```
/*****************************************************************************
 *
 *
 * 1. Device starts up in LPM3 + blinking LED to indicate device is alive
 *    + Upon first button press, device transitions to application mode
 * 2. Application Mode
 *    + Continuously sample ADC Temp Sensor channel, compare result against
 *      initial value
 *    + Set PWM based on measured ADC offset: Red LED for positive offset, Green
 *      LED for negative offset
 *    + Transmit temperature value via TimerA  to PC
 *    + Button Press --> Calibrate using current temperature
 *                       Send character '°' via UART, notifying PC
 *****************************************************************************/

#include <msp430g2231.h>

#define     green_LED             BIT0
#define     red_LED               BIT6
#define     BUTTON                BIT3
#define     TXD                   BIT1                    // TXD on P1.1
#define     RXD                   BIT2                    // RXD on P1.2

#define     APP_STANDBY_MODE      0
#define     APP_APPLICATION_MODE  1

#define     TIMER_PWM_MODE        0
#define     TIMER_UART_MODE       1
#define     TIMER_PWM_PERIOD      2000
#define     TIMER_PWM_OFFSET      20

#define     TEMP_SAME             0
#define     TEMP_HOT              1
#define     TEMP_COLD             2

#define     TEMP_THRESHOLD        5

//   Conditions for 9600/4=2400 Baud SW UART, SMCLK = 1MHz
#define     Bitime_5              0x05*4                    // ~ 0.5 bit length + small
adjustment
#define     Bitime                13*4//0x0D

#define     UART_UPDATE_INTERVAL  1000 // Loops until byte is sent

unsigned char BitCnt; // Bit count, used when transmitting byte

unsigned char applicationMode = APP_STANDBY_MODE;
unsigned char timerMode = TIMER_PWM_MODE;

unsigned char tempMode;
unsigned char calibrateUpdate = 0;
unsigned char tempPolarity = TEMP_SAME;
unsigned int TXByte; // Value sent over UART when Transmit() is called

/* Using an 8-value moving average filter on sampled ADC values */
long tempMeasured[8];
unsigned char tempMeasuredPosition=0;
```

```
long tempAverage;

long tempCalibrated, tempDifference;

void InitializeLeds(void);
void InitializeButton(void);
void PreApplicationMode(void);                    // Blinks LED, waits for button press
void ConfigureAdcTempSensor(void);
void ConfigureTimerPwm(void);
void ConfigureTimerUart(void);
void Transmit(void);
void InitializeClocks(void);

void main(void)
{
  unsigned int uartUpdateTimer = UART_UPDATE_INTERVAL;
  unsigned char i;                          // Transmit value counter
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  InitializeClocks();
  InitializeButton();
  InitializeLeds();
  PreApplicationMode();                     // Blinks LEDs, waits for button press

  /* Application Mode begins */
  applicationMode = APP_APPLICATION_MODE;   // applicationMode = '1'
  ConfigureAdcTempSensor();
  ConfigureTimerPwm();

  __enable_interrupt();                     // Enable interrupts.

  /* Main Application Loop */
  while(1)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled

    /* Moving average filter out of 8 values to somewhat stabilize sampled ADC */
    tempMeasured[tempMeasuredPosition++] = ADC10MEM;
    if (tempMeasuredPosition == 8)
      tempMeasuredPosition = 0;
    tempAverage = 0;
    for (i = 0; i < 8; i++)
      tempAverage += tempMeasured[i];
    tempAverage >>= 3;                      // Divide by 8 to get average

    if ((--uartUpdateTimer == 0) || calibrateUpdate )
    {
      ConfigureTimerUart();
      if (calibrateUpdate)
      {
        TXByte = 248;                       // A character with high value, outside of temp
        range
        Transmit();
        calibrateUpdate = 0;
      }
      TXByte = (unsigned char)( (tempAverage * 423) / 1024 - 281 );
```

```
      Transmit();
      uartUpdateTimer = UART_UPDATE_INTERVAL;
      ConfigureTimerPwm();
    }

    tempDifference = tempAverage - tempCalibrated;
    if (tempDifference < -TEMP_THRESHOLD)
    {
      tempDifference = -tempDifference;
      tempPolarity = TEMP_COLD;
      P1OUT &= ~ red_LED;
    }
    else
    if (tempDifference > TEMP_THRESHOLD)
    {
      tempPolarity = TEMP_HOT;
      P1OUT &= ~ green_LED;
    }
    else
    {
      tempPolarity = TEMP_SAME;
      TACCTL0 &= ~CCIE;
      TACCTL1 &= ~CCIE;
      P1OUT &= ~(green_LED + red_LED);
    }

    if (tempPolarity != TEMP_SAME)
    {
      tempDifference <<= 3;
      tempDifference += TIMER_PWM_OFFSET;
      TACCR1 = ( (tempDifference) < (TIMER_PWM_PERIOD-1) ? (tempDifference) : (
      TIMER_PWM_PERIOD-1) );
      TACCTL0 |= CCIE;
      TACCTL1 |= CCIE;
    }
  }
}

void PreApplicationMode(void)
{
  P1OUT |= green_LED;                       // To enable the LED toggling effect
  P1OUT &= ~red_LED;

  BCSCTL1 |= DIVA_1;                        // ACLK = VLO/2 = 6 KHz
  BCSCTL3 |= LFXT1S_2;                      // VLO = 12kHz => ACLK
  TACCR0 = 1200;                            // capture compare reg value 0.2s
  TACTL = TASSEL_1 | MC_1;                  // Timer_A: ACLK selected as clock source,
                                            // Up mode: the timer counts up to TACCR0.
  TACCTL1 = CCIE + OUTMOD_3;                // Capture/compare interrupt enabled,
                                            // set/reset out mode: (figure 12-12)
                                            // The output is set when the timer counts to
                                            the TACCRx value.
                                            // It is reset when the timer counts to the
                                            TACCR0 value.
  TACCR1 = 600;                             // capture compare reg value 0.1s
  __bis_SR_register(LPM3_bits + GIE);       // LPM0 with interrupts enabled
}
```

```
void ConfigureAdcTempSensor(void)
{
  unsigned char i;
  /* Configure ADC Temp Sensor Channel */
  ADC10CTL1 = INCH_10 + ADC10DIV_3;         // Temp Sensor, ADC10OSC/4 (ADC10SSEL = '00')
  // VR+ = VREF+ and VR- = VSS, ADC10 sample-and-hold time = 64 × ADC10CLKs,
  // Reference generator on, ADC10 on, ADC10 interrupt enable = ADC10IFG
  // is set when conversion results are loaded into ADC10MEM
  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE;
  __delay_cycles(1000);                     // Wait for ADC Ref to settle
  ADC10CTL0 |= ENC + ADC10SC;               // Enable conversion, Start conversion.
  __bis_SR_register(CPUOFF + GIE);          // LPM0 with interrupts enabled
  tempCalibrated = ADC10MEM;
  for (i=0; i < 8; i++)
    tempMeasured[i] = tempCalibrated;
  tempAverage = tempCalibrated;
}

void ConfigureTimerPwm(void)
{
  timerMode = TIMER_PWM_MODE;

  TACCR0 = TIMER_PWM_PERIOD;                //
  TACTL = TASSEL_2 | MC_1;                  // TACLK = SMCLK, Up mode.
  TACCTL0 = CCIE;
  TACCTL1 = CCIE + OUTMOD_3;                // TACCTL1 Capture Compare
  TACCR1 = 1;
}

void ConfigureTimerUart(void)
{
  timerMode = TIMER_UART_MODE;               // Configure TimerA0 UART TX

  CCTL0 = OUT;                               // TXD Idle as Mark
  TACTL = TASSEL_2 + MC_2 + ID_3;            // SMCLK/8, continuous mode
  P1SEL |= TXD + RXD;                        // P1.1 & 2 TA0, rest GPIO
  P1DIR |= TXD;                              // P1.1 output,  other inputs
}

// Function Transmits Character from TXByte
void Transmit()
{
  BitCnt = 0xA;                              // Load Bit counter, 8data + ST/SP
  while (CCR0 != TAR)                        // Prevent async capture
    CCR0 = TAR;                              // Current state of TA counter
  CCR0 += Bitime;                            // Set time till first bit
  TXByte |= 0x100;                           // Add mark stop bit to TXByte
  TXByte = TXByte << 1;                      // Add start bit (which is logical 0)
  CCTL0 =  CCIS0 + OUTMOD0 + CCIE;           // TXD = mark = idle, Set signal, intial value,
  enable interrupt
  while ( CCTL0 & CCIE );                    // Wait for TX completion
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
```

```
{
  if (timerMode == TIMER_UART_MODE)
  {
    CCR0 += Bitime;                          // Add Offset to CCR0
    if (CCTL0 & CCIS0)                       // TX on CCI0B?
    {
      if ( BitCnt == 0)
        CCTL0 &= ~ CCIE;                     // If all bits TXed, disable interrupt
      else
      {
        CCTL0 |=  OUTMOD2;                   // TX Space
        if (TXByte & 0x01)
        CCTL0 &= ~ OUTMOD2;                  // TX Mark
        TXByte = TXByte >> 1;
        BitCnt --;
      }
    }
  }
  else
  {
    if (tempPolarity == TEMP_HOT)
      P1OUT |= red_LED;
    if (tempPolarity == TEMP_COLD)
      P1OUT |= green_LED;
    TACCTL0 &= ~CCIFG;
  }
}

#pragma vector=TIMERA1_VECTOR
__interrupt void ta1_isr(void)
{
  TACCTL1 &= ~CCIFG;
  if (applicationMode == APP_APPLICATION_MODE)
    P1OUT &= ~(green_LED + red_LED);
  else
    P1OUT ^= (green_LED + red_LED);
}

void InitializeClocks(void)
{
  BCSCTL1 = CALBC1_1MHZ;                     // Set range
  DCOCTL = CALDCO_1MHZ;
}

void InitializeButton(void)                  // Configure Push Button
{
  P1DIR &= ~BUTTON;                          // The pin is an input
  P1OUT |= BUTTON;                           // pulled up
  P1REN |= BUTTON;                           // pullup/pulldown resistor enabled
  P1IES |= BUTTON;                           // Flag triggered by the pin going from '1' to '0'
  P1IFG &= ~BUTTON;                          // Clearing flag
  P1IE |= BUTTON;                            // Interrupt enabled
}

void InitializeLeds(void)
{
  P1DIR |= green_LED + red_LED;              // P1.0 and P1.6 are
```

```
  output
  P1OUT &= ~(green_LED + red_LED);          // P1.0 and P1.6 set to '0', leds off
}

/* ********************************************************
 * Port Interrupt for Button Press
 * 1. During standby mode: to exit and enter application mode
 * 2. During application mode: to recalibrate temp sensor
 * ******************************************************** */
#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
  P1IFG = 0;
  P1IE &= ~BUTTON;            /* Debounce */
  WDTCTL = WDT_ADLY_250;           // WDTCTL = WDT_ADLY_250
  // WDT_ADLY_250 sets the following bits
  // (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS0) 250ms interval:
  //    • WDTPW: WDT password
  //    • WDTTMSEL: Selects interval timer mode
  //    • WDTCNTCL: Clears count value
  //    • WDTSSEL: WDT clock source select
  IFG1 &= ~WDTIFG;                /* clear interrupt flag */
  IE1 |= WDTIE;                   // Enable WDT interrupt

  if (applicationMode == APP_APPLICATION_MODE)
  {
    tempCalibrated = tempAverage;
    calibrateUpdate  = 1;
  }
  else
  {
    applicationMode = APP_APPLICATION_MODE; // Switch from STANDBY to APPLICATION MODE
    __bic_SR_register_on_exit(LPM3_bits);
  }
}

#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{
    IE1 &= ~WDTIE;                    /* disable interrupt */
    IFG1 &= ~WDTIFG;                  /* clear interrupt flag */
    WDTCTL = WDTPW + WDTHOLD;         /* put WDT back in hold state */
    P1IE |= BUTTON;              /* Debouncing complete */
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Return to active mode
}
```