFlag = 1;
                        TACCTL1 &= ~CCIE;
                break;

                case 10:         // Timer_A3 overflow
                        // Generate OneSecondFlag every 2 s
                        OneSecondFlag = 1;
                break;
        }
        LPM3_EXIT;                                                                              // Exit LPM3 on reti
}       // TA0_IV_ISR


// ================================================
//  Port 1 interrupt service routine
// ================================================
#pragma vector=PORT1_VECTOR
__interrupt void Port_1_ISR(void)
{
        // The only interrupt can come from P1.3 (Button)
        Port1IRQHappened = 1;
        P1IE  &= ~BUTTON;                                               // Button (P1.3) interrupt disabled
        LPM3_EXIT;                                                              // Exit LPM3 on reti
}       // Port_1_ISR
```

```
// ================================================
//  ADC10 interrupt service routine
// ================================================
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
        ADC10ReadyFlag = 1;
        LPM3_EXIT;                                              // Exit LPM3 on reti
}       // ADC10_ISR


// ================================================
//  USCIA0 Transmit interrupt service routine
// ================================================
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCIA0_TX_ISR (void)
{
        static unsigned short i;

        UCA0TXBUF = tr.b[i++];                                  // TX next byte
        if (i == sizeof tr - 1)                                 // TX over?
        {
                IE2 &= ~UCA0TXIE;                                       // Disable USCI_A0 TX interrupt
                i = 0;                                                  // Set index to 0
        }
}       // USCIA0_TX_ISR


// ================================================
//  Unused interrupt service routine
// ================================================
#pragma vector=TIMER1_A0_VECTOR
#pragma vector=TIMER1_A1_VECTOR
#pragma vector=COMPARATORA_VECTOR
#pragma vector=PORT2_VECTOR
#pragma vector=WDT_VECTOR
#pragma vector=USCIAB0RX_VECTOR
#pragma vector=NMI_VECTOR
__interrupt void Unused_ISR(void)
{
}
```