```
1
2  /*****************************************************************************
3   *
4   * main.c
5   *
6   * Out of Box Demo for the MSP-EXP430FR2311 Light sensing demo with FRAM write
7   *
8   * Main loop, initialization, and interrupt service routines
9   *
10  *   Description: Sample Op-Amp output with ADC10, PWM LED using Timer1 B1
11  *   with ADC value, store current ADC value to FRAM and enter LPM3, wake up
12  *   every ~50ms to start a new ADC conversion.
13  *   NOTE: Running this example for extended periods will impact the FRAM
14  *   endurance.
15  *   ACLK = REFO= ~32768kHz, MCLK = SMCLK = ~1.0MHz
16  *
17  *   Tested On: MSP430FR2311
18  *            -----------------
19  *        /|\|                 |
20  *         | |        P1.2/OA0-|<---External Pad
21  *         --|RST     P1.3/OA0O|--->A3(Internal)
22  *           |        P1.4/OA0+|<---External Pad
23  *           |                 |
24  *           |       P2.0/TB1.1|---> LED
25  *           |                 |
26  *
27  * December 2015
28  * J. Collins
29  *
30  *****************************************************************************/
31
32 #include "driverlib.h"
33
34 //void initPWM(void);
35 void initSleepTimer(uint16_t);
36 //void initSACOA(void);
37 //void initADC10(int x);
38 void initGPIO(void);
39 //void initTRIOA(void);
40 //void initRtc(void);
41
42 #define CALADC_15V_30C *((unsigned int *)0x1A1A)
43 #define CALADC_15V_85C *((unsigned int *)0x1A1C)
44
45 volatile float temp, IntDegF, IntDegC;
46
47
48 /* FRAM Variable that stores ADC results*/
49 //#pragma PERSISTENT(ADC_Conversion_Result)
50 unsigned int ADC_Conversion_Result = 0;
51 #pragma PERSISTENT(ADJ64)
52 unsigned int ADJ64=0xffff, adflag=0, n;          // ADC conversion result
53 #pragma PERSISTENT(ADCTEMP)
54 unsigned int ADCTEMP=256, rtc_cnt=0;
55 #pragma PERSISTENT(RTC_MOD)
56 unsigned int RTC_MOD=0xffff;
57 #pragma PERSISTENT(SMK_TRC)
58 unsigned int SMK_TRC=0xffff;
59 unsigned int CurTemp, k;
60 unsigned int buffer30[32],buf30[32],j=0, i=0, predata, postdata, kdata, ScanCnt=0;
61 unsigned int InitFlag=0, TmCnt=0, ADC32=0, t30, t85, rtccount, rtcdelt, rtccnt1;
62 unsigned int ScanFlag=0, TempFlag=0, AlarmFlag=0, AlarmCnt=0, BatCnt=0, BatFlag=0;
```

```
63 unsigned int BatLowFlag=0, AlarmAct=0, BatVolt, ManuIn, ManuInFlag=0, ManuInCnt=0;
64 const unsigned int AlarmValue=118, ActCnt=3, BatChkCnt=60, BatLowValue=750; //750=2.2v
65 //
66 //
67 int main(void) {
68     //Stop WDT
69     WDT_A_hold(WDT_A_BASE);
70
71
72     initGPIO();
73     PMM_disableSVSH ();
74 /*
75     RTC_init(RTC_BASE, 0x2000, RTC_CLOCKPREDIVIDER_1);
76 //    RTC_setModulo(RTC_BASE, RTC_MOD);
77     RTCCTL |= RTCSR;
78     RTC_enableInterrupt(RTC_BASE, RTCIV__RTCIFG);
79     RTC_start(RTC_BASE, RTC_CLOCKSOURCE_VLOCLK);
80     TB0CTL = 0x0000;   //stop Timer B0
81     CSCTL4 = 0x0000;   //SELA = 0
82     */
83     while(1){
84
85         TmCnt++;
86 //        RTC_start(RTC_BASE, RTC_CLOCKSOURCE_VLOCLK);
87 //        __bis_SR_register(LPM3_bits + GIE);
88         initSleepTimer(65535);  //
89         __bis_SR_register(LPM3_bits + GIE);
90         PM5CTL0 &= ~LOCKLPM5;
91
92     }
93
94 }
95
96 void initSleepTimer(uint16_t period){
97     //Start timer
98     Timer_B_clearTimerInterrupt(TIMER_B0_BASE);
99
100    Timer_B_initUpModeParam param = {0};
101    param.clockSource = TIMER_B_CLOCKSOURCE_ACLK;
102    param.clockSourceDivider = TIMER_B_CLOCKSOURCE_DIVIDER_1;
103    param.timerPeriod = period;
104    param.timerInterruptEnable_TBIE = TIMER_B_TBIE_INTERRUPT_DISABLE;
105    param.captureCompareInterruptEnable_CCR0_CCIE =
106        TIMER_B_CAPTURECOMPARE_INTERRUPT_ENABLE;
107    param.timerClear = TIMER_B_DO_CLEAR;
108    param.startTimer = true;
109    Timer_B_initUpMode(TIMER_B0_BASE, &param);
110 }
111
112
113 void initGPIO(void){
114
115 //    P1DIR = 0xFC;   //p1 IO-In
116 //    P1OUT = 0xFF;  //p1 pull-up
117    P1DIR = 0x1C;   //p1 IO-In
118    P1OUT = 0x1F;  //p1 pull-up
119
120    P1REN = 0x1D;  //P1.0 pull high
121    P1SEL0 = 0xe2;
122    P1SEL1 = 0xe2;
123        P2DIR = 0x7f;
124        P2OUT = 0xe0;  //P2.5~6  low active
```

```
125         P2REN = 0x80;
126         TRI_disable(TRI0_BASE);
127     PM5CTL0 &= ~LOCKLPM5;
128
129 }
130
131 // ADC interrupt service routine
132 #if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
133 #pragma vector=ADC_VECTOR
134 __interrupt void ADC_ISR(void)
135 #elif defined(__GNUC__)
136 void __attribute__ ((interrupt(ADC_VECTOR))) ADC_ISR (void)
137 #else
138 #error Compiler not supported!
139 #endif
140 {
141     //Get previous write protection setting
142     uint8_t state = HWREG8(SYS_BASE + OFS_SYSCFG0_L);
143 #ifdef DFWP
144     uint8_t wp = DFWP | PFWP;
145 #else
146     uint8_t wp = PFWP;
147 #endif
148
149 #ifdef FRWPPW
150     HWREG16(SYS_BASE + OFS_SYSCFG0) = FWPW | (state & ~wp);
151 #else
152     HWREG8(SYS_BASE + OFS_SYSCFG0_L) &= ~wp;
153 #endif
154
155     //ADC_Conversion_Result = (ADCMEM0 >> 1);       // Get conversion result and scale for
    lower light level situations (divide by 2)
156     ADC_Conversion_Result = ADCMEM0;
157     //Restore previous write protection setting
158 #ifdef FRWPPW
159     HWREG16(SYS_BASE + OFS_SYSCFG0) = FWPW | state;
160 #else
161     HWREG8(SYS_BASE + OFS_SYSCFG0_L) = state;
162 #endif
163      adflag=0;
164  // __bic_SR_register_on_exit(LPM3_bits);              // Sleep Timer Exits LPM3
165 }
166
167 //******************************************************************************
168 //
169 //This is the Timer B0 interrupt vector service routine.
170 //
171 //******************************************************************************
172 #if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
173 #pragma vector=TIMER0_B0_VECTOR
174 __interrupt
175 #elif defined(__GNUC__)
176 __attribute__((interrupt(TIMER0_B0_VECTOR)))
177 #endif
178 void TIMER0_B0_ISR(void)
179 {
180     __bic_SR_register_on_exit(LPM3_bits);              // Sleep Timer Exits LPM3
181 }
182 //RTC_VECTOR
183 //******************************************************************************
184 //
185 //This is the RTC interrupt vector service routine.
```

```
186 //
187 //*****************************************************************************
188
189 #if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
190 #pragma vector=RTC_VECTOR
191 __interrupt
192 #elif defined(__GNUC__)
193 __attribute__((interrupt(RTC_VECTOR)))
194 #endif
195 void RTC_ISR(void)
196 {
197     RTC_clearInterrupt(RTC_BASE,
198                        RTCIV__RTCIFG);
199     rtc_cnt++;
200     if(rtc_cnt >0)
201     {
202         rtc_cnt = 0;
203         __bic_SR_register_on_exit(LPM3_bits); // Sleep Timer Exits LPM3
204
205     }
206 }
207
208
209
210
```