```
#include <msp430.h>

#define CALADC_15V_30C  *((unsigned int *)0x1A1A)               // Temperature Sensor
Calibration-30 C //6682                                         // See device datasheet for TLV table
memory mapping //6684

#define CALADC_15V_85C  *((unsigned int *)0x1A1C)               // Temperature Sensor
Calibration-High Temperature (85 for Industrial, 105 for Extended)




volatile long temp1;

volatile float IntDegF1,light;

volatile float IntDegC1,motion;


volatile long temp2;

volatile float IntDegF2;

volatile float IntDegC2;


char result[100];

int count;


void uart_init(void);

void ConfigClocks(void);

void strreverse(char* begin, char* end);

void itoa(int value, char* str, int base);

void Software_Trim();

void port_init();

void ConfigureAdc_temp();
```

```
void ConfigureAdc_motion();
void ConfigureAdc_light();
void ConfigureAdc_Gas_sensor();
void initialize_Adc();

void main(void)
{
        WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;
    int m=0;

    ConfigClocks();
    port_init();
    uart_init();
    //spi_init();
    //lcd_init();

    _delay_cycles(5);              // Wait for ADC Ref to settle

    while(1){
       //Transmit a check byte B
            if(m == 0){
               _delay_cycles(20000);

            int acount =0;
```

```
            result[acount]='B';


            while((UCA1IFG & UCTXIFG)==0);
                UCA1TXBUF = result[acount] ;           //Transmit the received data.


             m++;




          if(m==1){
        P6DIR |= BIT3;                          // Set P6.0/LED to output direction
        P6OUT &= ~BIT3;
        //initialize_Adc();
        PMMCTL0_H = PMMPW_H;                          // Unlock the PMM registers read 2.2.8 &
2.2.9 form the manual
        PMMCTL2 |= INTREFEN | TSENSOREN | REFVSEL_0;              // Enable internal 1.5V
reference and temperature sensor


        ConfigureAdc_temp();
        ADCCTL0 |= ADCENC + ADCSC +ADCMSC;      // Converter Enable, Sampling/conversion
start
        while((ADCCTL0 & ADCIFG) == 0);      // check the Flag, while its low just wait
        _delay_cycles(200000);


        temp1 = ADCMEM0;                // read the converted data into a variable
        ADCCTL0 &= ~ADCIFG;
        IntDegC1 = (temp1-CALADC_15V_30C)*(85-30)/(CALADC_15V_85C-CALADC_15V_30C)+30;
```

```
            if (IntDegC1<30){
               P6OUT |= BIT3;
               //P6OUT &= ~BIT1;
            }

            else if (IntDegC1>30)
              P6OUT |= BIT3;
              //P6OUT &= ~BIT0;


        itoa(IntDegC1,result,10);
        acount =0;
        while(result[acount]!='\0')
          {
              while((UCA1IFG & UCTXIFG)==0);                    //Wait Unitl the UART
transmitter is ready //UCTXIFG
                          UCA1TXBUF = result[acount++] ;          //Transmit the received data.
          }
          m=2;


      if(m==2){
        P6DIR |= BIT2;                          // Set P6.0/LED to output direction
        P6OUT &= ~BIT2;
        ConfigureAdc_Gas_sensor();
```

```
            ADCCTL0 |= ADCENC + ADCSC +ADCMSC;       // Converter Enable,
Sampling/conversion start

            while((ADCCTL0 & ADCIFG) == 0);          // check the Flag, while its low just wait

            _delay_cycles(200000);


            temp1 = ADCMEM0;                 // read the converted data into a variable

            ADCCTL0 &= ~ADCIFG;

            IntDegC1 =
(temp1-CALADC_15V_30C)*(85-30)/(CALADC_15V_85C-CALADC_15V_30C)+30;


                if (IntDegC1<30){

                   P6OUT |= BIT2;

                   //P6OUT &= ~BIT1;

                }


                else if (IntDegC1>30)

                  P6OUT |= BIT2;

                  //P6OUT &= ~BIT0;



            itoa(IntDegC1,result,10);

            acount =0;

            while(result[acount]!='\0')

              {

                  while((UCA1IFG & UCTXIFG)==0);                    //Wait Unitl the UART
transmitter is ready //UCTXIFG

                      UCA1TXBUF = result[acount++] ;            //Transmit the received data.
```

```
                }

            m=3;

        if(m==3){

          ConfigureAdc_light();


          P6DIR |= BIT1;                            // Set P6.0/LED to output direction

           P6OUT &= ~BIT1;

                ADCCTL0 |= ADCENC + ADCSC +ADCMSC;     // Converter Enable,
Sampling/conversion start

                while((ADCCTL0 & ADCIFG) == 0);     // check the Flag, while its low just wait

                _delay_cycles(200000);


               light = ADCMEM0;              // read the converted data into a variable

                ADCCTL0 &= ~ADCIFG;

               // IntDegC1 =
(temp1-CALADC_15V_30C)*(85-30)/(CALADC_15V_85C-CALADC_15V_30C)+30;


                   if (light<200){

                     P6OUT |= BIT1;

                     //P6OUT &= ~BIT1;

                   }


                   else if (light>200)

                    P6OUT |= BIT1;

                    //P6OUT &= ~BIT0;
```

```
            itoa(light,result,10);
            acount =0;
            while(result[acount]!='\0')
               {
                  while((UCA1IFG & UCTXIFG)==0);                    //Wait Unitl the UART transmitter is ready //UCTXIFG
                        UCA1TXBUF = result[acount++] ;          //Transmit the received data.
               }
            m=4;

      if(m==4){
         ConfigureAdc_motion();
         // Configure GPIO
           P6DIR |= BIT0;                          // Set P6.0/LED to output direction
           P6OUT &= ~BIT0;                           // P6.0 LED off

           // Configure P1.1 as input (for motion sensor)
           P1DIR &= ~BIT1;
           P1REN |= BIT1; // Enable pull-up/pull-down resistor
           P1OUT |= BIT1; // Select pull-up resistor


         if (P1IN & BIT1)
         {
            // Motion detected, turn on LED
            P6OUT |= BIT0;
```

```
    motion = 1;
    P1DIR |= BIT6 | BIT7;                   // P1.6 and P1.7 output
    P1SEL1 |= BIT6 | BIT7;                  // P1.6 and P1.7 options select

      // Disable the GPIO power-on default high-impedance mode to activate
      // previously configured port settings
      PM5CTL0 &= ~LOCKLPM5;

      TB0CCR0 = 128;                        // PWM Period/2
      TB0CCTL1 = OUTMOD_6;                  // TBCCR1 toggle/set
      TB0CCR1 = 32;                         // TBCCR1 PWM duty cycle
      TB0CCTL2 = OUTMOD_6;                  // TBCCR2 toggle/set
      TB0CCR2 = 96;                         // TBCCR2 PWM duty cycle
      TB0CTL = TBSSEL_1 | MC_3;             // ACLK, up-down mode

    __delay_cycles(5000);
}
else
{
    // No motion, turn off LED
    motion = 0;
    P6OUT &= ~BIT0;
    P1DIR &= ~BIT6;                 // P1.6 and P1.7 output
    P1SEL1 &= ~BIT6;                // P1.6 and P1.7 options select
    P1DIR &= ~BIT7;                 // P1.6 and P1.7 output
    P1SEL1 &= ~BIT7;                // P1.6 and P1.7 options select
```

```
            }


                                itoa(motion,result,10);

                                acount =0;

                                while(result[acount]!='\0')

                                   {

                                      while((UCA1IFG & UCTXIFG)==0);                    //Wait Unitl
the UART transmitter is ready //UCTXIFG

                                          UCA1TXBUF = result[acount++] ;          //Transmit the
received data.

                                      }

                               m=1;


         }

                }//m=4

              }

      }

          }//m=1


}//while

      }//main



void uart_init(void){

          UCA1CTLW0 |= UCSWRST;

          UCA1CTLW0 |= UCSSEL__SMCLK;

          UCA1BRW = 8;                        // 115200
```

```
        UCA1MCTLW = 0xD600;
        UCA1CTLW0 &= ~UCSWRST;                  // Initialize eUSCI
        UCA1IE |= UCRXIE;                       // Enable USCI_A0 RX interrupt
}

void ConfigClocks(void)
{

        CSCTL3 = SELREF__REFOCLK;               // Set REFO as FLL reference source
        CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_0;// DCOFTRIM=3,
DCO Range = 1MHz
        CSCTL2 = FLLD_0 + 30;                   // DCODIV = 1MHz
        __delay_cycles(3);
        __bic_SR_register(SCG0);                // Enable FLL
   Software_Trim();                     // Software Trim to get the best DCOFTRIM value
        CSCTL4 = SELMS__DCOCLKDIV | SELA__REFOCLK; // set default REFO(~32768Hz) as
ACLK source, ACLK = 32768Hz
                                // default DCODIV as MCLK and SMCLK source

}

void strreverse(char* begin, char* end)         // Function to reverse the order of the ASCII char
array elements
{
        char aux;
        while(end>begin)
      aux=*end, *end--=*begin, *begin++=aux;
```

```
}

void itoa(int value, char* str, int base) {  //Function to convert the signed int to an ASCII char array

    static char num[] = "0123456789abcdefghijklmnopqrstuvwxyz";
    char* wstr=str;
    int sign;

    // Validate that base is between 2 and 35 (inlcusive)
    if (base<2 || base>35){
*wstr='\0';
return;
    }

    // Get magnitude and th value
    sign=value;
    if (sign < 0)
value = -value;

    do // Perform interger-to-string conversion.
*wstr++ = num[value%base]; //create the next number in converse by taking the modolus
    while(value/=base);  // stop when you get  a 0 for the quotient

    if(sign<0) //attch sign character, if needed
*wstr++='-';
```

```
        *wstr='\0'; //Attach a null character at end of char array. The string is in revers order at this point
   strreverse(str,wstr-1); // Reverse string

}

void port_init(){
        P6DIR |= BIT0|BIT1|BIT2|BIT3; // P1.6 outputs
        P6OUT |= BIT0;
        P1SEL0 |= BIT6 | BIT7;                // set 2-UART pin as second function
        P4SEL0 |= BIT2 | BIT3;                // set 2-UART pin as second function
        P4SEL1 &= ~BIT2;                // set 2-UART pin as second function
        P4SEL1 &= ~ BIT3;                // set 2-UART pin as second function
        // Configure GPIO
        P6DIR |= BIT0;                        // Set P6.0/LED to output direction
   P6OUT &= ~BIT0;                        // P6.0 LED off

        // Configure P1.1 as input (for motion sensor)
        P1DIR &= ~BIT1;
        P1REN |= BIT1; // Enable pull-up/pull-down resistor
        P1OUT |= BIT1; // Select pull-up resistor
        P1DIR &= ~BIT6;                // P1.6 and P1.7 output
        P1SEL1 &= ~BIT6;                // P1.6 and P1.7 options select
        P1DIR &= ~BIT7;                // P1.6 and P1.7 output
```

```
        P1SEL1 &= ~BIT7;                  // P1.6 and P1.7 options select

}

void Software_Trim()
{
        unsigned int oldDcoTap = 0xffff;
        unsigned int newDcoTap = 0xffff;
        unsigned int newDcoDelta = 0xffff;
        unsigned int bestDcoDelta = 0xffff;
        unsigned int csCtl0Copy = 0;
        unsigned int csCtl1Copy = 0;
        unsigned int csCtl0Read = 0;
        unsigned int csCtl1Read = 0;
        unsigned int dcoFreqTrim = 3;
        unsigned char endLoop = 0;

        do
        {
    CSCTL0 = 0x100;                   // DCO Tap = 256
    do
    {
       CSCTL7 &= ~DCOFFG;             // Clear DCO fault flag
    }while (CSCTL7 & DCOFFG);         // Test DCO fault flag
```

```
    //__delay_cycles((unsigned int)3000 * MCLK_FREQ_MHZ);// Wait FLL lock status
(FLLUNLOCK) to be stable
                                        // Suggest to wait 24 cycles of divided FLL reference clock
    while((CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)) && ((CSCTL7 & DCOFFG) == 0));

    csCtl0Read = CSCTL0;                // Read CSCTL0
    csCtl1Read = CSCTL1;                // Read CSCTL1

    oldDcoTap = newDcoTap;              // Record DCOTAP value of last time
    newDcoTap = csCtl0Read & 0x01ff;    // Get DCOTAP value of this time
    dcoFreqTrim = (csCtl1Read & 0x0070)>>4;// Get DCOFTRIM value

    if(newDcoTap < 256)                 // DCOTAP < 256
    {
      newDcoDelta = 256 - newDcoTap;    // Delta value between DCPTAP and 256
      if((oldDcoTap != 0xffff) && (oldDcoTap >= 256)) // DCOTAP cross 256
        endLoop = 1;                    // Stop while loop
      else
      {
        dcoFreqTrim--;
        CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
      }
    }
    else                                // DCOTAP >= 256
    {
      newDcoDelta = newDcoTap - 256;    // Delta value between DCPTAP and 256
```

```
            if(oldDcoTap < 256)             // DCOTAP cross 256
               endLoop = 1;                 // Stop while loop
            else
            {
               dcoFreqTrim++;
               CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
            }
         }

         if(newDcoDelta < bestDcoDelta)          // Record DCOTAP closest to 256
         {
            csCtl0Copy = csCtl0Read;
            csCtl1Copy = csCtl1Read;
            bestDcoDelta = newDcoDelta;
         }

          }while(endLoop == 0);                  // Poll until endLoop == 1

          CSCTL0 = csCtl0Copy;                   // Reload locked DCOTAP
          CSCTL1 = csCtl1Copy;                   // Reload locked DCOFTRIM
          while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)); // Poll until FLL is locked
}



// Configure ADC Temperature
```

```
void ConfigureAdc_temp(){

        ADCCTL0 |= ADCSHT_8 | ADCON;                    // ADC ON,temperature sample
period>30us

        ADCCTL1 |= ADCSHP;                             // s/w trig, single ch/conv, MODOSC

        ADCCTL2 &= ~ADCRES;                            // clear ADCRES in ADCCTL

        ADCCTL2 |= ADCRES_2;                           // 12-bit conversion results

        ADCMCTL0 |= ADCSREF_1 | ADCINCH_12;                 // ADC input ch A12 => temp
sense

        ADCMCTL0 |= ADCSREF_1 | ADCINCH_2;

        ADCIE |=ADCIE0;

}



// Configure Motion

  void ConfigureAdc_motion(){

      ADCCTL0 &= ~ADCENC;

      ADCCTL0 |= ADCSHT_8 | ADCON;                    // ADC ON,temperature sample
period>30us

      ADCCTL1 |= ADCSHP|ADCCONSEQ_1;                          // s/w trig, single ch/conv,
MODOSC

      ADCCTL2 &= ~ADCRES;                           // clear ADCRES in ADCCTL

      ADCCTL2 |= ADCRES_2;                          // 12-bit conversion results

      ADCMCTL0 |= ADCSREF_1 | ADCINCH_1;                 // ADC input ch A12 => temp sense

      ADCIE |=ADCIE0;

}



// Configure Light
```

```
void ConfigureAdc_light(){

    ADCCTL0 &= ~ADCENC;

    ADCCTL0 |= ADCSHT_8 | ADCON;                    // ADC ON,temperature sample
period>30us

    ADCCTL1 |= ADCSHP|ADCCONSEQ_1;                          // s/w trig, single ch/conv,
MODOSC

    ADCCTL2 &= ~ADCRES;                             // clear ADCRES in ADCCTL

    ADCCTL2 |= ADCRES_2;                            // 12-bit conversion results

    ADCMCTL0 |= ADCSREF_1 | ADCINCH_4;                  // ADC input ch A12 => temp sense

    ADCIE |=ADCIE0;

}



// Configure Gas Sensor

void ConfigureAdc_Gas_sensor(){

    ADCCTL0 &= ~ADCENC;

    ADCCTL0 |= ADCSHT_8 | ADCON;                    // ADC ON,temperature sample
period>30us

    ADCCTL1 |= ADCSHP|ADCCONSEQ_1;                          // s/w trig, single ch/conv,
MODOSC

    ADCCTL2 &= ~ADCRES;                             // clear ADCRES in ADCCTL

    ADCCTL2 |= ADCRES_2;                            // 12-bit conversion results

    ADCMCTL0 |= ADCSREF_1 | ADCINCH_3;                  // ADC input ch A12 => temp sense

    ADCIE |=ADCIE0;

}



void initialize_Adc(){
```

```
        ADCCTL0 &= ~ADCIFG;//CLEAR FLAG
    ADCMEM0=0x00000000;
        //ADCAE0=0x00;
    ADCCTL0=0x0000;
    ADCCTL1=0x0000;
}
```