**SSI LAB**

**Videos with the following names are found in Dropbox:**

**Task 1 -** Lab11-T01.mov

**Task 2 -** Lab11-T02.mov

**Task 3 -** Lab11-T03.mov

**Task 1: Adding comments to original code.**

//main code for this task is found in OrbitDemo2

/\*

\* main.c

\*/

/\* ------------------------------------------------------------ \*/

/\* Include File Definitions \*/

/\* ------------------------------------------------------------ \*/

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <time.h>

**#include** <stdio.h>

**#include** <math.h>

**#include** "LaunchPad.h"

**#include** "OrbitBoosterPackDefs.h"

**#include** "OrbitOled.h"

**#include** "OrbitOledGrph.h"

**#include** "OrbitOledChar.h"

**#include** "FillPat.h"

**#include** "I2CEEPROM.h"

**#include** "delay.h"

/\* ------------------------------------------------------------ \*/

/\* General Type Definitions \*/

/\* ------------------------------------------------------------ \*/

/\* ------------------------------------------------------------ \*/

/\* Local Type Definitions \*/

/\* ------------------------------------------------------------ \*/

**#define** DEMO\_0 0

**#define** DEMO\_1 2

**#define** DEMO\_2 1

**#define** DEMO\_3 3

/\* ------------------------------------------------------------ \*/

/\* Global Variables \*/

/\* ------------------------------------------------------------ \*/

**extern** **int** xchOledMax; // defined in OrbitOled.c

**extern** **int** ychOledMax; // defined in OrbitOled.c

/\* ------------------------------------------------------------ \*/

/\* Local Variables \*/

/\* ------------------------------------------------------------ \*/

**char** chSwtCur;

**char** chSwtPrev;

bool fClearOled;

/\*

\* Rocket Definitions

\*/

// Define the top left corner of rocket

**int** xcoRocketStart = 48; //8\*6

**int** ycoRocketStart = 11;

**int** xcoExhstStart = 39;

**int** ycoExhstStart = 11;

**int** cRocketWidth = 24;

**int** cRocketHeight = 16;

**int** cExhstWidth = 9;

**int** cExhstHeight = 16;

**int** fExhstSwt = 0;

**char** rgBMPRocket[] = {0xFF, 0x11, 0xF1, 0x11, 0xF1, 0x12, 0x14, 0x18,

0x90, 0x10, 0x10, 0x10, 0x10, 0x10, 0x90, 0x10,

0x10, 0xE0, 0xC0, 0x80, 0x80, 0x80, 0x80, 0x80,

0xFF, 0x88, 0x8F, 0x88, 0x8F, 0x48, 0x28, 0x19,

0x0A, 0x09, 0x08, 0x08, 0x08, 0x09, 0x0A, 0x09,

0x08, 0x07, 0x03, 0x01, 0x01, 0x01, 0x01, 0x01};

**char** rgBMPExhst1[] = {0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF0,

0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x07, 0x0F, 0x0F};

**char** rgBMPExhst2[] = {0x00, 0x80, 0x80, 0xC0, 0xE0, 0xE0, 0xF0, 0xF0, 0xF0,

0x00, 0x01, 0x01, 0x03, 0x07, 0x07, 0x0F, 0x0F, 0x0F};

/\* ------------------------------------------------------------ \*/

/\* Forward Declarations \*/

/\* ------------------------------------------------------------ \*/

**void** **DeviceInit**();

**char** **CheckSwitches**();

**void** **OrbitSetOled**();

**void** **OrbitDemo0**();

**void** **OrbitDemo1**();

**void** **OrbitDemo2**();

**void** **OrbitDemo3**();

**void** **RocketRight**(**int** xcoUpdate, **int** ycoUpdate);

**void** **RocketLeft**(**int** xcoUpdate, **int** ycoUpdate);

**void** **RocketStop**(**int** xcoUpdate, **int** ycoUpdate, bool fDir);

**char** **I2CGenTransmit**(**char** \* pbData, **int** cSize, bool fRW, **char** bAddr);

bool **I2CGenIsNotIdle**();

/\* ------------------------------------------------------------ \*/

/\* Procedure Definitions \*/

/\* ------------------------------------------------------------ \*/

/\* ------------------------------------------------------------ \*/

/\*\*\* main()

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Main program loop

\*/

**#define** RED\_LED GPIO\_PIN\_1

**#define** BLUE\_LED GPIO\_PIN\_2

**#define** GREEN\_LED GPIO\_PIN\_3

**int** **main**(**void**) {

**char** bDemoState = 0;

**volatile** uint32\_t ui32Loop;

**DeviceInit**();

**while**(1) {

bDemoState = **CheckSwitches**(); //check state of switches

**for**(ui32Loop = 0; ui32Loop < 200000; ui32Loop++)

{

}

**switch**(bDemoState) { //run demo according to switches

**case** DEMO\_0:

**OrbitDemo0**();

**break**;

**case** DEMO\_1:

**OrbitDemo1**();

**break**;

**case** DEMO\_2:

**OrbitDemo2**();

**break**;

**case** DEMO\_3:

**OrbitDemo3**();

**break**;

**default**:

**OrbitDemo0**();

**break**;

}

}

**return** 0;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* DeviceInit

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Initialize I2C Communication, and GPIO

\*/

**void** **DeviceInit**()

{

/\*

\* First, Set Up the Clock.

\* Main OSC -> SYSCTL\_OSC\_MAIN

\* Runs off 16MHz clock -> SYSCTL\_XTAL\_16MHZ

\* Use PLL -> SYSCTL\_USE\_PLL

\* Divide by 4 -> SYSCTL\_SYSDIV\_4

\*/

SysCtlClockSet(SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ | SYSCTL\_USE\_PLL | SYSCTL\_SYSDIV\_4);

/\*

\* Enable and Power On All GPIO Ports

\*/

//SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOA | SYSCTL\_PERIPH\_GPIOB | SYSCTL\_PERIPH\_GPIOC |

// SYSCTL\_PERIPH\_GPIOD | SYSCTL\_PERIPH\_GPIOE | SYSCTL\_PERIPH\_GPIOF);

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOA );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOB );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOC );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOD );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOE );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOF );

/\*

\* Pad Configure.. Setting as per the Button Pullups on

\* the Launch pad (active low).. changing to pulldowns for Orbit

\*/

GPIOPadConfigSet(SWTPort, SWT1 | SWT2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(BTN1Port, BTN1, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(BTN2Port, BTN2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(LED1Port, LED1, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED2Port, LED2, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED3Port, LED3, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED4Port, LED4, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

/\*

\* Initialize Switches as Input

\*/

GPIOPinTypeGPIOInput(SWTPort, SWT1 | SWT2);

/\*

\* Initialize Buttons as Input

\*/

GPIOPinTypeGPIOInput(BTN1Port, BTN1);

GPIOPinTypeGPIOInput(BTN2Port, BTN2);

/\*

\* Initialize LEDs as Output

\*/

GPIOPinTypeGPIOOutput(LED1Port, LED1);

GPIOPinTypeGPIOOutput(LED2Port, LED2);

GPIOPinTypeGPIOOutput(LED3Port, LED3);

GPIOPinTypeGPIOOutput(LED4Port, LED4);

/\*

\* Enable ADC Periph

\*/

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

GPIOPinTypeADC(AINPort, AIN);

/\*

\* Enable ADC with this Sequence

\* 1. ADCSequenceConfigure()

\* 2. ADCSequenceStepConfigure()

\* 3. ADCSequenceEnable()

\* 4. ADCProcessorTrigger();

\* 5. Wait for sample sequence ADCIntStatus();

\* 6. Read From ADC

\*/

ADCSequenceConfigure(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

ADCSequenceStepConfigure(ADC0\_BASE, 0, 0, ADC\_CTL\_IE | ADC\_CTL\_END | ADC\_CTL\_CH0);

ADCSequenceEnable(ADC0\_BASE, 0);

/\*

\* Initialize the OLED

\*/

**OrbitOledInit**();

/\*

\* Reset flags

\*/

chSwtCur = 0;

chSwtPrev = 0;

fClearOled = true;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* CheckSwitches()

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Return the state of the Switches

\*/

**char** **CheckSwitches**() {

**long** lSwt1;

**long** lSwt2;

chSwtPrev = chSwtCur; //current switch state becomes previous state

lSwt1 = GPIOPinRead(SWT1Port, SWT1); //read state of switch 1

lSwt2 = GPIOPinRead(SWT2Port, SWT2); //read state of switch 2

chSwtCur = (lSwt1 | lSwt2) >> 6; //shift switches' bits

**if**(chSwtCur != chSwtPrev) { //clear LED when switches change

fClearOled = true;

}

**return** chSwtCur; //return current state of switches

}

/\* ------------------------------------------------------------ \*/

/\*\*\* OrbitDemo2

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Reads the temperature and then updates the OLED display

\*\* with the temperature and alerts! if necessary

\*/

**void** **OrbitDemo2**() {

//define variables

**char** szTempLabel[] = {'T', 'e', 'm', 'p', ':', ' ', '\0'};

**char** szC[] = {' ', 'C', '\0'};

**char** rgchReadTemp[] = {0, 0, 0};

**char** rgchWriteTemp[] = {1, 0x20};

**short** tempReg;

**short** tempWhole;

**short** tempDec;

**int** i;

**char** szTemp[6];

/\*

\* If applicable, reset OLED

\*/

**if**(fClearOled == true) {

**OrbitOledClear**();

**OrbitOledMoveTo**(0,0);

**OrbitOledSetCursor**(0,0);

fClearOled = false;

/\*

\* Setup Oled for Temperature

\*/

**OrbitOledSetCursor**(0, 0);

**OrbitOledPutString**(szTempLabel);

/\*

\* Enable I2C Peripheral

\*/

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0);

SysCtlPeripheralReset(SYSCTL\_PERIPH\_I2C0);

/\*

\* Set I2C GPIO pins

\*/

GPIOPinTypeI2C(I2CSDAPort, I2CSDA\_PIN);

GPIOPinTypeI2CSCL(I2CSCLPort, I2CSCL\_PIN);

GPIOPinConfigure(I2CSCL);

GPIOPinConfigure(I2CSDA);

/\*

\* Setup I2C

\*/

I2CMasterInitExpClk(I2C0\_BASE, SysCtlClockGet(), false);

/\*

\* Setup Temperature Sensor

\*/

**I2CGenTransmit**(rgchWriteTemp, 1, WRITE, TEMPADDR);

}

rgchReadTemp[0] = 0; //read temperature

**I2CGenTransmit**(rgchReadTemp, 2, READ, TEMPADDR);

tempReg = (rgchReadTemp[1] << 8) | rgchReadTemp[2];

tempWhole = 0; //keeps integer part of temp

tempDec = 0; //keeps decimal part of temp

**for**(i = 0; i < 7; i++) { //convert integer from binary value to decimal

**if**(tempReg & (1 << (8 + i))) {

tempWhole += pow(2,i);

}

}

**if**(tempReg & (1 << 7) ) { //convert decimal part and store it

tempDec += 50;

}

**if**(tempReg & (1 << 6) ) {

tempDec += 25;

}

sprintf(szTemp, "%d.%d", tempWhole, tempDec); //convert whole part and decimal part to string

**if**(tempDec == 0) { //if decimal part is zero, insert blank for alligning

szTemp[4] = ' ';

}

szTemp[5] = '\0'; //null terminate the string

**OrbitOledSetCursor**(6,0); //set cursor

**OrbitOledPutString**(szTemp); //display "temp:" banner

**OrbitOledSetCursor**(11, 0); //set cursor

**OrbitOledPutString**(szC); //display temperature value

}

/\* ------------------------------------------------------------ \*/

/\*\*\* I2CGenTransmit

\*\*

\*\* Parameters:

\*\* pbData - Pointer to transmit buffer (read or write)

\*\* cSize - Number of byte transactions to take place

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Transmits data to a device via the I2C bus. Differs from

\*\* I2C EEPROM Transmit in that the registers in the device it

\*\* is addressing are addressed with a single byte. Lame, but..

\*\* it works.

\*\*

\*/

**char** **I2CGenTransmit**(**char** \* pbData, **int** cSize, bool fRW, **char** bAddr) {

**int** i;

**char** \* pbTemp;

pbTemp = pbData;

/\*Start\*/

/\*Send Address High Byte\*/

/\* Send Write Block Cmd

\*/

I2CMasterSlaveAddrSet(I2C0\_BASE, bAddr, WRITE);

I2CMasterDataPut(I2C0\_BASE, \*pbTemp);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

**DelayMs**(1);

/\* Idle wait

\*/

**while**(**I2CGenIsNotIdle**());

/\* Increment data pointer

\*/

pbTemp++;

/\*Execute Read or Write\*/

**if**(fRW == READ) {

/\* Resend Start condition

\*\* Then send new control byte

\*\* then begin reading

\*/

I2CMasterSlaveAddrSet(I2C0\_BASE, bAddr, READ);

**while**(I2CMasterBusy(I2C0\_BASE));

/\* Begin Reading

\*/

**for**(i = 0; i < cSize; i++) {

**if**(cSize == i + 1 && cSize == 1) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE);

**DelayMs**(1);

**while**(I2CMasterBusy(I2C0\_BASE));

}

**else** **if**(cSize == i + 1 && cSize > 1) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_FINISH);

**DelayMs**(1);

**while**(I2CMasterBusy(I2C0\_BASE));

}

**else** **if**(i == 0) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_START);

**DelayMs**(1);

**while**(I2CMasterBusy(I2C0\_BASE));

/\* Idle wait

\*/

**while**(**I2CGenIsNotIdle**());

}

**else** {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**DelayMs**(1);

**while**(I2CMasterBusy(I2C0\_BASE));

/\* Idle wait

\*/

**while**(**I2CGenIsNotIdle**());

}

**while**(I2CMasterBusy(I2C0\_BASE));

/\* Read Data

\*/

\*pbTemp = (**char**)I2CMasterDataGet(I2C0\_BASE);

pbTemp++;

}

}

**else** **if**(fRW == WRITE) {

/\*Loop data bytes

\*/

**for**(i = 0; i < cSize; i++) {

/\* Send Data

\*/

I2CMasterDataPut(I2C0\_BASE, \*pbTemp);

**while**(I2CMasterBusy(I2C0\_BASE));

**if**(i == cSize - 1) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH);

**DelayMs**(1);

**while**(I2CMasterBusy(I2C0\_BASE));

}

**else** {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_CONT);

**DelayMs**(1);

**while**(I2CMasterBusy(I2C0\_BASE));

/\* Idle wait

\*/

**while**(**I2CGenIsNotIdle**());

}

pbTemp++;

}

}

/\*Stop\*/

**return** 0x00;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* I2CGenIsNotIdle()

\*\*

\*\* Parameters:

\*\* pbData - Pointer to transmit buffer (read or write)

\*\* cSize - Number of byte transactions to take place

\*\*

\*\* Return Value:

\*\* TRUE is bus is not idle, FALSE if bus is idle

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Returns TRUE if the bus is not idle

\*\*

\*/

bool **I2CGenIsNotIdle**() {

**return** !I2CMasterBusBusy(I2C0\_BASE); //return i2c bus status

}

**Task 2: Change the code to display the temperature in Centigrade and Fahrenheit alternately on the OLED.**

//main code for this task is found in OrbitDemo2

/\*

\* main.c

\*/

/\* ------------------------------------------------------------ \*/

/\* Include File Definitions \*/

/\* ------------------------------------------------------------ \*/

#include <stdint.h>

#include <stdbool.h>

#include <time.h>

#include <stdio.h>

#include <math.h>

#include "LaunchPad.h"

#include "OrbitBoosterPackDefs.h"

#include "OrbitOled.h"

#include "OrbitOledGrph.h"

#include "OrbitOledChar.h"

#include "FillPat.h"

#include "I2CEEPROM.h"

#include "delay.h"

/\* ------------------------------------------------------------ \*/

/\* General Type Definitions \*/

/\* ------------------------------------------------------------ \*/

/\* ------------------------------------------------------------ \*/

/\* Local Type Definitions \*/

/\* ------------------------------------------------------------ \*/

#define DEMO\_0 0

#define DEMO\_1 2

#define DEMO\_2 1

#define DEMO\_3 3

/\* ------------------------------------------------------------ \*/

/\* Global Variables \*/

/\* ------------------------------------------------------------ \*/

extern int xchOledMax; // defined in OrbitOled.c

extern int ychOledMax; // defined in OrbitOled.c

bool toggle; //used to toggle between celsius and fahrenheit

/\* ------------------------------------------------------------ \*/

/\* Local Variables \*/

/\* ------------------------------------------------------------ \*/

char chSwtCur;

char chSwtPrev;

bool fClearOled;

/\*

\* Rocket Definitions

\*/

// Define the top left corner of rocket

int xcoRocketStart = 48; //8\*6

int ycoRocketStart = 11;

int xcoExhstStart = 39;

int ycoExhstStart = 11;

int cRocketWidth = 24;

int cRocketHeight = 16;

int cExhstWidth = 9;

int cExhstHeight = 16;

int fExhstSwt = 0;

char rgBMPRocket[] = {0xFF, 0x11, 0xF1, 0x11, 0xF1, 0x12, 0x14, 0x18,

0x90, 0x10, 0x10, 0x10, 0x10, 0x10, 0x90, 0x10,

0x10, 0xE0, 0xC0, 0x80, 0x80, 0x80, 0x80, 0x80,

0xFF, 0x88, 0x8F, 0x88, 0x8F, 0x48, 0x28, 0x19,

0x0A, 0x09, 0x08, 0x08, 0x08, 0x09, 0x0A, 0x09,

0x08, 0x07, 0x03, 0x01, 0x01, 0x01, 0x01, 0x01};

char rgBMPExhst1[] = {0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF0,

0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x07, 0x0F, 0x0F};

char rgBMPExhst2[] = {0x00, 0x80, 0x80, 0xC0, 0xE0, 0xE0, 0xF0, 0xF0, 0xF0,

0x00, 0x01, 0x01, 0x03, 0x07, 0x07, 0x0F, 0x0F, 0x0F};

/\* ------------------------------------------------------------ \*/

/\* Forward Declarations \*/

/\* ------------------------------------------------------------ \*/

void DeviceInit();

char CheckSwitches();

void OrbitSetOled();

void OrbitDemo0();

void OrbitDemo1();

void OrbitDemo2();

void OrbitDemo3();

void RocketRight(int xcoUpdate, int ycoUpdate);

void RocketLeft(int xcoUpdate, int ycoUpdate);

void RocketStop(int xcoUpdate, int ycoUpdate, bool fDir);

char I2CGenTransmit(char \* pbData, int cSize, bool fRW, char bAddr);

bool I2CGenIsNotIdle();

/\* ------------------------------------------------------------ \*/

/\* Procedure Definitions \*/

/\* ------------------------------------------------------------ \*/

/\* ------------------------------------------------------------ \*/

/\*\*\* main()

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Main program loop

\*/

#define RED\_LED GPIO\_PIN\_1

#define BLUE\_LED GPIO\_PIN\_2

#define GREEN\_LED GPIO\_PIN\_3

int main(void) {

char bDemoState = 0;

volatile uint32\_t ui32Loop;

toggle = false;

DeviceInit();

while(1) {

bDemoState = CheckSwitches();

for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++)

{

}

switch(bDemoState) {

case DEMO\_0:

OrbitDemo0();

break;

case DEMO\_1:

OrbitDemo1();

break;

case DEMO\_2:

OrbitDemo2();

break;

case DEMO\_3:

OrbitDemo3();

break;

default:

OrbitDemo0();

break;

}

}

return 0;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* DeviceInit

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Initialize I2C Communication, and GPIO

\*/

void DeviceInit()

{

/\*

\* First, Set Up the Clock.

\* Main OSC -> SYSCTL\_OSC\_MAIN

\* Runs off 16MHz clock -> SYSCTL\_XTAL\_16MHZ

\* Use PLL -> SYSCTL\_USE\_PLL

\* Divide by 4 -> SYSCTL\_SYSDIV\_4

\*/

SysCtlClockSet(SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ | SYSCTL\_USE\_PLL | SYSCTL\_SYSDIV\_4);

/\*

\* Enable and Power On All GPIO Ports

\*/

//SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOA | SYSCTL\_PERIPH\_GPIOB | SYSCTL\_PERIPH\_GPIOC |

// SYSCTL\_PERIPH\_GPIOD | SYSCTL\_PERIPH\_GPIOE | SYSCTL\_PERIPH\_GPIOF);

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOA );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOB );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOC );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOD );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOE );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOF );

/\*

\* Pad Configure.. Setting as per the Button Pullups on

\* the Launch pad (active low).. changing to pulldowns for Orbit

\*/

GPIOPadConfigSet(SWTPort, SWT1 | SWT2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(BTN1Port, BTN1, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(BTN2Port, BTN2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(LED1Port, LED1, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED2Port, LED2, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED3Port, LED3, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED4Port, LED4, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

/\*

\* Initialize Switches as Input

\*/

GPIOPinTypeGPIOInput(SWTPort, SWT1 | SWT2);

/\*

\* Initialize Buttons as Input

\*/

GPIOPinTypeGPIOInput(BTN1Port, BTN1);

GPIOPinTypeGPIOInput(BTN2Port, BTN2);

/\*

\* Initialize LEDs as Output

\*/

GPIOPinTypeGPIOOutput(LED1Port, LED1);

GPIOPinTypeGPIOOutput(LED2Port, LED2);

GPIOPinTypeGPIOOutput(LED3Port, LED3);

GPIOPinTypeGPIOOutput(LED4Port, LED4);

/\*

\* Enable ADC Periph

\*/

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

GPIOPinTypeADC(AINPort, AIN);

/\*

\* Enable ADC with this Sequence

\* 1. ADCSequenceConfigure()

\* 2. ADCSequenceStepConfigure()

\* 3. ADCSequenceEnable()

\* 4. ADCProcessorTrigger();

\* 5. Wait for sample sequence ADCIntStatus();

\* 6. Read From ADC

\*/

ADCSequenceConfigure(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

ADCSequenceStepConfigure(ADC0\_BASE, 0, 0, ADC\_CTL\_IE | ADC\_CTL\_END | ADC\_CTL\_CH0);

ADCSequenceEnable(ADC0\_BASE, 0);

/\*

\* Initialize the OLED

\*/

OrbitOledInit();

/\*

\* Reset flags

\*/

chSwtCur = 0;

chSwtPrev = 0;

fClearOled = true;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* CheckSwitches()

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Return the state of the Switches

\*/

char CheckSwitches() {

long lSwt1;

long lSwt2;

chSwtPrev = chSwtCur;

lSwt1 = GPIOPinRead(SWT1Port, SWT1);

lSwt2 = GPIOPinRead(SWT2Port, SWT2);

chSwtCur = (lSwt1 | lSwt2) >> 6;

if(chSwtCur != chSwtPrev) {

fClearOled = true;

}

return chSwtCur;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* OrbitDemo2

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Reads the temperature and then updates the OLED display

\*\* with the temperature and alerts! if necessary

\*/

void OrbitDemo2() {

char szTempLabel[] = {'T', 'e', 'm', 'p', ':', ' ', '\0'}; //temp banner

char szC[] = {' ', 'C', '\0'};

char szF[] = {' ', 'F', '\0'} //more temp banner

char rgchReadTemp[] = {0, 0, 0};

char rgchWriteTemp[] = {1, 0x20};

short tempReg;

short tempWhole;

short tempDec;

int i;

char szTemp[6];

char szTempF[7]; //array to store temp

short tempWholeF; //array to store integer part of temp

/\*

\* If applicable, reset OLED

\*/

if(fClearOled == true) {

OrbitOledClear();

OrbitOledMoveTo(0,0);

OrbitOledSetCursor(0,0);

fClearOled = false;

/\*

\* Setup Oled for Temperature

\*/

OrbitOledSetCursor(0, 0);

OrbitOledPutString(szTempLabel);

/\*

\* Enable I2C Peripheral

\*/

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0);

SysCtlPeripheralReset(SYSCTL\_PERIPH\_I2C0);

/\*

\* Set I2C GPIO pins

\*/

GPIOPinTypeI2C(I2CSDAPort, I2CSDA\_PIN);

GPIOPinTypeI2CSCL(I2CSCLPort, I2CSCL\_PIN);

GPIOPinConfigure(I2CSCL);

GPIOPinConfigure(I2CSDA);

/\*

\* Setup I2C

\*/

I2CMasterInitExpClk(I2C0\_BASE, SysCtlClockGet(), false);

/\*

\* Setup Temperature Sensor

\*/

I2CGenTransmit(rgchWriteTemp, 1, WRITE, TEMPADDR);

}

rgchReadTemp[0] = 0;

I2CGenTransmit(rgchReadTemp, 2, READ, TEMPADDR);

tempReg = (rgchReadTemp[1] << 8) | rgchReadTemp[2];

tempWhole = 0;

tempDec = 0;

for(i = 0; i < 7; i++) {

if(tempReg & (1 << (8 + i))) {

tempWhole += pow(2,i);

}

}

if(tempReg & (1 << 7) ) {

tempDec += 50;

}

if(tempReg & (1 << 6) ) {

tempDec += 25;

}

tempWholeF = tempWhole; //truncate

tempWholeF = ((tempWholeF \* 9) + 160) / 5; //convert to fahrenheit

sprintf(szTemp, "%d.%d", tempWhole, tempDec);

sprintf(szTempF, "%d ", tempWholeF);

if(tempDec == 0) {

szTemp[4] = ' ';

}

szTemp[5] = '\0';

szTempF[6] = '\0'; //null terminated

if(toggle == false){

toggle = true;

OrbitOledSetCursor(6,0);

OrbitOledPutString(szTemp);

OrbitOledSetCursor(11, 0);

OrbitOledPutString(szC);

}

else{

toggle = false;

OrbitOledSetCursor(6,0);

OrbitOledPutString(szTempF);

OrbitOledSetCursor(11, 0);

OrbitOledPutString(szF);

}

DelayMs(2000);

}

/\* ------------------------------------------------------------ \*/

/\*\*\* I2CGenIsNotIdle()

\*\*

\*\* Parameters:

\*\* pbData - Pointer to transmit buffer (read or write)

\*\* cSize - Number of byte transactions to take place

\*\*

\*\* Return Value:

\*\* TRUE is bus is not idle, FALSE if bus is idle

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Returns TRUE if the bus is not idle

\*\*

\*/

bool I2CGenIsNotIdle() {

return !I2CMasterBusBusy(I2C0\_BASE);

}

**Task 3: Change the code to display the values from 0 to 100 on the OLED.**

//main code for this task is found in OrbitDemo0

/\*

\* main.c

\*/

/\* ------------------------------------------------------------ \*/

/\* Include File Definitions \*/

/\* ------------------------------------------------------------ \*/

#include <stdint.h>

#include <stdbool.h>

#include <time.h>

#include <stdio.h>

#include <math.h>

#include "LaunchPad.h"

#include "OrbitBoosterPackDefs.h"

#include "OrbitOled.h"

#include "OrbitOledGrph.h"

#include "OrbitOledChar.h"

#include "FillPat.h"

#include "I2CEEPROM.h"

#include "delay.h"

/\* ------------------------------------------------------------ \*/

/\* General Type Definitions \*/

/\* ------------------------------------------------------------ \*/

/\* ------------------------------------------------------------ \*/

/\* Local Type Definitions \*/

/\* ------------------------------------------------------------ \*/

#define DEMO\_0 0

#define DEMO\_1 2

#define DEMO\_2 1

#define DEMO\_3 3

/\* ------------------------------------------------------------ \*/

/\* Global Variables \*/

/\* ------------------------------------------------------------ \*/

extern int xchOledMax; // defined in OrbitOled.c

extern int ychOledMax; // defined in OrbitOled.c

/\* ------------------------------------------------------------ \*/

/\* Local Variables \*/

/\* ------------------------------------------------------------ \*/

char chSwtCur;

char chSwtPrev;

bool fClearOled;

/\* ------------------------------------------------------------ \*/

/\* Forward Declarations \*/

/\* ------------------------------------------------------------ \*/

void DeviceInit();

char CheckSwitches();

void OrbitSetOled();

void OrbitDemo0();

void OrbitDemo1();

void OrbitDemo2();

void OrbitDemo3();

void RocketRight(int xcoUpdate, int ycoUpdate);

void RocketLeft(int xcoUpdate, int ycoUpdate);

void RocketStop(int xcoUpdate, int ycoUpdate, bool fDir);

char I2CGenTransmit(char \* pbData, int cSize, bool fRW, char bAddr);

bool I2CGenIsNotIdle();

/\* ------------------------------------------------------------ \*/

/\* Procedure Definitions \*/

/\* ------------------------------------------------------------ \*/

/\* ------------------------------------------------------------ \*/

/\*\*\* main()

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Main program loop

\*/

#define RED\_LED GPIO\_PIN\_1

#define BLUE\_LED GPIO\_PIN\_2

#define GREEN\_LED GPIO\_PIN\_3

int main(void) {

char bDemoState = 0;

volatile uint32\_t ui32Loop;

DeviceInit();

while(1) {

bDemoState = CheckSwitches();

for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++)

{

}

switch(bDemoState) {

case DEMO\_0:

OrbitDemo0();

break;

case DEMO\_1:

OrbitDemo1();

break;

case DEMO\_2:

OrbitDemo2();

break;

case DEMO\_3:

OrbitDemo3();

break;

default:

OrbitDemo0();

break;

}

}

return 0;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* DeviceInit

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Initialize I2C Communication, and GPIO

\*/

void DeviceInit()

{

/\*

\* First, Set Up the Clock.

\* Main OSC -> SYSCTL\_OSC\_MAIN

\* Runs off 16MHz clock -> SYSCTL\_XTAL\_16MHZ

\* Use PLL -> SYSCTL\_USE\_PLL

\* Divide by 4 -> SYSCTL\_SYSDIV\_4

\*/

SysCtlClockSet(SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ | SYSCTL\_USE\_PLL | SYSCTL\_SYSDIV\_4);

/\*

\* Enable and Power On All GPIO Ports

\*/

//SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOA | SYSCTL\_PERIPH\_GPIOB | SYSCTL\_PERIPH\_GPIOC |

// SYSCTL\_PERIPH\_GPIOD | SYSCTL\_PERIPH\_GPIOE | SYSCTL\_PERIPH\_GPIOF);

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOA );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOB );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOC );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOD );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOE );

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOF );

/\*

\* Pad Configure.. Setting as per the Button Pullups on

\* the Launch pad (active low).. changing to pulldowns for Orbit

\*/

GPIOPadConfigSet(SWTPort, SWT1 | SWT2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(BTN1Port, BTN1, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(BTN2Port, BTN2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(LED1Port, LED1, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED2Port, LED2, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED3Port, LED3, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

GPIOPadConfigSet(LED4Port, LED4, GPIO\_STRENGTH\_8MA\_SC, GPIO\_PIN\_TYPE\_STD);

/\*

\* Initialize Switches as Input

\*/

GPIOPinTypeGPIOInput(SWTPort, SWT1 | SWT2);

/\*

\* Initialize Buttons as Input

\*/

GPIOPinTypeGPIOInput(BTN1Port, BTN1);

GPIOPinTypeGPIOInput(BTN2Port, BTN2);

/\*

\* Initialize LEDs as Output

\*/

GPIOPinTypeGPIOOutput(LED1Port, LED1);

GPIOPinTypeGPIOOutput(LED2Port, LED2);

GPIOPinTypeGPIOOutput(LED3Port, LED3);

GPIOPinTypeGPIOOutput(LED4Port, LED4);

/\*

\* Enable ADC Periph

\*/

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

GPIOPinTypeADC(AINPort, AIN);

/\*

\* Enable ADC with this Sequence

\* 1. ADCSequenceConfigure()

\* 2. ADCSequenceStepConfigure()

\* 3. ADCSequenceEnable()

\* 4. ADCProcessorTrigger();

\* 5. Wait for sample sequence ADCIntStatus();

\* 6. Read From ADC

\*/

ADCSequenceConfigure(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

ADCSequenceStepConfigure(ADC0\_BASE, 0, 0, ADC\_CTL\_IE | ADC\_CTL\_END | ADC\_CTL\_CH0);

ADCSequenceEnable(ADC0\_BASE, 0);

/\*

\* Initialize the OLED

\*/

OrbitOledInit();

/\*

\* Reset flags

\*/

chSwtCur = 0;

chSwtPrev = 0;

fClearOled = true;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* CheckSwitches()

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Return the state of the Switches

\*/

char CheckSwitches() {

long lSwt1;

long lSwt2;

chSwtPrev = chSwtCur;

lSwt1 = GPIOPinRead(SWT1Port, SWT1);

lSwt2 = GPIOPinRead(SWT2Port, SWT2);

chSwtCur = (lSwt1 | lSwt2) >> 6;

if(chSwtCur != chSwtPrev) {

fClearOled = true;

}

return chSwtCur;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* OrbitDemo0

\*\*

\*\* Parameters:

\*\* none

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Buttons turn on LEDs, and the ADC reading

\*\* (altered with the potentiometer, VR1) is continuously

\*\* output to the OLED.

\*/

void OrbitDemo0() {

long lBtn1;

long lBtn2;

int num;

char numString[4];

char szDemo1[] = {'N', 'u', 'm', 'b', 'e', 'r', 's', ' ', '1', '-', '1', '0', '0', ':', '\0'};

/\*

\* If applicable, reset OLED

\*/

if(fClearOled == true) {

OrbitOledClear();

OrbitOledMoveTo(0,0);

OrbitOledSetCursor(0,0);

fClearOled = false;

}

/\* Display Demo Banner

\*

\*/

OrbitOledSetCursor(0, 0);

OrbitOledPutString(szDemo1);

//clear numbers from previous iterations

OrbitOledSetCursor(0, 1);

OrbitOledPutString(" ");

//display numbers

for(num = 0; num <= 100; ){

sprintf(numString, "%d", num);

OrbitOledSetCursor(0, 1);

OrbitOledPutString(numString);

DelayMs(100);

num++;

}

/\* Check SWT and BTN states and update LEDs

\*

\*/

lBtn1 = GPIOPinRead(BTN1Port, BTN1);

lBtn2 = GPIOPinRead(BTN2Port, BTN2);

if(lBtn1 == BTN1) {

GPIOPinWrite(LED1Port, LED1, LED1);

GPIOPinWrite(LED2Port, LED2, LED2);

}

else {

GPIOPinWrite(LED1Port, LED1, LOW);

GPIOPinWrite(LED2Port, LED2, LOW);

}

if(lBtn2 == BTN2) {

GPIOPinWrite(LED3Port, LED3, LED3);

GPIOPinWrite(LED4Port, LED4, LED4);

}

else {

GPIOPinWrite(LED3Port, LED3, LOW);

GPIOPinWrite(LED4Port, LED4, LOW);

}

}

/\* ------------------------------------------------------------ \*/

/\*\*\* I2CGenTransmit

\*\*

\*\* Parameters:

\*\* pbData - Pointer to transmit buffer (read or write)

\*\* cSize - Number of byte transactions to take place

\*\*

\*\* Return Value:

\*\* none

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Transmits data to a device via the I2C bus. Differs from

\*\* I2C EEPROM Transmit in that the registers in the device it

\*\* is addressing are addressed with a single byte. Lame, but..

\*\* it works.

\*\*

\*/

char I2CGenTransmit(char \* pbData, int cSize, bool fRW, char bAddr) {

int i;

char \* pbTemp;

pbTemp = pbData;

/\*Start\*/

/\*Send Address High Byte\*/

/\* Send Write Block Cmd

\*/

I2CMasterSlaveAddrSet(I2C0\_BASE, bAddr, WRITE);

I2CMasterDataPut(I2C0\_BASE, \*pbTemp);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

DelayMs(1);

/\* Idle wait

\*/

while(I2CGenIsNotIdle());

/\* Increment data pointer

\*/

pbTemp++;

/\*Execute Read or Write\*/

if(fRW == READ) {

/\* Resend Start condition

\*\* Then send new control byte

\*\* then begin reading

\*/

I2CMasterSlaveAddrSet(I2C0\_BASE, bAddr, READ);

while(I2CMasterBusy(I2C0\_BASE));

/\* Begin Reading

\*/

for(i = 0; i < cSize; i++) {

if(cSize == i + 1 && cSize == 1) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE);

DelayMs(1);

while(I2CMasterBusy(I2C0\_BASE));

}

else if(cSize == i + 1 && cSize > 1) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_FINISH);

DelayMs(1);

while(I2CMasterBusy(I2C0\_BASE));

}

else if(i == 0) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_START);

DelayMs(1);

while(I2CMasterBusy(I2C0\_BASE));

/\* Idle wait

\*/

while(I2CGenIsNotIdle());

}

else {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

DelayMs(1);

while(I2CMasterBusy(I2C0\_BASE));

/\* Idle wait

\*/

while(I2CGenIsNotIdle());

}

while(I2CMasterBusy(I2C0\_BASE));

/\* Read Data

\*/

\*pbTemp = (char)I2CMasterDataGet(I2C0\_BASE);

pbTemp++;

}

}

else if(fRW == WRITE) {

/\*Loop data bytes

\*/

for(i = 0; i < cSize; i++) {

/\* Send Data

\*/

I2CMasterDataPut(I2C0\_BASE, \*pbTemp);

while(I2CMasterBusy(I2C0\_BASE));

if(i == cSize - 1) {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH);

DelayMs(1);

while(I2CMasterBusy(I2C0\_BASE));

}

else {

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_CONT);

DelayMs(1);

while(I2CMasterBusy(I2C0\_BASE));

/\* Idle wait

\*/

while(I2CGenIsNotIdle());

}

pbTemp++;

}

}

/\*Stop\*/

return 0x00;

}

/\* ------------------------------------------------------------ \*/

/\*\*\* I2CGenIsNotIdle()

\*\*

\*\* Parameters:

\*\* pbData - Pointer to transmit buffer (read or write)

\*\* cSize - Number of byte transactions to take place

\*\*

\*\* Return Value:

\*\* TRUE is bus is not idle, FALSE if bus is idle

\*\*

\*\* Errors:

\*\* none

\*\*

\*\* Description:

\*\* Returns TRUE if the bus is not idle

\*\*

\*/

bool I2CGenIsNotIdle() {

return !I2CMasterBusBusy(I2C0\_BASE);

}