TI ARM Lab 8
Accelerometers

National Science Foundation
Funded in part, by a grant from the National Science Foundation
DUE 1068182

Acknowledgements
Developed by Craig Kief and Brian Zufelt from the COSMIAC Research Center at the University of New Mexico. Co-Developers are Bassam Matar from Chandler-Gilbert and Karl Henry from Drake State. Originally Funded by the National Science Foundation.

Lab Summary
This lab continues with the introduction of the concepts of the I^2C and how it is implemented on the ARM processor.

Lab Goal
The goal of this lab is to continue to build upon the skills learned from previous labs. Previous work with the temperature sensor has prepared the initial understanding of the two wire bus architecture.

Each of these labs add upon the previous labs and it is the intention of the authors that the students will build (with each lab) a better understanding of the ARM processor and basic C code. Even though these tutorials assume the student has not entered with a knowledge of C code, it is the desire that by the time the student completes the entire series of tutorials that they will have a sufficient knowledge of C code so as to be able to accomplish useful projects on the ARM processor.

It is assumed that the "student" will be installing the packages in accordance with the instructions in Lab1. That process will provide the framework and load all the support files/projects for the associated workshop. Each Lab will have two files. For example, there is a generally a Lab x project and a Lab x Solution project. The only difference between these two projects is the code in the main.c file. The exception is in the interrupt project (Lab 5) where the …startup_ccs.c file is also edited in the solution. The reason this was done this way was to provide the student with two choices. They can follow the tutorial in the Lab and type in the associated code identified in the tutorial. This will allow for errors to be made and learning to be achieved through the debugging process. The alternative is that the student has a fall back option if everything goes bad and they can just look at/copy the source file contents from the "Solution" projects.

Learning Objectives
The student should begin to become familiar with the concept of the accelerometer and ways to accomplish simple projects. An accelerometer is a device that will measure acceleration forces. These forces may be static, like the constant force of gravity pulling at your feet, or they could be dynamic - caused by moving or vibrating the accelerometer. By measuring the amount of static acceleration due to gravity, you can find out the angle the device is tilted at with respect to the earth. By sensing the amount of dynamic acceleration, you can analyze the way the device is moving. An accelerometer can help the designer understand the system better. Is it driving uphill? Is it flying horizontally or is it dive bombing? A designer can write code to answer all of these questions using the data provided by an accelerometer. In the computing world, IBM and Apple have recently started using accelerometers in their laptops to protect hard drives from damage. If you accidentally drop the laptop, the accelerometer detects the sudden free-fall, and switches the hard drive off so the heads don't crash on the platters. In a similar fashion, high G accelerometers are the industry standard way of detecting car crashes and deploying airbags at just the right time.

The ADXL345 (used on the ORBIT board) is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 *g*. Its functional block diagram is shown in Figure 1. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I^2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 m*g*/LSB) enables measurement of inclination changes less than 1.0°.

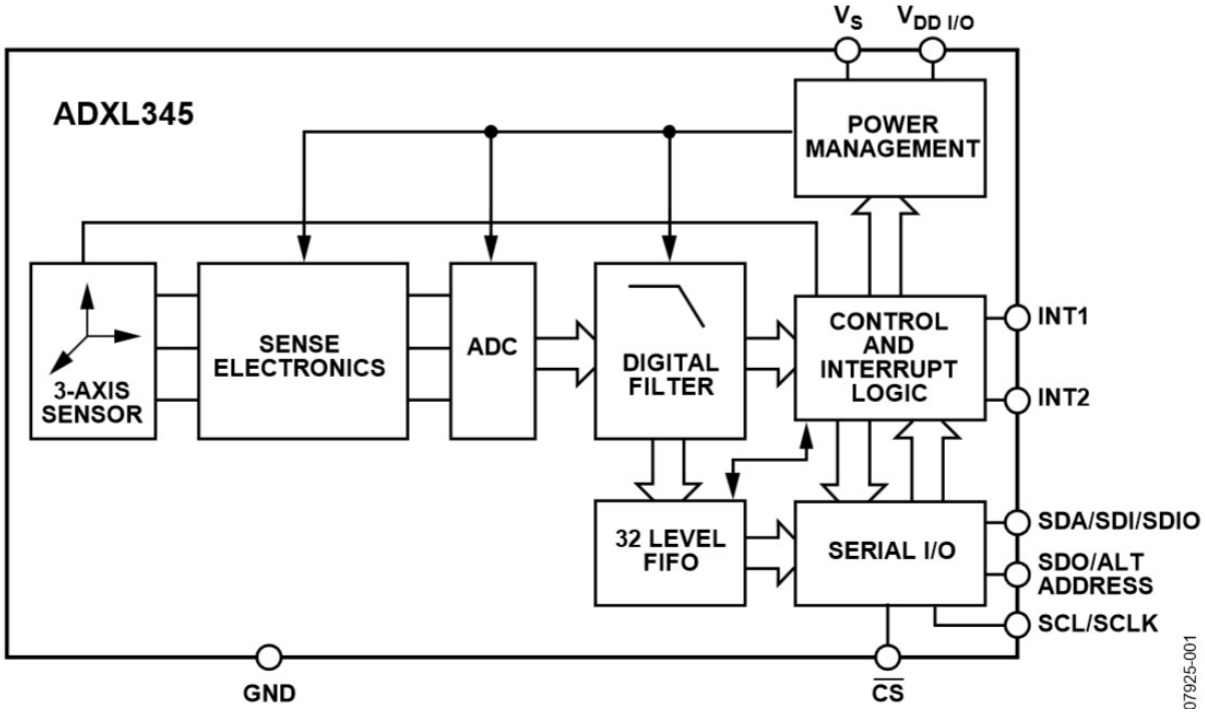


Figure 1. Functional Block Diagram for the Accelorometer

Time Required
Approximately three hours

Lab Preparation

It is highly recommended that the student read through this procedure once before actually using it was a tutorial. By understanding the final goal it will be easier to use this as a tutorial. This Lab will introduce the Tiva C Series TM4C123G LaunchPad Evaluation Kit which is a low-cost evaluation platform for ARM® Cortex™-M4F-based microcontrollers from Texas Instruments. The design of the TM4C123G LaunchPad highlights the TM4C123GH6PM microcontroller with a USB 2.0 device interface and hibernation module.

The EK-TM4C123GXL evaluation kit also features programmable user buttons and a Red, Green, RGB LED for custom applications. The stackable headers of the Tiva C Series TM4C123G LaunchPad BoosterPack XL Interface make it easy and simple to expand the functionality of the TM4C123G LaunchPad when interfacing to other peripherals with Texas Instruments' MCU BoosterPacks.

Equipment and Materials

Access to the four software packages: Code Composer, Tivaware, Driver file and ATE Workshop installer are required and should have been installed in accordance with Lab 1. In addition, the user should have access to the Tiva evaluation kit (EK-TM4C123GXL) and (for Labs 3-9) the Digilent Orbit board. It is assumed that the student has already completed Lab 1 and the software is installed.

Software needed	Quantity
The installation files are covered in Lab 1. The software package can be downloaded as one large zip file from this URL: http://cosmiac.org/thrust-areas/education-and-workforce-development/community-portal/	1
Hardware needed	Quantity
The hardware required is the Tiva LaunchPad Kit. The kit and the ORBIT daughter card can be purchased from the Digilent Corporation www.digilentinc.com	1

Additional References

This page is for general information: http://www.ti.com/tool/ek-tm4c123gxl on the Tiva LauchPad Kit. If the link above is no longer valid, just google the Tiva LaunchPad and it will pop up. The full set of tutorials is available on this website: http://cosmiac.org/thrust-areas/education-and-workforce-development/microcontrollers/ate-developed-material/

Lab Procedure: Install/Connect board to computer

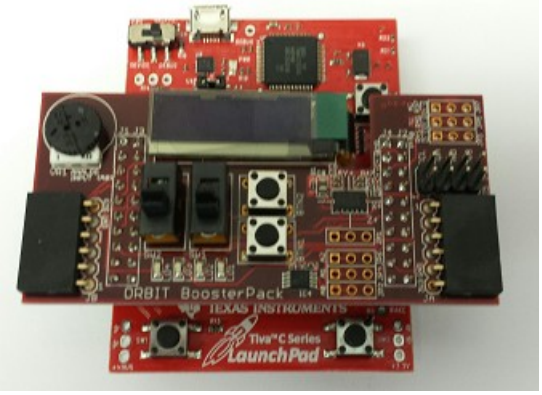


Figure 2. ARM and ORBIT Combination

This picture of the correct way to mate the Tiva LaunchPad and the Digilent Orbit boards together. Please note that the accelerometer chip is located to the right of the two push buttons on the BoosterPack board.

Figure 3. Code Composer Icon

Launch Code Composer and where prompted, chose the workspaces location to store your project (as shown in Figure 3).

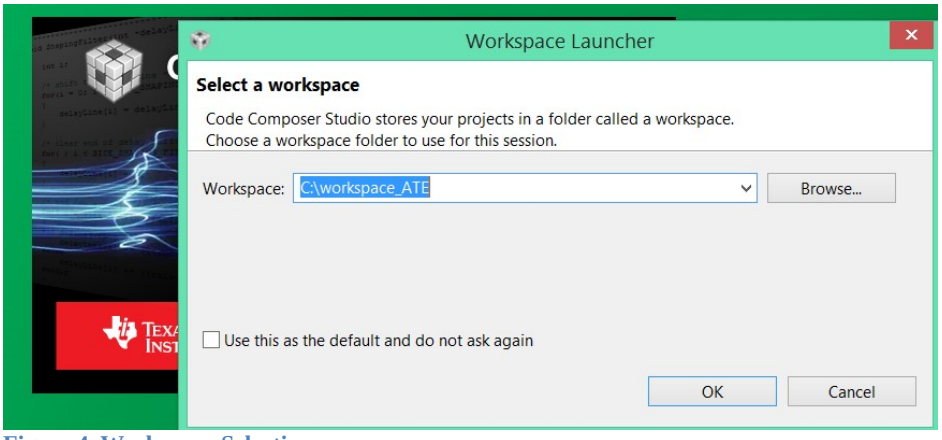


Figure 4. Workspace Selection

Since the installer for the workshop has been run prior to this, the user will be presented with the following view (Figure 4) where all lab projects exist.

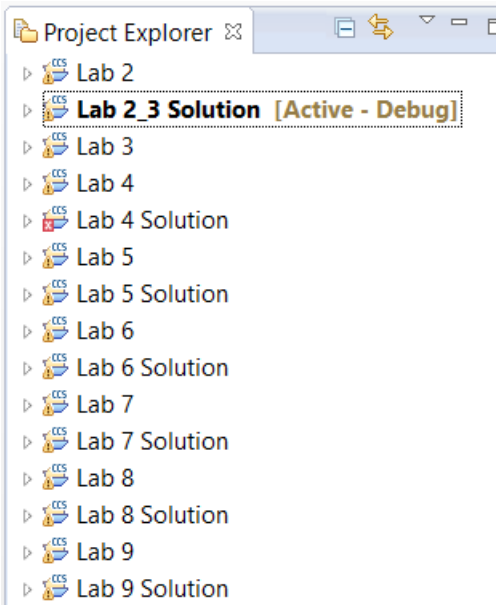


Figure 5. CCS Starting Point

The laboratory material is created to have the students type in a lot of the code. Only by typing the code and then debugging the errors will a user ever really understand how to do projects. For the sake of this activity, the source code is provided at the end of the tutorial. In Lab 8, open main.c. Then either type in all the code from attachment 2 or copy and paste in the code from Attachement 2 into main.c.

It is now important to spend a couple of minutes to explain a little more about the accelerometer used in this project. If you are looking on the Orbit board, there is a small chip in the upper right quadrant. It has a silkscreened cross on the board around the chip with a "X" and "Z" lettering. This is the ADXL345 chip.

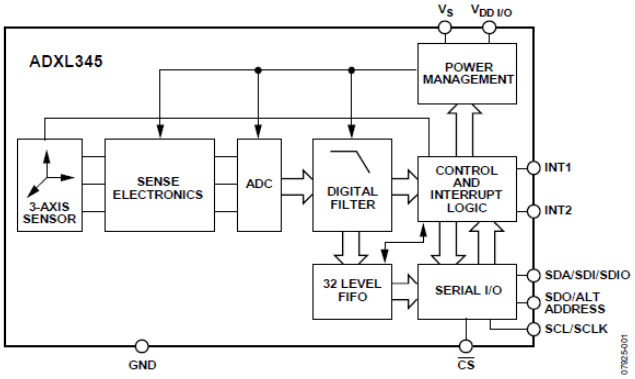


Figure 6. ADXL345

As can be seen in Figure 6, it has a 3-axis sensor that can be used to detect the orientation of the board. In this tutorial, only the X axis will be measured. This is to keep the tutorial as simple as possible. As a result,

the function will have three returned values: -1 (going left), 0 (standing still), or 1 (going right). The chip is very sensitive. The system transfers data on an I²C line. The ADXL345 has an I²C output and the ARM processor has I²C inputs as shown in Figure 7 from the datasheet. The Rp are what are called pull up resistors and they are critical in board designs as they help to ensure the data lines operate with clean transition pulses.

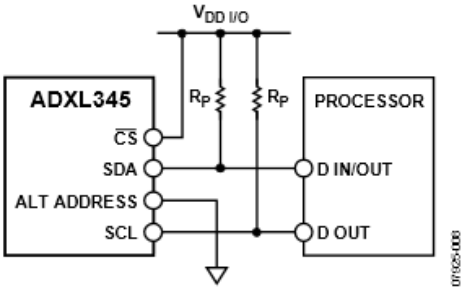


Figure 7. Connection Diagram

The I²C protocol interface has a unique set of waveforms to allow the system to know when to start and stop the transfer of data. This waveform timing diagram is shown in Figure 8 and shows how the processor knows when to receive data.

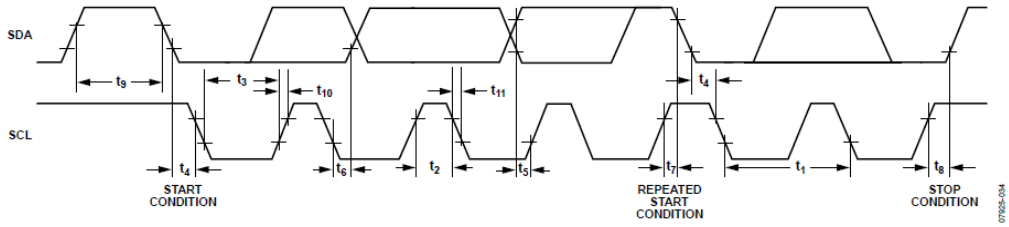
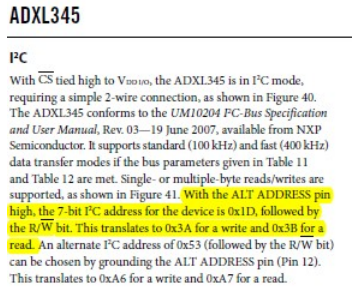


Figure 8. I2C Waveform Timing Diagram

The entire solution code for this project is given at the end of this tutorial in attachment 2. This initially looks like a large amount of very complex code however, when you remember that much of the LED portion has been presented in previous labs it is less scary. Add to that the fact that another large portion of the code is nothing more than delays to slow the system down for human consumption and it is less scary. Some of the code that is presented in this lab for the first time is shown below:

```
#define ACCEL_W 0x3A
#define ACCEL_R 0x3B
#define ACCEL_ADDR 0x1D
```

These three lines above are the definition parameters for the accelorometer. They are taken from the ADXL345 datasheet as shown below. It clearly defines the address as x1D, the read as x3B and the write as x3A. The key to I²C is to remember that it is a very powerful way to attach a wide variety of different sensors onto a single bus. This bus is also only two wires wide. So, to do this, each device on the bus must have a unique address as well as unique address for all functions associated with the specific sensor. With I²C devices, if you go into their datasheets, it is possible to find their unique addresses (although not always quickly). Some devices often have up to four addresses possible through the use of onboard jumpers.



ADXL345

I²C

With $\overline{CS}$ tied high to $V_{DD\ I/O}$, the ADXL345 is in I²C mode, requiring a simple 2-wire connection, as shown in Figure 40. The ADXL345 conforms to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, available from NXP Semiconductor. It supports standard (100 kHz) and fast (400 kHz) data transfer modes if the bus parameters given in Table 11 and Table 12 are met. Single- or multiple-byte reads/writes are supported, as shown in Figure 41. With the ALT ADDRESS pin high, the 7-bit I²C address for the device is 0x1D, followed by the R/$\overline{W}$ bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I²C address of 0x53 (followed by the R/$\overline{W}$ bit) can be chosen by grounding the ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

Figure 9. Datasheet Excerpt

The next set of code shown below is a new topic. It is called a "function prototype." Most experienced designers like to have the main starting portion of their code appear at the top of the main.c file. The beginning of the program is normally identified with "**void main**(**void**) {". To make things cleaner, most designers will also use the function prototypes to declare the functions at the top of the file and then further down in the program after the main function, will do the full explanation and declaration of what the functions actually do. In this case, there are two function prototypes. The first is to initialize the accelerometer and the second is to read the chip.

```
void Accel_int();       // Function to initialize the Accelerometer

signed int Accel_read();// Function to read the Accelerometer
```

The next portion of the code initializes the LEDs. Refer to attachment 3 at the end of this document. Find the accelerometer and look at what pins are required. This is very similar to what has been done in previous labs. All ports that are required must be enabled and peripherals must be initialized before they can be used. The way this code is designed to operate is to have the LEDs fall or flow downhill based on the way the board is tilted. If the board is tilted 90 degrees to the left, the LEDs will fall down to the bottom. If the board is tilted the other direction, the LEDs will still attempt to fall to the bottom. After this section of the code, the two functions that were prototyped earlier are expanded upon.

Open the main.c file that is in the lab installer for Lab 8. When this is compared to the code in attachment 2, it is clear to see that the code for the LED waterfall is missing. The task of this tutorial is to type in this code. When done, debug/compile and download. Run the program and make sure the LEDs flow appropriately.

As a challenge exercise, reduce the time to flow the lights from one end to the other or light up the Tiva board with different color LEDs when extremes are hit. A better challenge is to add the other two axis.

Attachment 1: main.c file (starting)

```
/*****************************************************

Project : Orbit Lab 8 ATE (ACCEL)
Version : 2.0
Date    : 2/20/2015
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
This source provides an introduction to the sensing capabilities
for embedded systems. The student will read accelerometer data
and toggle the proper LED to provide tilt measurement


*****************************************************
Chip type               : ARM TM4C123GH6PM
Program type            : Firmware
Core Clock frequency    : 80.000000 MHz
*****************************************************/

#define ACCEL_W 0x3A                                    // Addresses for the accelerometer
#define ACCEL_R 0x3B
#define ACCEL_ADDR 0x1D

// Define needed for pin_map.h
#define PART_TM4C123GH6PM

#include <stdbool.h>
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "inc/hw_i2c.h"
#include "driverlib/i2c.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"


void Accel_int();                               // Function prototype to initialize the Accelerometer

signed int Accel_read(); // Function prototype to read the Accelerometer

void main(void) {

        signed short int LED_value = 1;

        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);  //setup clock

        SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);                     // Enable I2C hardware
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);            // Enable Pin hardware

        GPIOPinConfigure(GPIO_PB3_I2C0SDA);                                     // Configure GPIO pin for I2C Data line
        GPIOPinConfigure(GPIO_PB2_I2C0SCL);                                     // Configure GPIO Pin for I2C clock line

        GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3);  // Set Pin Type

        // Enable Peripheral ports for output
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);                                    //PORTC
        GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7);          // LED 1 LED 2

        GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);                             // LED 4

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);                                    //PORT D
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6);                             // LED 3

        //setup the I2C
        GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);            // SDA MUST BE STD
        GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_OD);             // SCL MUST BE OPEN DRAIN
        I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);                                        // The False sets the controller
to 100kHz communication
```

```
	Accel_int();		// Function to initialize the Accelerometer

	while(1){

		/* ********************************************************************************************************
		 * Fill in this section to read data from the Accelerometer and move the LEDs according to the X axis   *
		 * Task: call the Accel_read function and add it to LED_value. Using a number between 1-4 turn on the   *
		 * LEDs 1-4 to show tilt in the x axis.
		 *                                                                                                      *
		 ********************************************************************************************************/

	}
}

void Accel_int(){		// Function to initialize the Accelerometer

	I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, false);  // false means transmit

	I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);	// Send Start condition

	I2CMasterDataPut(I2C0_BASE, 0x2D);                                                  // Writing to the Accel control
reg
	SysCtlDelay(20000);
		// Delay for first transmission
	I2CMasterDataPut(I2C0_BASE, 0x08);                                                  // Send Value to control
Register

	I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);	// Send Stop condition

	while(I2CMasterBusBusy(I2C0_BASE)){};                                               // Wait for I2C controller to finish operations

}

signed int Accel_read() {	// Function to read the Accelerometer

	signed int data;
	signed short value = 0;			// value of x

	unsigned char MSB;
	unsigned char LSB;

	I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, false);                                // false means transmit

	I2CMasterDataPut(I2C0_BASE, 0x32);
	SysCtlDelay(20000);
	I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);                            //Request LSB of X Axis
	SysCtlDelay(2000000);
		// Delay for first transmission

	I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, true);                         // false means transmit

	I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);                         //Request LSB of X Axis
	SysCtlDelay(20000);

	LSB = I2CMasterDataGet(I2C0_BASE);
	SysCtlDelay(20000);

	I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, false);                                // false means transmit
	I2CMasterDataPut(I2C0_BASE, 0x33);

	I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);                            //Request LSB of X Axis
	SysCtlDelay(2000000);
		// Delay for first transmission

	I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, true);                         // false means transmit

	I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);                         //Request LSB of X Axis
	SysCtlDelay(20000);

	MSB = I2CMasterDataGet(I2C0_BASE);

	value = (MSB << 8 | LSB);                       // Bit shift MSB the OR it with LSB to get 16-bit value

	if(value < -250 ){                                      // testing axis for value
		data = -1;
	}
	else if (value > 250){
```

```
                data = 1;
        }

        else{
                data = 0;
        }

        SysCtlDelay(20000);

        return data;                                    // return value
}
```

Attachment 2: main.c file (solution)

```
/*****************************************************

Project : Orbit Lab 8 ATE (ACCEL)
Version : 2.0
Date    : 2/20/2015
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
This source provides an introduction to the sensing capabilities
for embedded systems. The student will read accelerometer data
and toggle the proper LED to provide tilt measurement


*****************************************************
Chip type               : ARM TM4C123GH6PM
Program type            : Firmware
Core Clock frequency            : 80.000000 MHz
*****************************************************/

#define ACCEL_W 0x3A                            // Addresses for the accelerometer
#define ACCEL_R 0x3B
#define ACCEL_ADDR 0x1D

// Define needed for pin_map.h
#define PART_TM4C123GH6PM

#include <stdbool.h>
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "inc/hw_i2c.h"
#include "driverlib/i2c.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"

void Accel_int();                       // Function prototype to initialize the Accelerometer

signed int Accel_read();        // Function prototype to read the Accelerometer

void main(void) {

        signed short int LED_value = 1;

        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);  //setup clock

        SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);             // Enable I2C hardware
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);            // Enable Pin hardware

        GPIOPinConfigure(GPIO_PB3_I2C0SDA);                                     // Configure GPIO pin for I2C Data line
        GPIOPinConfigure(GPIO_PB2_I2C0SCL);                                     // Configure GPIO Pin for I2C clock line

        GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3);  // Set Pin Type

        // Enable Peripheral ports for output
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);                                    //PORTC
        GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7);          // LED 1 LED 2

        GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);                             // LED 4

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);                                    //PORT D
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6);                             // LED 3

        //setup the I2C
        GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);            // SDA MUST BE STD
        GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_OD);             // SCL MUST BE OPEN
DRAIN
        I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);                                                        //
The False sets the controller to 100kHz communication
```

```
        Accel_int();                    // Function to initialize the Accelerometer

        while(1){

                // Fill in this section to read data from the Accelerometer and move the LEDs according to the X axis

                LED_value = LED_value + Accel_read();

                if(LED_value <= 1){
                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40); // LED 1 on LED 2 Off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);        // LED 3 off, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);                                        // LED 4 off
                        LED_value = 1;      // reset value to maintain range
                }

                else if(LED_value == 2){
                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80); // LED 1 off LED 2 on
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 on
                }
                else if(LED_value == 3){
                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); // LED 1 off LED 2 off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); // LED 3 on, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 0ff
                }
                else if(LED_value >= 4){
                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); // LED 1 off LED 2 Off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 4 on
                        LED_value = 4;      // reset value to maintain range
                }

        }
}

void Accel_int(){               // Function to initialize the Accelerometer

        I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, false);  // false means transmit

        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);           // Send Start condition

        I2CMasterDataPut(I2C0_BASE, 0x2D);                                      // Writing to the Accel control reg
        SysCtlDelay(20000);
                        // Delay for first transmission
        I2CMasterDataPut(I2C0_BASE, 0x08);                                      // Send Value to control Register

        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);          // Send Stop condition

        while(I2CMasterBusBusy(I2C0_BASE)){};                                           // Wait for I2C controller to finish operations

}

signed int Accel_read() {// Function to read the Accelerometer

        signed int data;
        signed short value = 0;                         // value of x

        unsigned char MSB;
        unsigned char LSB;

        I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, false);                                    // false means transmit

        I2CMasterDataPut(I2C0_BASE, 0x32);
        SysCtlDelay(20000);
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);                                        //Request LSB of X Axis
        SysCtlDelay(2000000);
                        // Delay for first transmission

        I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, true);                                     // false means transmit

        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);                             //Request LSB of X Axis
```

```
        SysCtlDelay(20000);

        LSB = I2CMasterDataGet(I2C0_BASE);
        SysCtlDelay(20000);

        I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, false);                    // false means transmit
        I2CMasterDataPut(I2C0_BASE, 0x33);

        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);                        //Request LSB of X Axis
        SysCtlDelay(2000000);
                        // Delay for first transmission

        I2CMasterSlaveAddrSet(I2C0_BASE, ACCEL_ADDR, true);                     // false means transmit

        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);             //Request LSB of X Axis
        SysCtlDelay(20000);

        MSB = I2CMasterDataGet(I2C0_BASE);

        value = (MSB << 8 | LSB);

        if(value < -250 ){                                      // testing axis for value
                data = -1;
        }
        else if (value > 250){
                data = 1;
        }

        else{
                data = 0;
        }

        SysCtlDelay(20000);

        return data;                                                    // return value
}
```

Attachment 3: Block Diagram of the Pins Used in Projects