```
/*******************************************************************************
 *                  MSP-EXP430G2-LaunchPad User Experience Application
 *
 * 1. Device sTA1rts 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 UART to PC
 *    + Button Press --> Calibrate using current temperature
 *                       Send character '°' via UART, notifying PC
 *******************************************************************************/

#include  "msp430F5529.h"

#define     LED0                    BIT0
#define     LED1                    BIT6
#define     LED_DIR                 P1DIR
#define     LED_OUT                 P1OUT


#define     BUTTON                  BIT3
#define     BUTTON_OUT              P1OUT
#define     BUTTON_DIR              P1DIR
#define     BUTTON_IN               P1IN
#define     BUTTON_IE               P1IE
#define     BUTTON_IES              P1IES
#define     BUTTON_IFG              P1IFG
#define     BUTTON_REN              P1REN

#define     TXD                     BIT1                        // TXD on P1.1
#define     RXD                     BIT2                        // RXD on P1.2

#define     APP_STA1NDBY_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

```

```
 57
 58 unsigned char BitCnt;
 59
 60
 61 unsigned char applicationMode = APP_STA1NDBY_MODE;
 62 unsigned char timerMode = TIMER_PWM_MODE;
 63
 64 unsigned char tempMode;
 65 unsigned char calibrateUpdate = 0;
 66 unsigned char tempPolarity = TEMP_SAME;
 67 unsigned int TXByte;
 68
 69 /* Using an 8-value moving average filter on sampled ADC values */
 70 long tempMeasured[8];
 71 unsigned char tempMeasuredPosition=0;
 72 long tempAverage;
 73
 74 long tempCalibrated, tempDifference;
 75
 76
 77
 78 void InitializeLeds(void);
 79 void InitializeButton(void);
 80 void PreApplicationMode(void);                    // Blinks LED, waits for button press
 81 void ConfigureAdcTempSensor(void);
 82 void ConfigureTimerPwm(void);
 83 void ConfigureTimerUart(void);
 84 void Transmit(void);
 85 void InitializeClocks(void);
 86
 87 void main(void)
 88 {
 89   unsigned int uartUpdateTimer = UART_UPDATE_INTERVAL;
 90   unsigned char i;
 91   WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
 92
 93   InitializeClocks();
 94   InitializeButton();
 95   InitializeLeds();
 96   PreApplicationMode();                     // Blinks LEDs, waits for button press
 97
 98   /* Application Mode begins */
 99   applicationMode = APP_APPLICATION_MODE;
100   ConfigureAdcTempSensor();
101   ConfigureTimerPwm();
102
103   __enable_interrupt();                     // Enable interrupts.
104
105
106   /* Main Application Loop */
107   while(1)
108   {
109     ADC12CTL0 |= ENC + ADC12SC;             // Sampling and conversion sTA1rt
110     __bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled
111
112
113     /* Moving average filter out of 8 values to somewhat sTA1bilize sampled ADC */
```

```
114     tempMeasured[tempMeasuredPosition++] = ADC12MEM;
115     if (tempMeasuredPosition == 8)
116       tempMeasuredPosition = 0;
117     tempAverage = 0;
118     for (i = 0; i < 8; i++)
119       tempAverage += tempMeasured[i];
120     tempAverage >>= 3;                      // Divide by 8 to get average
121
122     if ((--uartUpdateTimer == 0) || calibrateUpdate )
123     {
124       ConfigureTimerUart();
125       if (calibrateUpdate)
126       {
127         TXByte = 248;                       // A character with high value, outside of temp
    range
128         Transmit();
129         calibrateUpdate = 0;
130       }
131       TXByte = (unsigned char)( ((tempAverage - 630) * 761) / 1024 );
132       Transmit();
133       uartUpdateTimer = UART_UPDATE_INTERVAL;
134       ConfigureTimerPwm();
135     }
136
137
138     tempDifference = tempAverage - tempCalibrated;
139     if (tempDifference < -TEMP_THRESHOLD)
140     {
141       tempDifference = -tempDifference;
142       tempPolarity = TEMP_COLD;
143       LED_OUT &= ~ LED1;
144     }
145     else
146     if (tempDifference > TEMP_THRESHOLD)
147     {
148       tempPolarity = TEMP_HOT;
149       LED_OUT &= ~ LED0;
150     }
151     else
152     {
153       tempPolarity = TEMP_SAME;
154       TA1TA1CCTL0 &= ~CCIE;
155       TA1CCTL1 &= ~CCIE;
156       LED_OUT &= ~(LED0 + LED1);
157     }
158
159     if (tempPolarity != TEMP_SAME)
160     {
161       tempDifference <<= 3;
162       tempDifference += TIMER_PWM_OFFSET;
163       TA1CCR1 = ( (tempDifference) < (TIMER_PWM_PERIOD-1) ? (tempDifference) :
    (TIMER_PWM_PERIOD-1) );
164       TA1TA1CCTL0 |= CCIE;
165       TA1CCTL1 |= CCIE;
166     }
167   }
168 }
```

```
169
170 void PreApplicationMode(void)
171 {
172   LED_DIR |= LED0 + LED1;
173   LED_OUT |= LED0;                          // To enable the LED toggling effect
174   LED_OUT &= ~LED1;
175
176   RTCCTL1 |= DIVA_1;                        // ACLK/2
177   RTCCTL3 |= LFXT1S_2;                      // ACLK = VLO
178
179   TA1TA1CCR0 = 1200;                             //
180   TA11CTL = TA1SSEL_1 | MC_1;                  // TA1CLK = SMCLK, Up mode.
181   TA1CCTL1 = CCIE + OUTMOD_3;                 // TA1CCTL1 Capture Compare
182   TA1CCR1 = 600;
183   __bis_SR_register(LPM3_bits + GIE);          // LPM0 with interrupts enabled
184 }
185
186 void ConfigureAdcTempSensor(void)
187 {
188   unsigned char i;
189   /* Configure ADC Temp Sensor Channel */
190   ADC12CTL1 = INCH_10 + ADC12DIV_3;         // Temp Sensor ADC12CLK/4
191   ADC12CTL0 = SREF_1 + ADC12SHT03 + REFON + ADC12ON + ADC12IE;
192   __delay_cycles(1000);                     // Wait for ADC Ref to settle
193   ADC12CTL0 |= ENC + ADC12SC;               // Sampling and conversion sTA1rt
194   __bis_SR_register(CPUOFF + GIE);          // LPM0 with interrupts enabled
195   tempCalibrated = ADC12MEM;
196   for (i=0; i < 8; i++)
197     tempMeasured[i] = tempCalibrated;
198   tempAverage = tempCalibrated;
199 }
200
201
202 void ConfigureTimerPwm(void)
203 {
204   timerMode = TIMER_PWM_MODE;
205
206   TA1TA1CCR0 = TIMER_PWM_PERIOD;                             //
207   TA1CTL = TA1SSEL_2 | MC_1;                  // TA1CLK = SMCLK, Up mode.
208   TA1TA1CCTL0 = CCIE;
209   TA1CCTL1 = CCIE + OUTMOD_3;                // TA1CCTL1 Capture Compare
210   TA1CCR1 = 1;
211 }
212
213 void ConfigureTimerUart(void)
214 {
215   timerMode = TIMER_UART_MODE;               // Configure TimerA0 UART TX
216
217   TA1CCTL0 = OUT;                               // TXD Idle as Mark
218   TA1CTL = TA1SSEL_2 + MC_2 + ID_3;            // SMCLK/8, continuous mode
219   P1SEL |= TXD + RXD;                        //
220   P1DIR |= TXD;                              //
221 }
222
223 // Function Transmits Character from TXByte
224 void Transmit()
225 {
```

```
226   BitCnt = 0xA;                               // Load Bit counter, 8daTA1 + ST/SP
227   while (TA1CCR0 != TA1R)                         // Prevent async capture
228     TA1CCR0 = TA1R;                               // Current sTA1te of TA1 counter
229   TA1CCR0 += Bitime;                        // Some time till first bit
230   TXByte |= 0x100;                           // Add mark stop bit to TXByte
231   TXByte = TXByte << 1;                     // Add space sTA1rt bit
232   TA1CCTL0 =  CCIS0 + OUTMOD0 + CCIE;           // TXD = mark = idle
233   while ( TA1CCTL0 & CCIE );                    // Wait for TX completion
234 }
235
236
237
238 // Timer A0 interrupt service routine
239 #pragma vector=TIMERA0_VECTOR
240 __interrupt void Timer_A (void)
241 {
242   if (timerMode == TIMER_UART_MODE)
243   {
244     TA1CCR0 += Bitime;                            // Add Offset to TA1CCR0
245     if (TA1TA1CCTL0 & CCIS0)                        // TX on CCI0B?
246     {
247       if ( BitCnt == 0)
248         TA1CCTL0 &= ~ CCIE;                         // All bits TXed, disable interrupt
249       else
250       {
251         TA1CCTL0 |=  OUTMOD2;                     // TX Space
252         if (TXByte & 0x01)
253         TA1CCTL0 &= ~ OUTMOD2;                    // TX Mark
254         TXByte = TXByte >> 1;
255         BitCnt --;
256       }
257     }
258   }
259   else
260   {
261     if (tempPolarity == TEMP_HOT)
262       LED_OUT |= LED1;
263     if (tempPolarity == TEMP_COLD)
264       LED_OUT |= LED0;
265     TA1TA1CCTL0 &= ~CCIFG;
266   }
267 }
268
269 #pragma vector=TIMERA1_VECTOR
270 __interrupt void TA11_isr(void)
271 {
272   TA1CCTL1 &= ~CCIFG;
273   if (applicationMode == APP_APPLICATION_MODE)
274     LED_OUT &= ~(LED0 + LED1);
275   else
276     LED_OUT ^= (LED0 + LED1);
277
278 }
279
280 void InitializeClocks(void)
281 {
282
```

```
283   //RTCCTL1 = CALBC1_1MHZ;                    // Set range
284  //DCOCTL = CALDCO_1MHZ;
285   RTCCTL2 &= ~(DIVS_3);                       // SMCLK = DCO / 8 = 1MHz
286 }
287
288 void InitializeButton(void)                 // Configure Push Button
289 {
290   BUTTON_DIR &= ~BUTTON;
291   BUTTON_OUT |= BUTTON;
292   BUTTON_REN |= BUTTON;
293   BUTTON_IES |= BUTTON;
294   BUTTON_IFG &= ~BUTTON;
295   BUTTON_IE |= BUTTON;
296 }
297
298
299 void InitializeLeds(void)
300 {
301   LED_DIR |= LED0 + LED1;
302   LED_OUT &= ~(LED0 + LED1);
303 }
304
305 /* ************************************************************
306  * Port Interrupt for Button Press
307  * 1. During sTA1ndby mode: to exit and enter application mode
308  * 2. During application mode: to recalibrate temp sensor
309  * ************************************************************ */
310 #pragma vector=PORT1_VECTOR
311 __interrupt void PORT1_ISR(void)
312 {
313   BUTTON_IFG = 0;
314   BUTTON_IE &= ~BUTTON;            /* Debounce */
315   WDTCTL = WDT_ADLY_250;
316   IFG1 &= ~WDTIFG;                 /* clear interrupt flag */
317   IE1 |= WDTIE;
318
319   if (applicationMode == APP_APPLICATION_MODE)
320   {
321     tempCalibrated = tempAverage;
322     calibrateUpdate  = 1;
323   }
324   else
325   {
326     applicationMode = APP_APPLICATION_MODE; // Switch from STA1NDBY to APPLICATION MODE
327     __bic_SR_register_on_exit(LPM3_bits);
328   }
329 }
330
331 #pragma vector=WDT_VECTOR
332 __interrupt void WDT_ISR(void)
333 {
334     IE1 &= ~WDTIE;                   /* disable interrupt */
335     IFG1 &= ~WDTIFG;                 /* clear interrupt flag */
336     WDTCTL = WDTPW + WDTHOLD;        /* put WDT back in hold sTA1te */
337     BUTTON_IE |= BUTTON;             /* Debouncing complete */
338 }
339
```

```
340 // ADC12 interrupt service routine
341 #pragma vector=ADC12_VECTOR
342 __interrupt void ADC12_ISR (void)
343 {
344   __bic_SR_register_on_exit(CPUOFF);        // Return to active mode
345 }
346
347
348
349
```