HW #10

ECEN 621

Mrinmoy Sarkar

Date: 11/25/2019

LaunchPad Implementation (Modified C code):

```
#include <ti/devices/msp432p4xx/inc/msp.h>
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <ti/grlib/grlib.h>
#include "LcdDriver/Crystalfontz128x128_ST7735.h"
#include "LcdDriver/HAL_MSP_EXP432P401R_Crystalfontz128x128_ST7735.h"
#include <stdio.h>

/* Graphic library context */
Graphics_Context g_sContext;

/* Variable for storing lux value returned from OPT3001 */
float lux;

/* Timer_A Up Configuration Parameter */
const Timer_A_UpModeConfig upConfig =
{
        TIMER_A_CLOCKSOURCE_ACLK,               // ACLK Clock SOurce
        TIMER_A_CLOCKSOURCE_DIVIDER_1,          // ACLK/1 = 3MHz
        200,                                    // 200 tick period
        TIMER_A_TAIE_INTERRUPT_DISABLE,         // Disable Timer interrupt
        TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE,    // Disable CCR0 interrupt
        TIMER_A_DO_CLEAR                        // Clear value
};

/* Timer_A Compare Configuration Parameter  (PWM) */
Timer_A_CompareModeConfig compareConfig_PWM =
{
        TIMER_A_CAPTURECOMPARE_REGISTER_3,          // Use CCR3
        TIMER_A_CAPTURECOMPARE_INTERRUPT_DISABLE,   // Disable CCR interrupt
        TIMER_A_OUTPUTMODE_TOGGLE_SET,              // Toggle output but
        100                                         // 50% Duty Cycle
};


void init_I2C(void)
{
    P6->SEL0 |= BIT4 | BIT5; // set sda and scl for i2c communication
    // Configure USCI_B1 for I2C mode
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_SWRST; // Software reset enabled
    EUSCI_B1->CTLW0 = EUSCI_B_CTLW0_SWRST | // Remain eUSCI in reset mode
           EUSCI_B_CTLW0_MODE_3 |          // I2C mode
           EUSCI_B_CTLW0_MST |             // Master mode
           EUSCI_B_CTLW0_SYNC |            // Sync mode
           EUSCI_B_CTLW0_SSEL__SMCLK;      // SMCLK
    EUSCI_B1->CTLW1 |= EUSCI_B_CTLW1_ASTP_0;// No Automatic stop generated
    EUSCI_B1->BRW = 480;                     // baudrate = SMCLK / 480 = 100kHz
    EUSCI_B1->CTLW0 &= ~EUSCI_B_CTLW0_SWRST;// Release eUSCI from reset
}
```

```c
void init_lightSensor(void)
{
    EUSCI_B1->I2CSA = 0x0044;               // Slave address
    EUSCI_B1->IFG &= ~(EUSCI_B_IFG_TXIFG0 | EUSCI_B_IFG_RXIFG0); // clear interrupt
flag
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_TR; //set transmit mode
    /* Wait until ready to write PL */
    while (EUSCI_B1->STATW & EUSCI_B_STATW_BBUSY);
    //Store current transmit interrupt enable
    uint16_t txieStatus = EUSCI_B1->IE & EUSCI_B_IE_TXIE0;
    //Disable transmit interrupt enable
    EUSCI_B1->IE &= ~EUSCI_B_IE_TXIE0;
    //Send start condition.
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_TR + EUSCI_B_CTLW0_TXSTT;
    //Poll for transmit interrupt flag and start condition flag.
    while ((EUSCI_B1->CTLW0 & EUSCI_B_CTLW0_TXSTT) || !(EUSCI_B1->IFG &
EUSCI_B_IFG_TXIFG0));
    //Send single byte data.
    EUSCI_B1->TXBUF = 0x01;
    //Reinstate transmit interrupt enable
    EUSCI_B1->IE |= txieStatus;
    //If interrupts are not used, poll for flags
    if (!(EUSCI_B1->IE & EUSCI_B_IE_TXIE0))
    {
        //Poll for transmit interrupt flag.
        while (!(EUSCI_B1->IFG & EUSCI_B_IFG_TXIFG0));

    }
    //Send single byte data.
    EUSCI_B1->TXBUF = 0xC4;
    //If interrupts are not used, poll for flags
    if (!(EUSCI_B1->IE & EUSCI_B_IE_TXIE0))
    {
        //Poll for transmit interrupt flag.
        while (!(EUSCI_B1->IFG & EUSCI_B_IFG_TXIFG0));
    }
    //Send single byte data.
    EUSCI_B1->TXBUF = 0x10;
    //Poll for transmit interrupt flag.
    while (!(EUSCI_B1->IFG & EUSCI_B_IFG_TXIFG0) && !(EUSCI_B1->IFG &
EUSCI_B_IFG_NACKIFG));
    //Send stop condition.
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_TXSTP;
}

unsigned long int get_sensordata(void)
{
    uint16_t exponent = 0;
    uint32_t result = 0;
    int16_t raw;
    /* Specify slave address for OPT3001 */
    EUSCI_B1->I2CSA = 0x0044;               // Slave address
    EUSCI_B1->IFG &= ~(EUSCI_B_IFG_TXIFG0 | EUSCI_B_IFG_RXIFG0); // clear interrupt
flags
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_TR; //set transmit mode
```

```
    EUSCI_B1->IFG &= ~(EUSCI_B_IFG_TXIFG0 | EUSCI_B_IFG_RXIFG0); // clear interrupt
flag
    while (EUSCI_B1->STATW & EUSCI_B_STATW_BBUSY); // check if i2c bus is busy
    //Store current transmit interrupt enable
    uint16_t txieStatus = EUSCI_B1->IE & EUSCI_B_IE_TXIE0;
    //Disable transmit interrupt enable
    EUSCI_B1->IE &= ~EUSCI_B_IE_TXIE0;
    //Send start condition.
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_TR + EUSCI_B_CTLW0_TXSTT;
    //Poll for transmit interrupt flag and start condition flag.
    while ((EUSCI_B1->CTLW0 & EUSCI_B_CTLW0_TXSTT) || !(EUSCI_B1->IFG &
EUSCI_B_IFG_TXIFG0));
    //Send single byte data.
    EUSCI_B1->TXBUF = 0x00; // result register address
    //Reinstate transmit interrupt enable
    EUSCI_B1->IE |= txieStatus;
    while (!(EUSCI_B1->IFG & EUSCI_B_IFG_TXIFG0)); // check if transmit complete or
not
    if (!(EUSCI_B1->IE & EUSCI_B_IE_TXIE0))
    {
        //Poll for transmit interrupt flag.
        while (!(EUSCI_B1->IFG & EUSCI_B_IFG_TXIFG0));
    }
    //Send stop condition.
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_TXSTP;
    while (!(EUSCI_B1->IFG & EUSCI_B_IFG_STPIFG)); //check stop flag
    EUSCI_B1->CTLW0 = (EUSCI_B1->CTLW0 & (~EUSCI_B_CTLW0_TR)) |
EUSCI_B_CTLW0_TXSTT;
    while (!(EUSCI_B1->IFG & EUSCI_B_IFG_RXIFG0));
    int val = 0;
    int valScratch = 0;
    val = EUSCI_B1->RXBUF; // read sensor data MSB
    //Send stop condition.
    EUSCI_B1->CTLW0 |= EUSCI_B_CTLW0_TXSTP;
    //Wait for Stop to finish
    while (EUSCI_B1->CTLW0 & EUSCI_B_CTLW0_TXSTP)
    {
        // Wait for RX buffer
        while (!(EUSCI_B1->IFG & EUSCI_B_IFG_RXIFG));
    }
    valScratch = EUSCI_B1->RXBUF; // read sensor data LSB
    /* Shift val to top MSB */
    val = (val << 8);
    /* Read from I2C RX Register and write to LSB of val */
    val |= valScratch;
    raw = (int16_t)val;
    /*Convert to LUX*/
    //extract result & exponent data from raw readings
    result = raw&0x0FFF;
    exponent = (raw>>12)&0x000F;
    //convert raw readings to LUX
    switch(exponent){
    case 0: //*0.015625
       result = result>>6;
       break;
```

```
    case 1: //*0.03125
       result = result>>5;
       break;
    case 2: //*0.0625
       result = result>>4;
       break;
    case 3: //*0.125
       result = result>>3;
       break;
    case 4: //*0.25
       result = result>>2;
       break;
    case 5: //*0.5
       result = result>>1;
       break;
    case 6:
       result = result;
       break;
    case 7: //*2
       result = result<<1;
       break;
    case 8: //*4
       result = result<<2;
       break;
    case 9: //*8
       result = result<<3;
       break;
    case 10: //*16
       result = result<<4;
       break;
    case 11: //*32
       result = result<<5;
       break;
    }
    return result;
}

/*
 * Main function
 */
int main(void)
{
    /* Halting WDT and disabling master interrupts */
    MAP_WDT_A_holdTimer();
    MAP_Interrupt_disableMaster();

    /* Set the core voltage level to VCORE1 */
    MAP_PCM_setCoreVoltageLevel(PCM_VCORE1);

    /* Set 2 flash wait states for Flash bank 0 and 1*/
    MAP_FlashCtl_setWaitState(FLASH_BANK0, 2);
    MAP_FlashCtl_setWaitState(FLASH_BANK1, 2);

    /* Initializes Clock System */
    MAP_CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_48);
```

```
    MAP_CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
    MAP_CS_initClockSignal(CS_HSMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
    MAP_CS_initClockSignal(CS_SMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
    MAP_CS_initClockSignal(CS_ACLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_1);

    /* Initializes display */
    Crystalfontz128x128_Init();

    /* Set default screen orientation */
    Crystalfontz128x128_SetOrientation(0);

    /* Initializes graphics context */
    Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128,
&g_sCrystalfontz128x128_funcs);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
    GrContextFontSet(&g_sContext, &g_sFontFixed6x8);
    Graphics_clearDisplay(&g_sContext);
    Graphics_drawStringCentered(&g_sContext,
                                    (int8_t *)"Light Sensor:",
                                    AUTO_STRING_LENGTH,
                                    64,
                                    30,
                                    OPAQUE_TEXT);

    /* Configures P2.6 to PM_TA0.3 for using Timer PWM to control LCD backlight */
    MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2, GPIO_PIN6,
            GPIO_PRIMARY_MODULE_FUNCTION);

    /* Configuring Timer_A0 for Up Mode and starting */
    MAP_Timer_A_configureUpMode(TIMER_A0_BASE, &upConfig);
    MAP_Timer_A_startCounter(TIMER_A0_BASE, TIMER_A_UP_MODE);

    /* Initialize compare registers to generate PWM */
    MAP_Timer_A_initCompare(TIMER_A0_BASE, &compareConfig_PWM);

    /* Initialize I2C communication */
    init_I2C();

    /* Initialize OPT3001 digital ambient light sensor */
    init_lightSensor();

    __delay_cycles(100000);

    while(1)
    {
        /* Obtain lux value from OPT3001 */
        lux = get_sensordata();

        char string[20];
        sprintf(string, "%f", lux);
        Graphics_drawStringCentered(&g_sContext,
                                        (int8_t *)string,
                                        6,
                                        48,
```

```
                                        70,
                                        OPAQUE_TEXT);

        sprintf(string, "lux");
        Graphics_drawStringCentered(&g_sContext,
                                        (int8_t *)string,
                                        3,
                                        86,
                                        70,
                                        OPAQUE_TEXT);

        /* Adjust LCD Backlight */
        if (lux < 2000) {
            compareConfig_PWM.compareValue = (int)(((2000*0.1f) + (lux*0.9f))/2000
* 200);
        } else {
            compareConfig_PWM.compareValue = 200;
        }
        Timer_A_initCompare(TIMER_A0_BASE, &compareConfig_PWM);
    }
}
```