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  }  **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);  }  } |