	EMBRY-RIDDLE AERONAUTICAL UNIVERSITY **Department of Electrical, Computer, and Software Engineering** **CEC322**: Microprocessor Laboratory (Spring 2019), **SECTION** PC51 Due Date: 9-APR-2019 Turned in: 9-APR-2019		
	PREPARED BY:	**EQUIPMENT S/N:**	**PERFORMED:**
	Gray, Andrea	DK-TM4C123G MPU-9150	26-MAR-2019 & 02-APR-2019

LABORATORY #8

Exploring the Use of the SensorLib & 3-Axis Accelerometer

OBJECTIVES

- Increase knowledge and experience with the DK-TM4C123G Development Kit
- Continued use of the 'C' programming language
- Increases understanding of the engineering design process
- Introduction to the SensorLib library
- Utilize an off-microcontroller, on-board I2C device, the MPU9150

PURPOSE

The purpose of this laboratory exercise is to utilize the TivaWare Sensor Library to interface to the MPU9150 on the DK-TM4C123G development board and use the accelerometer data to display functionality similar to that of a bubble-level.

SPECIFICATIONS

For this laboratory exercise, it is required to demonstrate to the appropriate faculty that the 'C' software project created in conjunction with the development board which is able to run under debug mode and completes the assignment requirements dictated below.
This program must make calls to the SensorLib, the adjunct library for the on-board sensors, and read the data that is generated from the accelerometer in the MPU9150. This data then must be able to be displayed to the user along with a splash screen and UART menu, as previously implemented in prior laboratory exercises.

PROCEDURE

The code is completed in the 'C' programming language with all sources listed in the process as they are used. The TivaWare® TM4C123G Development Kit is the only hardware utilized for this procedure. All new technical terms and acronyms are defined in Appendix D at the end of this report.

Process:

1. Code Retrieval
   1.1. This program was built from the Lab6.c code © Andrea Gray 2019 for reliability and compiling reassurance purposes.
   1.2. The code associated with the MPU-9150 was then imported into the Lab6.c file
2. Overview
   2.1. The objective of this software is to perform the following tasks successfully in 'C'
      2.1.1. Display accelerometer data
         2.1.1.1. Ideally as a bubble level functionality type display
      2.1.2. Splash screen as done in previous exercises
      2.1.3. I/O through UART to establish communication with the user and the program
         2.1.3.1. Quit option

2.1.4. Blinky “heartbeat”

3. Gathering Information
   3.1. Information was gathered from two primary sources
      3.1.1. The “Programming Example” in the Sensor Library for the MPU9150
      3.1.2. The qs-logger program in the DK-TM4C123G development package
         3.1.2.1. Specifically the acquire.c and its AcquireInit() function for I2C configuration is used
         3.1.2.2. This file cannot be compiled so only specific portions of the code as indicated were used
4. Sensors
   4.1. The 9-axis motion sensor allows for 9 channels of data to be read accurately with only a single chip mounted to the board
      4.1.1. The boards used in the laboratory already contain the chip and no peripherals were needed to be connected for this exercise
   4.2. The only sensor that is used for this assignment though is the accelerometer
   4.3. The motion sensor, MPU9150, has 2 signals for the sole use of the I2C and one entirely for interrupt use
   4.4. The sensors were not difficult to initiate, and there were no major errors (major being greater than an issue from mistyping or miscalculating; which I do a lot), and although [2] mentions a board error, this does not come into play due to the lack of need of a peripheral or wiring connections.
5. Sensor Communication
   5.1. To create the communication with the MPU, the main elements that are needed to establish such successfully were given in [2] and therefore very easy to implement; especially because the line where examples of the use of the element were given in said document
   5.2. The communication was created using the following
      5.2.1. The 12C3 peripheral was initialized
         5.2.1.1. This was a very easy initialization since an I2C peripheral has been used in a laboratory exercise before
      5.2.2. Creation of control structures for the I2C and mpu9150
         5.2.2.1. The lines where these can be found in the acquire.c document were given in [2]
      5.2.3. Types tI2CMInstance and tMPU9150 were required to be created through a structure for the sensor library
      5.2.4. I2C3 service routine must be created
         5.2.4.1. As the last few exercises have given me great experience using interrupts, the creation of the I2C3 interrupt was not difficult
            5.2.4.1.1. It was also important to ensure the inclusion of the interrupt in the interrupt service vector in startup_ewarm.c. While this is fundamental now, it has caused problems for me in the past, i.e. the FaultISR, and the emphasis on the inclusion in the startup file is crucial
      5.2.5. MPU9150 Callback function creation
         5.2.5.1. This call back function was a bit logically confusing at first, but once it was created and once the reference of the function, in the sensor library programming example, was understood, the function was easy to debug and incorporate into the program
      5.2.6. Initialization of the MPU9150
         5.2.6.1. The initialization process and its organization, for any item, has grown to be an enjoyable process to me.

5.2.6.1.1. The MPU9150Example() function was much help to the assurance of the correct initialization also.

5.2.7. Iteration of Acceleration data

5.2.7.1. The acceleration data was put into a three-value array which held the y value, x value, and z value respectively.

5.2.7.1.1. The creation and allocation of this data into the array assured that the data was easily arranged and accessible throughout the program.

6. The use of a callback function defined by the SensorLib is used in my code and through the use of the variable bMPU9150Done in an indefinite while loop, the ability to do commands while waiting for the 12C bus is awarded
7. Library Use
   7.1. The use of multiple #include header files had to be implemented into the code for the use of the SensorLib
      7.1.1. In this exercise, the sequence of the included files comes into play and it was necessary to use them in the order given in [2] so that the program will run and call header files as planned
      7.1.2. The files that were needed were
         7.1.2.1. hw_mpu9150.h
         7.1.2.2. hw_ak8975.h
         7.1.2.3. i2cm_drv.h
         7.1.2.4. mpu9150.h
   7.2. The addition of the SensorLib to the .a library files is also crucial to the functionality of the code.
      7.2.1. I almost forgot this step, but luckily, I reviewed [2] and caught my error before the compiler was called
8. Bubble Level
   8.1. The bubble level uses the GrCircleDraw() function, but I have used this function before in prior labs for the splash screen, so the use here was easily done
9. Final
   9.1. This lab was not at all difficult, and Dr. Davis saved me with his float to string function for the outputting of the accelerometer data
      9.1.1. I did not run into any significant problems, and the only help that was needed was the float to string function mentioned above

REPORT DISCUSSION

1. Definitions
   ⇒ **Accelerometer**: An instrument that measures the acceleration of a device between the x, y, and z coordinates with respect to gravitational pull additionally.
   ⇒ **Gyroscope**: An instrument with a rotational mechanism that allows for the spinning of the element between the axes to nullify the angle in which the instrument is mounted upon. This allows for an ideal environment for the stabilization of an object. It is often seen in the aid of the reduction of moment of a joint object.
   ⇒ **Magnetometer**: An instrument that measures the magnetism of an object. [4]
   ⇒ **Sensor Fusion**: Sensor fusion is the exactly what the name describes it to be. It is the conjunction, fusion, of multiple data entries from differing sensors for a more complex data analysis of an environment. [5]
2. The Sensor Library is not mandatory when communicating with the MPU-9150. The code could be written entirely from scratch, but it would be unnecessary and possibly detrimental to the

communication. The library makes the use of theses sensors much easier than interfacing to the device without the library would be. The library allows for most of the work to already be ready for the user to call from.

3. The MPU9150 designers included an INT pin in addition to the I2C bus on the IC most likely so that sampling data could be obtained simultaneously. INT = interrupt. This could mean that data can be interrupt driven and/or traditionally gathered through code in the main program.
4. Using a call-back function in the SensorLib rather than the polled I/O approach, which has been used prior to this laboratory exercise, is a very efficient addition. The call-back function allows for multiple tasks to be running at once. While one task is waiting to on the bus, another task can be underway. This increases productivity and decreases time used by a program drastically compared to a polled I/O which is inefficient because it will check on a task multiple times per second, maybe even every cycle, which can cause major delays in execution.
5. Functions that may be useful for drawing a moving bubble on the OLED of the TM4C123G development kit, sourced from the Graphics Library, are shown below:
   ⇒ GrCircleDraw()
   ⇒ GrCircleFill()
6. Beyond the bubble level, the MPU9150 could be used for implementations such as:
   ⇒ Stabilization of accelerometer data
   ⇒ Submarine navigation systems
   ⇒ Sensing the gravitational pull of the earth from the ERAU-Prescott location
   ⇒ Sensing the magnitude of an earthquake or vibrational disruption
7. Other sensors that can be communicated to/from using an I2C connection, from [6], are:
   ⇒ BMP180 Barometer
   ⇒ CM3218 Ambient Light Sensor
   ⇒ SHT21 Humidity and Temperature Sensor
8. Some surprises that I encountered in this exercise was the fact that, unlike my past projects, my code did not need the help of multiple peers and instructors to run. This exercise was fairly easy to implement. I also did not realize that the Z-axis on the accelerometer could be used and determined by the sensor. Now, it seems obvious, but when the project first ran, the level itself was very interesting to me. I also was shocked that the device knew which side it was resting on by moving the gravitational pull (9.8 m/s) between the axes. If this exercise were to be done again, I would wish that I was not behind on my other work so that more time could have been dedicated to this program and I could truly explore all of the uses and implementations of the motion sensor.

CONCLUSION

In conclusion, this laboratory exercise was not at all very challenging. It was a nice relief from the prior exercises that have been consuming any open opportunity this semester. Although the exercise was fairly simple to implement and debug, the implementation of the code and the final product was one of the most interesting. It is not initially thought by most that the TM4C123G development board houses an accelerometer that can gauge what angle the board is at and if it is currently moving or not. The use of this technology without there being any foreshadowed hint at the pre-included hardware was wonderful to witness. It may be that I am just not as adept to technology as some, or that I am easily entertained, but nonetheless, this laboratory exercise was one that I was excited to share with those around me. A truly rewarding and interesting learning experience.

APPENDICES:

APPENDIX A: Lab Code

```
//*****************************************************************************
// Author: Andrea Gray
//
// CEC322 Lab #8: Exploring the use of the SensorLib & 3-Axis Accelerometer
// Development reference: Lab6.c
//
// There are multiple errors in this code because many functions had to be multi-line
// split so that readability could be optimized for report documentation.
//
//*****************************************************************************
#include <stdio.h> // Standard IO
#include <stdint.h> // Supports integer usage
#include <stdbool.h> // Supports boolean logic
#include <string.h> // Supports strings

#include "inc/hw_ints.h" // Hardware
#include "inc/hw_memmap.h" // Hardware

#include "grlib/grlib.h" // Graphics library

#include "sensorlib/hw_mpu9150.h" // Defines MPU constants
#include "sensorlib/hw_ak8975.h" // Included in the example file
#include "sensorlib/i2cm_drv.h" // Defines many t prefixes and I2CM prefixes
#include "sensorlib/ak8975.h" // Defines tAK8975
#include "sensorlib/mpu9150.h" // Defines MPU functions

#include "drivers/cfal96x64x16.h" // For the OLED output

#include "driverlib/uart.h" // UART header file
#include "driverlib/gpio.h" // GPIO header file
#include "driverlib/fpu.h" // Lazy stacking header file
#include "driverlib/interrupt.h" // Interrupt usage header file
#include "driverlib/sysctl.h"
#include "driverlib/i2c.h" // I2C3 usage header file
#include "driverlib/debug.h" // Debugging header file

#define blinkyOnPeriod 100000 // defines how long the LED will stay lit
#define blinkyOffPeriod 100000 // defines how long the LED will remain off
#define MPU9150_I2C_ADDRESS  0x69 // MPU9150 I2C Address

//*****************************************************************************
//
// Globals
//
//*****************************************************************************
int whileLoop = 1;// Maintains indefinite while loop unless program exits
int bubble = 1; // Display the bubble level
int adata = 0; // Display the accelerometer data

float Accel[3]; // Store accelerometer data

int32_t blinkyHandler = 1;// Maintains LED 'heartbeat' unless specified otherwise
int32_t local_char; // Keeps the character input into PuTTy

uint32_t currentADC[10]; // The current value passed to the ADC

bool firstCycle = true; // Lets the program act accordingly if its the first ADC cycle or not
volatile bool bMPU9150Done; // Set when a MPU9150 command is ready

tI2CMInstance I2CInst; // Instance structure for the I2C master driver
tMPU9150 MPU9150Inst; // Instance structure for the MPU9150 sensor driver

tContext Context; // OLED drawing contextual structuring
tRectangle sRect; // Rectangle parameters for banner structuring

//*****************************************************************************
//
// Function Declarations
//
//*****************************************************************************
void splash(void); // splash screen display function
void blinky(void); // "Heartbeat" function
void clear(void); // clear the PuTTy window
void initializations(void); // Sets-up the software and hardware for usage
void printMenu(void); // re-prints the menu options to PuTTy
void putString(char *str); // prints a string to the OLED
void menuSwitch(void); // Switches between menu options depending on the input
int i2c_write(uint32_t i2cBaseAddress, unsigned char deviceAddress, unsigned NumCharWritten,
              unsigned char *WrittenBuffer); // Write to I2C
void DrawCircle(void); // Draw a circle for the Bubble Level
void MPU9150Callback(void *pvCallbackData, uint_fast8_t ui8Status); // Call back for MPU
void DisplayData(void); // Display accelerometer data


/*********************** Beginning of functions ***************************/


//*****************************************************************************
//
// The function that is called when the program is first executed to set up
// the TM4C123G board, the peripherals, the timers, and the interrupts.
//
//*****************************************************************************
void initializations(void) {

  //*****************************************************************************
  //                                  LED
  //*****************************************************************************
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG); // Enable GPIO G usage.
  while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG)); // Wait until LED is ready
  GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_2); // GPIO output is pin 2.

  //*****************************************************************************
  //                                  OLED
  //*****************************************************************************
  CFAL96x64x16Init(); // Initialize the OLED display driver.
  GrContextInit(&Context, &g_sCFAL96x64x16); // Initialize OLED graphics
  GrContextFontSet(&Context, g_psFontFixed6x8); // Fix the font type
```

```
109
110     //*****************************************************************************
111     //                                 UART
112     //*****************************************************************************
113     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); // Enable UART 0 usage.
114     while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0)) {}; // wait until UART 0 is ready
115     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // Enable GPIO A usage.
116     while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA)) {}; // Wait until GPIO A is ready
117     GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); // Set Pins A0 and A1 for UART
118
119     // Configure UART for 115200 baud rate, 8 in 1 operation
120     UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE
121                                                                | UART_CONFIG_PAR_NONE));
122
123     IntEnable(INT_UART0); // Enable the interrupt for UART 0
124     UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);  // Enable specific UART interrupt usage
125
126     //*****************************************************************************
127     //                               I2C3/MPU
128     //*****************************************************************************
129     SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C3); // Enable I2C3
130     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // Enable GPIO Port D
131     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); // Enable Port b for interrupt
132     SysCtlPeripheralReset(SYSCTL_PERIPH_I2C3); // Reset the I2C Peripheral
133     GPIOPinConfigure(GPIO_PD0_I2C3SCL); // Configure Port D Pin 0 to I2C SCL
134     GPIOPinConfigure(GPIO_PD1_I2C3SDA); // Configure Port D Pin 1 to I2C SDA
135
136     // Set Port D pin 0 and 1
137     GPIOPinTypeI2CSCL(GPIO_PORTD_BASE, GPIO_PIN_0);
138     GPIOPinTypeI2C(GPIO_PORTD_BASE, GPIO_PIN_1);
139
140     // Set up interrupts for Port B Pin 2
141     GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_2);
142     GPIOIntEnable(GPIO_PORTB_BASE, GPIO_PIN_2);
143     GPIOIntTypeSet(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_FALLING_EDGE);
144     IntEnable(INT_GPIOB);
145
146     I2CMasterInitExpClk(I2C3_BASE, SysCtlClockGet(), true); // Initialize I2C clock
147
148     // Enable I2C3 and the corresponding interrupts
149     I2CMasterEnable(I2C3_BASE);
150     I2CMasterIntEnable(I2C3_BASE);
151     IntEnable(INT_I2C3);
152     I2CMInit(&I2CInst, I2C3_BASE, INT_I2C3, 0xff, 0xff, SysCtlClockGet());
153
154     IntMasterEnable(); // Enable processor interrupts.
155
156     // Initialize the MPU9150 Driver.
157     MPU9150Init(&MPU9150Inst, &I2CInst, MPU9150_I2C_ADDRESS, MPU9150Callback, &MPU9150Inst);
158
159     // Configure MPU
160     MPU9150ReadModifyWrite(&MPU9150Inst, MPU9150_O_ACCEL_CONFIG, ~MPU9150_ACCEL_CONFIG_AFS_SEL_M,
161                            MPU9150_ACCEL_CONFIG_AFS_SEL_4G, MPU9150Callback, 0);
162
163     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // Enable the GPIO Peripheral used by the UART.
164     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); // Enable UART0
165
166     // Configure GPIO Pins for UART mode.
167     GPIOPinConfigure(GPIO_PA0_U0RX);
168     GPIOPinConfigure(GPIO_PA1_U0TX);
169     GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
170
171     //*****************************************************************************
172     //                              MISCELLANEOUS
173     //*****************************************************************************
174     clear(); // Clear any outputs or inputs from the PuTTY window.
175     GrFlush(&Context); // Flush any cached operations
176   }
177
178   //*****************************************************************************
179   //
180   // The error routine that is called if the driver library encounters an error.
181   //
182   //*****************************************************************************
183   #ifdef DEBUG
184   void __error__(char *pcFilename, uint32_t ui32Line) {}
185   #endif
186
187   //*****************************************************************************
188   //
189   // The UART interrupt handler.
190   //
191   //*****************************************************************************
192   void UARTIntHandler(void){
193     uint32_t ui32Status; // Holds the interrupt status
194     ui32Status = UARTIntStatus(UART0_BASE, true); // Get the interrupt status.
195     UARTIntClear(UART0_BASE, ui32Status); // Clear the interrupt for UART
196     // Loop while there are characters in the receive FIFO.
197     while(UARTCharsAvail(UART0_BASE)) {
198       // Read the next character from the UART and write it back to the UART.
199       local_char = UARTCharGetNonBlocking(UART0_BASE);
200       menuSwitch();
201     }
202   }
203
204   //*****************************************************************************
205   //
206   // PuTTY window clearing function.
207   //
208   //*****************************************************************************
209   void clear() { UARTCharPut(UART0_BASE, 12); }
210
211   //*****************************************************************************
212   //
213   // Using the character output function as a base for a parent function
214   // used to output an entire string to the OLED one character at a time.
215   //
216   //*****************************************************************************
```

```
void putString(char *str) {
  for(int i = 0; i < strlen(str); i++) {
    UARTCharPut(UART0_BASE, str[i]);
  }
}

//*****************************************************************************
//
// Print menu function that takes the complete menu as a string and
// utilizes the print string function created above to output the entire menu
// in one transmission block to the PuTTY window.
//
//*****************************************************************************
void printMenu() {
    // Menu is split SOLELY for report readability
  char*menu = "\rMenu Selection: \n\rB - Display Bubble Level\n\rC - Erase Terminal Window\n\rL - Flash LED
              \n\rM - Print the Menu\n\rQ - Quit this program\n\rV - Display Accelerometer Data\n\r";
  putString(menu);
}

//*****************************************************************
//
// If the input character is not invalid, begin the character matching
// statment below through the switch statment and act accordingly to
// the user input.
//
//*****************************************************************
void menuSwitch() {
  if (local_char != -1) { // Run only if the character in PuTTy is valid
    UARTCharPut(UART0_BASE, local_char); // Send a character to UART.

    // Begin character input matching to menu option.
    switch(local_char) {
    case 'B': // Display Bubble Level
      if (bubble == 0) {
        bubble = 1;
        adata = 0;
      }
      else {
        bubble = 0;
        adata = 1;
      }
      break;

    case 'C': // Clear PuTTY window
      clear();
      break;

    case 'L': //LED toggle
      if(blinkyHandler == 0)
        blinkyHandler = 1;
      else
        blinkyHandler = 0;
      GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, 0);
      break;

    case 'M': // Re-print menu
      UARTCharPut(UART0_BASE, 5);
      printMenu();
      break;

    case 'Q': // Quit program
      IntMasterDisable();
      putString("\n\rBYE!"); // Goodbye message to PuTTy
      sRect.i16XMin = 0;
      sRect.i16YMin = 0;
      sRect.i16XMax = GrContextDpyWidthGet(&Context) - 1;
      sRect.i16YMax = GrContextDpyHeightGet(&Context) - 1;
      GrContextForegroundSet(&Context, ClrBlack);
      GrContextBackgroundSet(&Context, ClrBlack);
      GrRectFill(&Context, &sRect);
      GrContextForegroundSet(&Context, ClrRed);
      GrContextFontSet(&Context, g_psFontFixed6x8);
      // Goodbye message to OLED in red.
      GrStringDrawCentered(&Context, "Goodbye", -1, GrContextDpyWidthGet(&Context) / 2, 30, false);
      // If the user said to quit the whileLoop will NO LONGER be able to be ran
      whileLoop = 0;
      break;

    case 'V': // Display accelerometer values
      if (adata == 0) {
        adata = 1;
        bubble = 0;
      }
      else {
        adata = 0;
        bubble = 1;
      }
      break;

    default:
      char invalid[25] = "\n\rInvalid. Try Again: ";
      char*ptr = invalid;
      putString(ptr);
      break;
    }
  }
}

//*****************************************************************************
//
// Blinky LED "heartbeat" function.
//
//*****************************************************************************
void blinky() {
  if(blinkyHandler == blinkyOnPeriod) {
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, GPIO_PIN_2);
    blinkyHandler = ~blinkyOffPeriod;
  }
```

```
  }

  if(blinkyHandler == -1) {
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, 0);
    blinkyHandler = 1;
  }
}

//*****************************************************************************
//
// Splash Screen
//
//*****************************************************************************
void splash() {
  // splash screen display parameters
  tRectangle screen;
  screen.i16XMin = 0;
  screen.i16XMax = 96; // Maximum width of the OLED
  screen.i16YMin = 0;
  screen.i16YMax = 64; // Maximum height of the OLED

  // clear the screen from any reminaing displays
  GrContextForegroundSet(&Context, ClrBlack);
  GrRectFill(&Context, &screen);

  // loading block parameters
  for(int length = 10; length <= 86; length++) {
    tRectangle loading;
    loading.i16XMin = 9;
    loading.i16XMax = length;
    loading.i16YMin = 26;
    loading.i16YMax = 39;
    GrContextForegroundSet(&Context, ClrSalmon);
    if (length%10 == 0) {
      GrStringDrawCentered(&Context, "Loading.  ", 11, 48, 20, true);
    }
    else if (length%10 == 1) {
      GrStringDrawCentered(&Context, "Loading.. ", 11, 48, 20, true);
    }
    else if (length%10 == 2) {
      GrStringDrawCentered(&Context, "Loading...", 11, 48, 20, true);
    }
    else {
      GrStringDrawCentered(&Context, "Loading   ", 11, 48, 20, true);
    }
    SysCtlDelay(150000);
    GrContextForegroundSet(&Context, ClrDeepSkyBlue);
    GrRectFill(&Context, &loading);
  }

  // clear the screen so that it is set for the main screen
  GrContextForegroundSet(&Context, ClrBlack);
  GrRectFill(&Context, &screen);
}


//*****************************************************************************
//
// Motion sensor interrupt handler
//
//*****************************************************************************
void MotionSensorIntHandler(void) {
  uint32_t ulStatus; // Holds the interrupt status
  ulStatus = GPIOIntStatus(GPIO_PORTB_BASE, true); // Get interrupt status
  GPIOIntClear(GPIO_PORTB_BASE, ulStatus); // Clear pin interrupts
}

//*****************************************************************************
//
// Writes to the I2C
//
//*****************************************************************************
int i2c_write(uint32_t i2cBaseAddress, unsigned char deviceAddress, unsigned NumCharWritten, unsigned char *WrittenBuffer) {
  int requested = 0;
  if (NumCharWritten == 0) {
    return requested;
  }

  while(I2CMasterBusy(i2cBaseAddress)); // Check
  if (NumCharWritten == 1) {
    I2CMasterSlaveAddrSet(i2cBaseAddress, deviceAddress , false); // Set the slave address with no recieving
    I2CMasterDataPut(i2cBaseAddress, WrittenBuffer[0]); // Give data to the I2C
    I2CMasterControl(i2cBaseAddress, I2C_MASTER_CMD_SINGLE_SEND); // Send command to I2C
    while(I2CMasterBusy(i2cBaseAddress)); // Wait until finished
  }

  else { // When the number of characters written is greater than 1
    int burst = (NumCharWritten - 2);
    int index = 0;

    I2CMasterSlaveAddrSet(i2cBaseAddress, deviceAddress , false); // Set the slave address with no recieving
    I2CMasterDataPut(i2cBaseAddress, WrittenBuffer[index++]); // Start reading
    I2CMasterControl(i2cBaseAddress, I2C_MASTER_CMD_BURST_SEND_START); // Begin burst
    while(I2CMasterBusy(i2cBaseAddress)); // Wait until done

    while (burst > 0) {
      I2CMasterDataPut(i2cBaseAddress, WrittenBuffer[index++]); // Begin reading
      I2CMasterControl(i2cBaseAddress, I2C_MASTER_CMD_BURST_SEND_CONT); // Continue burst
      while(I2CMasterBusy(i2cBaseAddress)); // Wait until finshed
      burst--; //decrement
    }

    I2CMasterDataPut(i2cBaseAddress, WrittenBuffer[index++]); // Start reading again
    I2CMasterControl(i2cBaseAddress, I2C_MASTER_CMD_BURST_SEND_FINISH); // Stop burst
    while(I2CMasterBusy(i2cBaseAddress)); // Wait until done
  }
  return(requested);
}
```

```
433
434   //*****************************************************************************
435   //
436   // Interrupt handler for I2C3
437   //
438   //*****************************************************************************
439   void i2c3IntHandler(void) {
440     I2CMasterIntClear(I2C3_BASE); // Clear the interrupt
441     I2CMIntHandler(&I2CInst); // I2CM interrupt handler provided by sensor library
442   }
443
444   //*****************************************************************************
445   //
446   // Draw a circle using accelerometer data
447   //
448   //*****************************************************************************
449   void DrawCircle(void) {
450     int32_t xCoord = (int)((-48 * Accel[1]) / 10) + 48; // hold the horizontal center point
451     int32_t yCoord = (int)((-32 * Accel[0]) / 10) + 32; // hold the vertical center point
452     GrContextForegroundSet(&Context, ClrSilver); // Sets the color to silver
453     GrCircleDraw(&Context, xCoord, yCoord, 2); // Draws a circle with radius 2
454     GrCircleFill(&Context, xCoord, yCoord, 2); // Fills in the circle
455     GrContextForegroundSet(&Context, ClrRed);
456     GrContextFontSet(&Context, g_psFontFixed6x8);
457     GrStringDrawCentered(&Context, "+", -1, GrContextDpyWidthGet(&Context) / 2, 30, false);
458
459   }
460
461   //*****************************************************************************
462   //
463   // Callback for when MPU9150 transactions are done
464   //
465   //*****************************************************************************
466   void MPU9150Callback(void *pvCallbackData, uint_fast8_t ui8Status) {
467     if(ui8Status != I2CM_STATUS_SUCCESS) { // Error check
468       whileLoop = 0; // If an error occurs, the program quits
469     }
470     bMPU9150Done = true; // Complete
471   }
472
473   //*****************************************************************************
474   //
475   // Float to String conversion
476   // (c)2019 Dr. Brian Davis
477   //
478   //*****************************************************************************
479   char *floatToString(float x)
480   {
481     // Use of static allocation here BAD code style
482     // Functional but quick & dirty use only
483     static char buf[32];
484     int mantissa = (int)(x);
485     int fraction = abs((int)((x - mantissa)*1000));
486     if (fraction > 99) {
487       sprintf(buf, "%d.%d    ", mantissa, fraction);
488     }
489     else if (fraction > 9) {
490       sprintf(buf, "%d.0%d    ", mantissa, fraction);
491     }
492     else {
493       sprintf(buf, "%d.00%d    ", mantissa, fraction);
494     }
495     return buf;
496   }
497
498   //*****************************************************************************
499   //
500   // Displaying the Accelerometer data to the OLED
501   //
502   //*****************************************************************************
503   void DisplayData(void) {
504     char xStr[16];
505     char yStr[16];
506     char zStr[16];
507     sprintf(xStr, "%s", floatToString(Accel[1]));
508     sprintf(yStr, "%s", floatToString(Accel[0]));
509     sprintf(zStr, "%s", floatToString(Accel[2]));
510     char*ptrx = xStr;
511     char*ptry = yStr;
512     char*ptrz = zStr;
513     GrContextForegroundSet(&Context, ClrWhite);
514     GrContextFontSet(&Context, g_psFontFixed6x8);
515     GrStringDraw(&Context, "X: ", 3, 5, 20, false);
516     GrStringDraw(&Context, ptrx, strlen(ptrx), 20, 20, false);
517     GrStringDraw(&Context, "Y: ", 3, 5, 30, false);
518     GrStringDraw(&Context, ptry, strlen(ptry), 20, 30, false);
519     GrStringDraw(&Context, "Z: ", 3, 5, 40, false);
520     GrStringDraw(&Context, ptrz, strlen(ptrz), 20, 40, false);
521   }
522
523   //*****************************************************************************
524   //
525   // Main function
526   //
527   //*****************************************************************************
528   int main(void) {
529     // Enable lazy stacking for interrupt handlers.  This allows floating-point
530     // instructions to be used within interrupt handlers, but at the expense of
531     // extra stack usage.
532     FPULazyStackingEnable();
533
534     // Setting the clock to run directly from the crystal.
535     SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
536
537     IntMasterDisable(); // Disable interrupts while set up is in progress
538
539     // Calling the initial functions to set up TM4C123G, its peripherals,
540     // OLED and clear PuTTy window for menu output.
```

```
    initializations();
    splash(); // Prints out the splash screen to OLED.
    clear(); // Clears the PuTTy window for neatness.
    printMenu(); // Prints the UART menu for user IO.

    // Fill the part of the screen defined below to create a banner.
    sRect.i16XMin = 0;
    sRect.i16YMin = 0;
    sRect.i16XMax = GrContextDpyWidthGet(&Context) - 1;
    sRect.i16YMax = 9;
    GrContextForegroundSet(&Context, ClrSlateGray);
    GrRectFill(&Context, &sRect);
    GrContextForegroundSet(&Context, ClrWhite);
    GrStringDrawCentered(&Context, "Gray Bubble Level", -1, GrContextDpyWidthGet(&Context) / 2, 4, 0);

    IntMasterEnable(); // Enables Interrupts

    //*****************************************************************************
    //
    // The main while loop the function will stay in unless acted upon by the
    // user through UART or an interrupt timer is being serviced.
    //
    //*****************************************************************************
    while(whileLoop != 0) {
      bMPU9150Done = false; // Indicate that the sensor is in progress
      MPU9150DataRead(&MPU9150Inst, MPU9150Callback, 0); // Activate motion sensor
      while(!bMPU9150Done); // Wait until motion sensor is finished

      // Get the new accelerometer readings
      MPU9150DataAccelGetFloat(&MPU9150Inst, &Accel[0], &Accel[1], &Accel[2]);

      // Clear the screen
      tRectangle rect;
      rect.i16XMin = 0;
      rect.i16XMax = GrContextDpyWidthGet(&Context);
      rect.i16YMin = 0;
      rect.i16YMax = 63;
      GrContextForegroundSet(&Context, ClrBlack);
      GrRectFill(&Context, &rect);

      // Draw the bubble for the bubble level
      if(bubble != 0) {
        DrawCircle();
      }

      // Display Accelerometer Data
      if(adata != 0) {
        DisplayData();
      }

      // Calling the 'heartbeat' function if specified to do so.
      if(blinkyHandler != 0) {
        blinky();
        blinkyHandler++;
      }

      // Checking to see if the user has input a character and acting accordingly.
      while(UARTCharsAvail(UART0_BASE)) {
        local_char = UARTCharGetNonBlocking(UART0_BASE);
        menuSwitch();
      }
    }
  }
```

APPENDIX B: Instructor Sign Off

Instructor Sign-off

Each laboratory group must complete a copy of this sheet.

Student A Name Andrea Gray

Student B Name Ø

Development Kit Used #25

Lab Station Used #16

Starting Example Project Lab 6.C

1. Demonstrate operation of a software project which reads the accelerometer data from the MPU-9150 on the EK-TM4C123G.

   Ideally this project will comply with the requirements in Step # 3 of the procedure.

   A functioning bubble level has met all the requirements of the assignment.
   Display of (3) accelerometer values {X, Y, Z} is sufficient, but falls slightly short of the assignment goals.

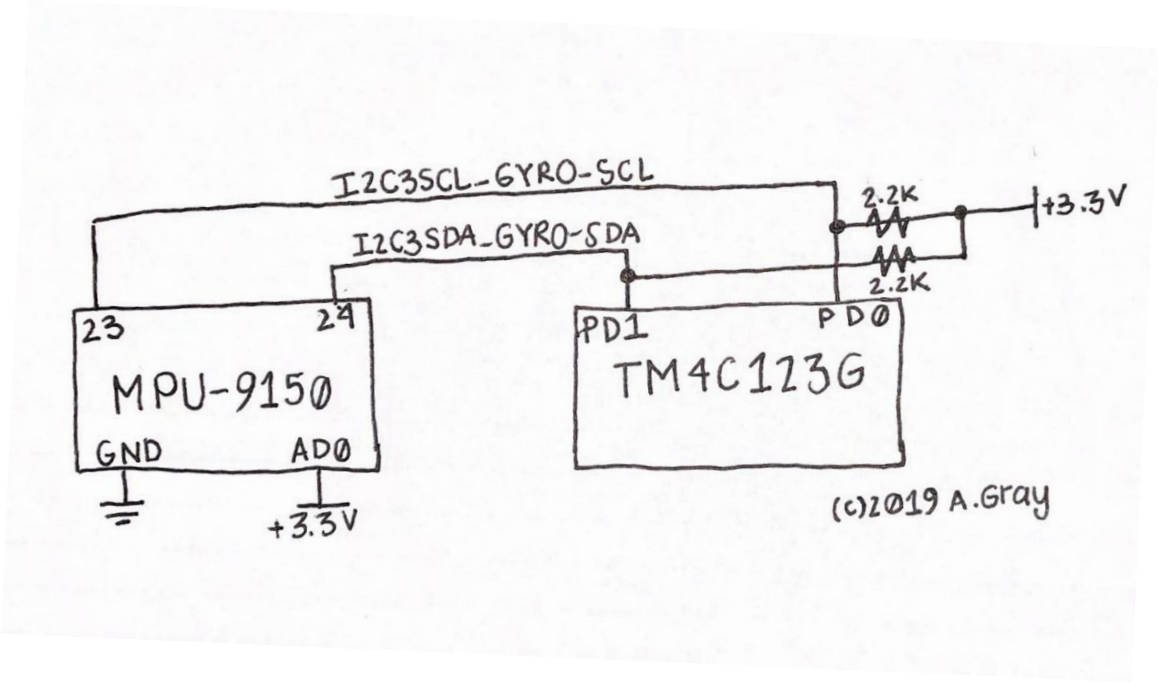
Meets All Req
- # And bubble - Small bubble

Instructors's Initials [signature] Date/Time 9-Apr 4:35pm

Good work

5
/ 5

APPENDIX C: Schematic



APPENDIX D: Definitions

I2C: Inter-Integrated Circuit

SensorLib: The sensor library that is included and accessed by this code for the instantiation and use of the hardware sensors.

APPENDIX E: Citation

[1] *Tiva™ TM4C123GH6PM Microcontroller: Data Sheet*. Rev. E. [eBook]. Austin: Texas Instruments, 2014, p.898. Available: http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf. [Accessed: 3-APR-2019].

[2] Dr. Brian Davis, *Laboratory Exercise #8*, Embry-Riddle Aeronautical University Prescott Department of Computer, Mechanical, and Software Engineering, 2019.

[3] *TivaWare™ Peripheral Driver Library: User's Guide*. Rev. 2016. [eBook]. Austin: Texas Instruments, 2013, p.354-355. Available: http://www.ti.com/lit/ug/spmu298d/spmu298d.pdf. [Accessed: 3-APR-2019].

[4] Kionix, "Sensor Fusion", *Kionix: A ROHM Group Company*, 2018. [Online]. Available: https://www.kionix.com/sensor-fusion. [Accessed: 6-APR-2019].

[5] Preeti, Jain, "Magnometers", *EngineersGarage*, 2012. [Online]. Available: https://www.engineersgarage.com/articles/magnetometer. [Accessed: 8-APR-2019].

[6] *TivaWare™: Sensor Library: User's Guide.* Rev. 2016. [eBook]. Austin: Texas Instruments, 2015, p.3-4. Available: http://www.ti.com/lit/ug/spmu371d/spmu371d.pdf. [Accessed: 8-APR-2019].