EECS 690

Project 2 Report

Accelerometer and Gyroscope on a TI Tiva C TI\_TM4C1294NCPDT using FreeRTOS

Written By

Ben Sokol

Kaiser Mittenburg

The University of Kansas

Electrical Engineering and Computer Science

October 18, 2018

Copyright © 2018 by Ben Sokol and Kaiser Mittenburg.

All rights reserved.

# Abstract

I2C is a commonly used data bus protocol in systems using peripheral sensors. A common bus protocol is a necessity for streamlined programs and allows developers to access a sensor’s data without having to research or write API for each individual peripheral. I2C abstracts the individuality of each sensor into common higher-level code. The TI Tiva C TI\_TM4C1294NCPDT has access to many peripheral sensors using the BOOSTXL-SENSHUB Sensor Hub BoosterPack such as a barometer, thermometer, accelerometer and gyroscope. Thanks to I2C, data can easily be collected from all of these sensors.

# Revision History

The following table (*Table 2-1*) lists the revision history for this document.

**Table 2‑1 Revision History**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Description** |
| October 18, 2018 | 1.0 | Initial Release |

# Table of Contents

[1 Abstract 1](#_Toc527615556)

[2 Revision History 1](#_Toc527615557)

[3 Table of Contents 2](#_Toc527615558)

[4 List of Figures 2](#_Toc527615559)

[5 List of Tables 2](#_Toc527615560)

[6 Principles of Operation (POP) 3](#_Toc527615561)

[7 Data Structure Descriptions 3](#_Toc527615562)

[8 Function Descriptions 4](#_Toc527615563)

[**8.1 Function API** 4](#_Toc527615564)

[8.1.1 BMP180SimpleCallback 4](#_Toc527615565)

[8.1.2 Task\_BMP180\_Handler 4](#_Toc527615566)

[8.1.3 MPU9150SimpleCallback 4](#_Toc527615567)

[8.1.4 Task\_MPU9150\_Handler 4](#_Toc527615568)

[**8.2 Function Pseudo Code** 5](#_Toc527615569)

[**8.3 Referenced External Functions** 8](#_Toc527615570)

[9 Parameters 9](#_Toc527615571)

[10 Testing 10](#_Toc527615572)

[11 Lessons Learned 11](#_Toc527615573)

[Appendix A – Program Source Code 12](#_Toc527615574)

[**A.1 Task\_MPU9150\_Handler.c** 12](#_Toc527615575)

[**A.2 Task\_BMP180\_Handler.c** 16](#_Toc527615576)

# List of Figures

[Figure 10‑1 Board movement for gryoscope testing 10](#_Toc527615577)

# List of Tables

[Table 2‑1 Revision History 1](#_Toc527615578)

[Table 8‑1 Functions 5](#_Toc527615579)

[Table 8‑2 Referenced External Function 8](#_Toc527615580)

[Table 9‑1 Parameters 9](#_Toc527615581)

[Table 10‑1 Accelerometer Testing 10](#_Toc527615582)

[Table 10‑2 Gyroscope Testing 10](#_Toc527615583)

# Principles of Operation (POP)

The first step to collecting data from the various sensors is to initialize the I2C bus. It is safe for every module intending to use I2C to call the initialization function for I2C because it will only initialize on the first call and fall through on all subsequent calls. I2C7\_Handler is our driver for I2C. The task in charge of collecting data from the accelerometer and gyroscope is Task\_MPU9150\_Handler. When the program is started, this task will be scheduled.

Task\_MPU9150\_Handler initializes the I2C bus, as it will rely on access to the bus to obtain sensor data. Next, a binary semaphore is created for synchronization purposes. The MPU9150 is then initialized and given our instance of the I2C pipeline, and our callback function that will be executed after data collection. We then loop indefinitely until externally terminated. The loop will continuously take the semaphore we created and poll the MPU9150 sensor for data. We then use our third party ReportData to queue our data for serialization to be printed on a separate console for viewing.

After each collection of data from the sensor, our provided callback is executed. We do not need to do much in this callback. We simply check for error from the data collection and give the semaphore back. Giving the semaphore back allows the mainline of our task to take the semaphore again. If the mainline loop has made it back to the top and attempts to take the semaphore before the callback gives it, it was block until the semaphore is given. This achieves synchronization, and allows for complete, and un-wasted polling of data.

This project uses the I2C7, BMP180, MPU9150, and UART modules.

# Data Structure Descriptions

There are no data structures used in this project besides ReportData\_Item. As soon as data is collected, it is immediately reported without storing it in a data structure besides basic C types (i.e. float).

# Function Descriptions

The following section contains function API, descriptions of each function, and any external functions referenced within those functions.

## **8.1 Function API**

### 8.1.1 BMP180SimpleCallback

Callback routine for the BMP180 for when transactions have completed

**Prototype:**

extern void BMP180SimpleCallback(void\* pvData, uint\_fast8\_t ui8Status);

**Parameters:**

pvData is data passed to callback function.

ui8Status is the status of the BMP180.

**Returns:**

None.

### 8.1.2 Task\_BMP180\_Handler

Task to report Temperature and Pressure from BMP180 sensor

**Prototype:**

extern void Task\_BMP180\_Handler(void\* pvParameters);

**Parameters:**

pvParameters is data passed to function. As function does not need any parameters, NULL should be used.

**Returns:**

None.

### 8.1.3 MPU9150SimpleCallback

Callback routine for the MPU9150 for when transactions have completed

**Prototype:**

extern void MPU9150SimpleCallback(void\* pvData, uint\_fast8\_t ui8Status);

**Parameters:**

pvData is data passed to callback function.

ui8Status is the status of the MPU9150.

**Returns:**

None.

### 8.1.4 Task\_MPU9150\_Handler

Task to report Accelerometer and Gyroscope from MPU9150 sensor

**Prototype:**

extern void Task\_MPU9150\_Handler(void\* pvParameters);

**Parameters:**

pvParameters is data passed to function. As function does not need any parameters, NULL should be used.

**Returns:**

None.

## **8.2 Function Pseudo Code**

The following table (Table 8-1) contains pseudo code for the specified function.

**Table 8‑1 Functions**

|  |  |
| --- | --- |
| **Function Name** | **Function Pseudo Code** |
| **BMP180SimpleCallback** | SET xHigherPriorityTaskWoken TO false  INCREMENT BMP180\_Callbacks\_Nbr  IF ui8Status IS NOT EQUAL I2CM\_STATUS\_SUCCESS THEN  CALL UARTprintf with BMP180 error and ui8Status  ENDIF  SET BMP180SimpleDone TO true  CALL xSemaphoreGiveFromISR with BMP180\_Semaphore and a  reference to xHigherPriorityTaskWoken  Call portYIELD\_FROM\_ISR with xHigherPriorityTaskWoken |
| **Task\_BMP180\_Handler** | CALL UARTStdio\_Initialization  CALL I2C7\_Initialization  CALL vSemaphoreCreateBinary WITH BMP180\_Semaphore  SET BMP180SimpleDone TO false  CALL BMP180Init with reference to sBMP180,  I2C7\_Instance\_Ref, BMP180\_ADDRESS,  BMP180SimpleCallback, and 0  CALL xSemaphoreTake with BMP180\_Semaphore and  portMAX\_DELAY  CALL UARTprintf to report BMP180 as Initialized  // Begin data collection and reporting  WHILE TRUE  INIT float fTemperature TO 0.0  INIT float fPressure TO 0.0  CALL BMP180DataRead WITH reference to sBMP180,  BMP180SimpleCallback, and 0  CALL xSemaphoreTake WITH BMP180\_Semaphore AND  portMAX\_DELAY  CALL BMP180DataPressureGetFloat WITH reference to  sBMP180 AND reference to fPressure  CALL BMP180DataTemperatureGetFloat WITH reference to  sBMP180 AND reference to fTemperature  INIT ReportData\_Item WITH NAME pressureItem  SET pressureItem.TimeStamp TO xPortSysTickCount  SET pressureItem.ReportName TO 0002  SET pressureItem.ReportValueType\_Flg TO 0b0001  SET pressureItem.ReportValue\_0 TO the bit value  contained within fPressure  SET pressureItem.ReportValue\_1 TO 0  SET pressureItem.ReportValue\_2 TO 0  SET pressureItem.ReportValue\_3 TO 0  INIT ReportData\_Item WITH NAME tempItem  SET tempItem.TimeStamp TO xPortSysTickCount  SET tempItem.ReportName TO 0003  SET tempItem.ReportValueType\_Flg TO 0b0001  SET tempItem.ReportValue\_0 TO the bit value  contained within fTemperature  SET tempItem.ReportValue\_1 TO 0  SET tempItem.ReportValue\_2 TO 0  SET tempItem.ReportValue\_3 TO 0  SEND reference to pressureItem to ReportData\_Queue  using xQueueSend  SEND reference to tempItem to ReportData\_Queue  using xQueueSend  CALL vTaskDelay with ((SysTickFrequency\*1000)/1000)  ENDWHILE |
| **MPU9150SimpleCallback** | SET xHigherPriorityTaskWoken TO false  INCREMENT MPU9150\_Callbacks\_Nbr  IF ui8Status IS NOT EQUAL I2CM\_STATUS\_SUCCESS THEN  CALL UARTprintf with MPU9150 error and ui8Status  ENDIF  SET MPU9150SimpleDone TO true  CALL xSemaphoreGiveFromISR with MPU9150\_Semaphore and  a reference to xHigherPriorityTaskWoken  Call portYIELD\_FROM\_ISR with xHigherPriorityTaskWoken |
| **Task\_MPU9150\_Handler** | CALL UARTStdio\_Initialization  CALL I2C7\_Initialization  CALL vSemaphoreCreateBinary WITH MPU9150\_Semaphore  SET MPU9150SimpleDone TO false  CALL MPU9150Init with reference to sMPU9150,  I2C7\_Instance\_Ref, MPU9150\_ADDRESS,  MPU9150SimpleCallback, and 0  CALL xSemaphoreTake with MPU9150\_Semaphore and  portMAX\_DELAY  CALL UARTprintf to report MPU9150 as Initialized  // Begin data collection and reporting  WHILE TRUE  INIT float fAccelX TO 0.0  INIT float fAccelY TO 0.0  INIT float fAccelZ TO 0.0  INIT float fGyroX TO 0.0  INIT float fGyroY TO 0.0  INIT float fGyroZ TO 0.0  CALL MPU9150DataRead WITH reference to s MPU9150,  MPU9150SimpleCallback, and 0  CALL xSemaphoreTake WITH MPU9150\_Semaphore AND  portMAX\_DELAY  CALL MPU9150DataAccelGetFloat WITH reference to  sMPU9150 AND references to fAccelX,  fAccelY, and fAccelZ  CALL MPU9150DataGyroGetFloat WITH reference to  sMPU9150 AND references to fGyroX,  fGyroY, and fGyroZ  INIT ReportData\_Item WITH NAME itemAccel  SET itemAccel.TimeStamp TO xPortSysTickCount  SET itemAccel.ReportName TO 0004  SET itemAccel.ReportValueType\_Flg TO 0b0111  SET itemAccel.ReportValue\_0 TO the bit value  contained within fAccelX  SET itemAccel.ReportValue\_1 TO the bit value  contained within fAccelY  SET itemAccel.ReportValue\_2 TO the bit value  contained within fAccelZ  SET itemAccel.ReportValue\_3 TO 0  INIT ReportData\_Item WITH NAME itemGyro  SET itemGyro.TimeStamp TO xPortSysTickCount  SET itemGyro.ReportName TO 0005  SET itemGyro.ReportValueType\_Flg TO 0b0111  SET itemGyro.ReportValue\_0 TO the bit value  contained within fGyroX  SET itemGyro.ReportValue\_1 TO the bit value  contained within fGyroY  SET itemGyro.ReportValue\_2 TO the bit value  contained within fGyroZ  SET itemGyro.ReportValue\_3 TO 0  SEND reference to itemAccel to ReportData\_Queue  using xQueueSend  SEND reference to itemGyro to ReportData\_Queue  using xQueueSend  CALL vTaskDelay with ((SysTickFrequency\*1000)/1000)  ENDWHILE |

## **8.3 Referenced External Functions**

The following table (Table 8-2) contains any externally referenced functions for the specified function.

**Table 8‑2 Referenced External Function**

|  |  |
| --- | --- |
| **Function Name** | **Referenced External Functions** |
| **BMP180SimpleCallback** | UARTprintf  xSemaphoreGiveFromISR  portYIELD\_PROM\_ISR |
| **Task\_BMP180\_Handler** | BMP180Init  BMP180DataPressureGetFloat  BMP180DataRead  BMP180DataTemperatureGetFloat  I2C7\_Initialization  UARTprintf  UARTStdio\_Initialization  vSemaphoreCreateBinary  vTaskDelay  xQueueSend  xSemaphoreTake |
| **MPU9150SimpleCallback** | UARTprintf  xSemaphoreGiveFromISR  portYIELD\_PROM\_ISR |
| **Task\_MPU9150\_Handler** | I2C7\_Initialization  MPU9150DataAccelGetFloat  MPU9150DataGyroGetFloat  MPU9150DataRead  MPU9150Init  UARTprintf  UARTStdio\_Initialization  vSemaphoreCreateBinary  vTaskDelay  xSemaphoreTake  xQueueSend |

# Parameters

# 

The following table (*Table 9-1*) contains a list of parameters used in this project

**Table 9‑1 Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Parameter Type** | **Parameter**  **Default Value** | **Parameter Description** |
| **BMP180\_ADDRESS** | const int | 0x77 | The I2C Address of the BMP180 |
| **sBMP180** | tBMP180 | N/A | The BMP180 control block |
| **BMP180SimpleDone** | volatile bool | false | A boolean that is set when an I2C transaction is completed. |
| **BMP180\_Callbacks\_Nbr** | uint32\_t | 0 | The number of BMP180 callbacks taken. |
| **BMP180\_Semaphore** | xSemaphoreHandle | N/A | Semaphore to indicate completion of the callback operation |
| **MPU9150\_ADDRESS** | const int | 0x68 | The I2C Address of the MPU9150 |
| **sMPU9150** | tMPU9150 | N/A | The MPU9150 control block |
| **MPU9150SimpleDone** | volatile bool | false | A boolean that is set when an I2C transaction is completed. |
| **MPU9150\_Callbacks\_Nbr** | uint32\_t | 0 | The number of MPU9150 callbacks taken. |
| **MPU9150\_Semaphore** | xSemaphoreHandle | N/A | Semaphore to indicate completion of the callback operation |

# 

# Testing

**Table 10‑1 Accelerometer Testing**

Our initial testing was of the BMP180 temperature and pressure sensors. We decided that while this was not required for this project, it would be useful practice and would provide a easy way to check if the data received from the sensor was accurate (this is because we knew the temperature in the lab would be between 70º F and 80º F). When testing the temperature, we received a value of 26.4º Celsius, which when converted to Fahrenheit is ~79.52º. The data we received from the pressure sensor showed 99,623 Pascals, given that normal atmospheric pressure at sea level is 101,325 Pascals. Both of these values suggested the sensor was working correctly.

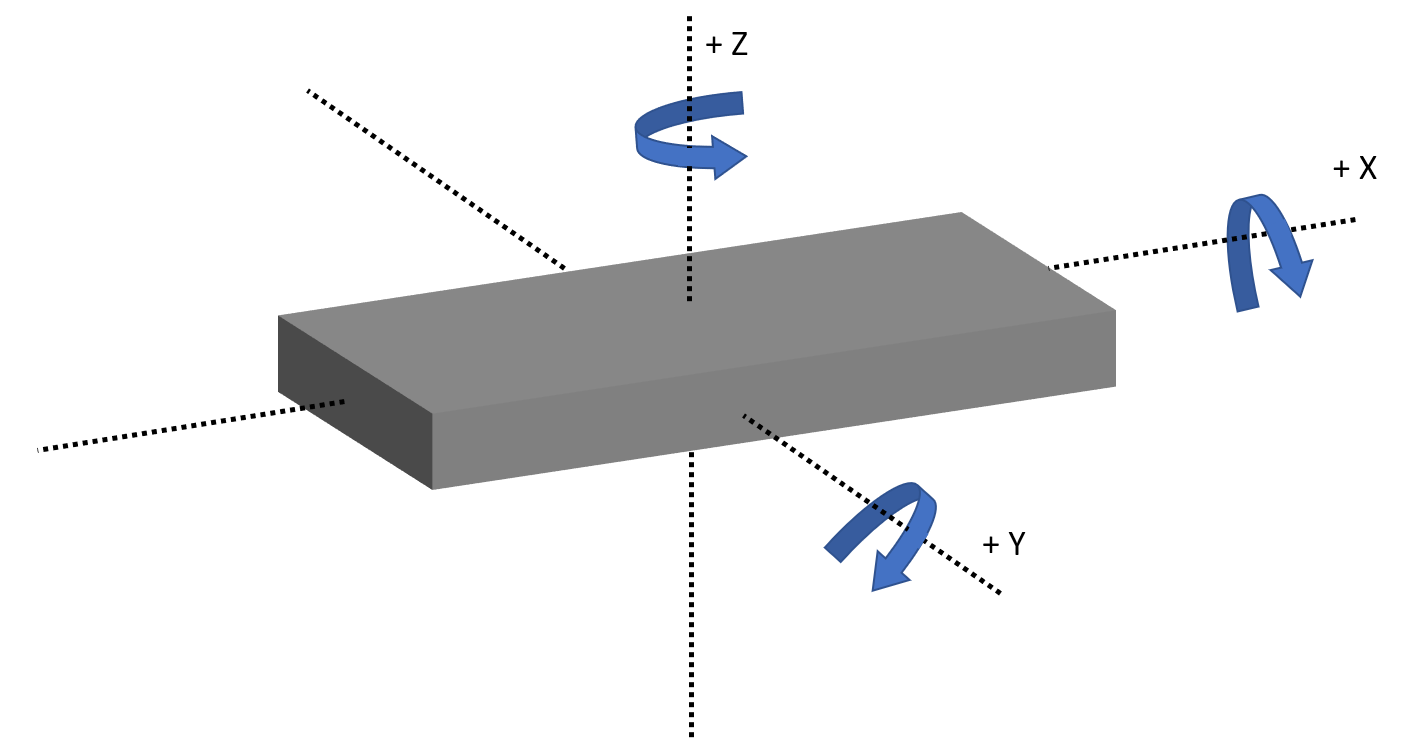
|  |  |  |  |
| --- | --- | --- | --- |
|  | **X** | **Y** | **Z** |
| **+ Z** | -0.065 | -0.113 | 9.682 |
| **- Z** | -0.180 | -0.496 | -9.864 |
| **+ X** | 8.593 | -0.405 | 0.046 |
| **- X** | -9.854 | -0.196 | -0.005 |
| **+ Y** | 0.606 | 9.558 | 0.206 |
| **- Y** | -0.709 | -9.819 | -0.232 |

Our data table (*Table 10-1*) shows data received from the accelerometer. By manipulating the board to sit on each of its six faces, we can observe gravity’s force affect the sensor on each axis in positive and negative direction. For the most part, gravity is being observed in the absolute range of ~9.6-9.8 while the other two axes are relatively close to zero. If the board was made to stand on one of its edges, it could be observed that the force of gravity would be acting in a ratio split between two axes.

**Figure 10‑1 Board movement for gryoscope testing**

**Table 10‑2 Gyroscope Testing**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **X** | **Y** | **Z** |
| **+ Z** | -0.116 | -0.667 | 3.980 |
| **- Z** | -0.221 | -0.249 | -3.857 |
| **+ X** | 3.102 | -0.325 | 0.333 |
| **- X** | -4.366 | 0.773 | -0.363 |
| **+ Y** | -0.732 | 3.605 | 0.398 |
| **- Y** | -0.344 | -4.336 | -0.776 |



The above table (*Table 10-2*) shows our data received from the gyroscope after moving the board in many directions (Figure 10-1). There are six possible cardinal directions for which the board can be rotated to be noticed by the gyroscope. In collecting our data, we attempted to rotate the board in only one direction while keeping the other axes level on their plane.

Both sets of data are plausible and suggest that we are receiving valid data from the accelerometer and gyroscope via the I2C bus.

# Lessons Learned

It can be noted how little I2C has been mentioned in the mainline and callback of our task. The generalization of the bus protocol, and the fact that all sensors’ API has abstracted away the need to deal with I2C has made it very simple to transfer data. Without a common bus protocol, data transfer would be very messy and cumbersome for the developer.

# Appendix A – Program Source Code

The following is a listing of all source code written for this project.

## **A.1 Task\_MPU9150\_Handler.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196 | */\*\**  *\* @Filename: Task\_MPU9150\_Handler.c*  *\* @Author: Kaiser Mittenburg and Ben Sokol*  *\* @Email: ben@bensokol.com*  *\* @Email: kaisermittenburg@gmail.com*  *\* @Created: October 4th, 2018 [1:02pm]*  *\* @Modified: October 18th, 2018 [6:45am]*  *\* @Version: 1.0.0*  *\**  *\* @Description: Periodically read and report accelerometer and*  *\* gyroscope readings.*  *\**  *\* Copyright (C) 2018 by Kaiser Mittenburg and Ben Sokol.*  *\* All Rights Reserved.*  *\*/*  *#include "inc/hw\_ints.h"*  *#include "inc/hw\_memmap.h"*  *#include "inc/hw\_sysctl.h"*  *#include "inc/hw\_types.h"*  *#include "inc/hw\_uart.h"*  *#include <math.h>*  *#include <stdarg.h>*  *#include <stdbool.h>*  *#include <stddef.h>*  *#include <stdint.h>*  *#include <stdio.h>*  *#include <stdlib.h>*  *#include "Drivers/I2C7\_Handler.h"*  *#include "Drivers/UARTStdio\_Initialization.h"*  *#include "Drivers/uartstdio.h"*  *#include "sensorlib/ak8975.h"*  *#include "sensorlib/hw\_ak8975.h"*  *#include "sensorlib/hw\_mpu9150.h"*  *#include "sensorlib/mpu9150.h"*  *#include "driverlib/gpio.h"*  *#include "driverlib/interrupt.h"*  *#include "driverlib/pin\_map.h"*  *#include "driverlib/sysctl.h"*  *#include "driverlib/timer.h"*  *#include "Tasks/Task\_ReportData.h"*  *#include "FreeRTOS.h"*  *#include "semphr.h"*  *#include "task.h"*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* External variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// Access to current SysTick*  **extern** **volatile** **long** **int** xPortSysTickCount**;**  *// SysTickClock Frequency*  *#define SysTickFrequency configTICK\_RATE\_HZ*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local constant variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// The I2C Address of the MPU9150*  **const** **int** MPU9150\_ADDRESS **=** **0x68;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// The MPU9150 control block*  tMPU9150 sMPU9150**;**  *// A boolean that is set when an I2C transaction is completed.*  **volatile** **bool** MPU9150SimpleDone **=** false**;**  *// The number of MPU9150 callbacks taken.*  **uint32\_t** MPU9150\_Callbacks\_Nbr **=** **0;**  *// Semaphore to indicate completion of the callback operation*  xSemaphoreHandle MPU9150\_Semaphore**;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task function declarations*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** MPU9150SimpleCallback**(void\*** pvData**,** **uint\_fast8\_t** ui8Status**);**  **extern** **void** Task\_MPU9150\_Handler**(void\*** pvParameters**);**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task function definitions*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: MPU9150SimpleCallback*  *\* Description: Callback Routine for the MPU9150 for when*  *\* transactions have completed*  *\* Parameters: void\* pvData*  *\* uint\_fast8\_t ui8Status*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **void** MPU9150SimpleCallback**(void\*** pvData**,** **uint\_fast8\_t** ui8Status**)** **{**  portBASE\_TYPE xHigherPriorityTaskWoken **=** pdFALSE**;**  MPU9150\_Callbacks\_Nbr**++;**  **if** **(**ui8Status **!=** I2CM\_STATUS\_SUCCESS**)** **{**  *// An error occurred*  UARTprintf**(**">>>>MPU9150 Error: %02X\n"**,** ui8Status**);**  **}**  *// Indicate that the I2C transaction has completed.*  MPU9150SimpleDone **=** true**;**  *// "Give" the MPU9150\_Semaphore*  xSemaphoreGiveFromISR**(**MPU9150\_Semaphore**,** **&**xHigherPriorityTaskWoken**);**  portYIELD\_FROM\_ISR**(**xHigherPriorityTaskWoken**);**  **}**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: Task\_MPU9150\_Handler*  *\* Description: Task to report Gyroscope and Accelerometer from*  *\* MPU9150 sensor*  *\* Parameters: void\* pvParameters;*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** Task\_MPU9150\_Handler**(void\*** pvParameters**)** **{**  *// Initialize UART*  UARTStdio\_Initialization**();**  *// Initialize I2C7*  I2C7\_Initialization**();**  *// Initialize MPU9150\_Semaphore*  vSemaphoreCreateBinary**(**MPU9150\_Semaphore**);**  *// Initialize the MPU9150.*  MPU9150SimpleDone **=** false**;**  MPU9150Init**(&**sMPU9150**,** I2C7\_Instance\_Ref**,** MPU9150\_ADDRESS**,**  MPU9150SimpleCallback**,** **0);**  xSemaphoreTake**(**MPU9150\_Semaphore**,** portMAX\_DELAY**);**  UARTprintf**(**">>>>MPU9150: Initialized!\n"**);**  *// Loop forever reading and reporting data from the MPU9150.*  **while** **(1)** **{**  **float** fAccelX **=** **0.0;**  **float** fAccelY **=** **0.0;**  **float** fAccelZ **=** **0.0;**  **float** fGyroX **=** **0.0;**  **float** fGyroY **=** **0.0;**  **float** fGyroZ **=** **0.0;**  *// Request a reading from the MPU9150.*  MPU9150DataRead**(&**sMPU9150**,** MPU9150SimpleCallback**,** **0);**  xSemaphoreTake**(**MPU9150\_Semaphore**,** portMAX\_DELAY**);**  *// Get the new accelerometer and pressure reading.*  MPU9150DataAccelGetFloat**(&**sMPU9150**,** **&**fAccelX**,** **&**fAccelY**,** **&**fAccelZ**);**  MPU9150DataGyroGetFloat**(&**sMPU9150**,** **&**fGyroX**,** **&**fGyroY**,** **&**fGyroZ**);**  *// By taking the reference of a float, casting that pointer to an*  *// int32\_t pointer, then dereferencing that, the float is converted*  *// to an int32\_t bitwise, without any conversions. This is done*  *// instead of using an assembly function such as Float\_to\_Int32.*  *// Create ReportData\_Item for Acceleration*  ReportData\_Item itemAccel**;**  itemAccel**.**TimeStamp **=** xPortSysTickCount**;**  itemAccel**.**ReportName **=** **0004;**  itemAccel**.**ReportValueType\_Flg **=** **0**b0111**;**  itemAccel**.**ReportValue\_0 **=** **\*(int32\_t\*)&**fAccelX**;**  itemAccel**.**ReportValue\_1 **=** **\*(int32\_t\*)&**fAccelY**;**  itemAccel**.**ReportValue\_2 **=** **\*(int32\_t\*)&**fAccelZ**;**  itemAccel**.**ReportValue\_3 **=** **0;**  *// Create ReportData\_Item for Gyroscope*  ReportData\_Item itemGyro**;**  itemGyro**.**TimeStamp **=** xPortSysTickCount**;**  itemGyro**.**ReportName **=** **0005;**  itemGyro**.**ReportValueType\_Flg **=** **0**b0111**;**  itemGyro**.**ReportValue\_0 **=** **\*(int32\_t\*)&**fGyroX**;**  itemGyro**.**ReportValue\_1 **=** **\*(int32\_t\*)&**fGyroY**;**  itemGyro**.**ReportValue\_2 **=** **\*(int32\_t\*)&**fGyroZ**;**  itemGyro**.**ReportValue\_3 **=** **0;**  *// Send ReportData\_Items to queue to print*  xQueueSend**(**ReportData\_Queue**,** **&**itemAccel**,** **0);**  xQueueSend**(**ReportData\_Queue**,** **&**itemGyro**,** **0);**  *// Delay*  vTaskDelay**((**SysTickFrequency **\*** **1000)** **/** **1000);**  **}**  **}** |

## **A.2 Task\_BMP180\_Handler.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184 | */\*\**  *\* @Filename: Task\_BMP180\_Handler.c*  *\* @Author: Kaiser Mittenburg and Ben Sokol*  *\* @Email: ben@bensokol.com*  *\* @Email: kaisermittenburg@gmail.com*  *\* @Created: October 2nd, 2018 [1:10pm]*  *\* @Modified: October 18th, 2018 [6:41am]*  *\* @Version: 1.0.0*  *\**  *\* @Description: Periodically read and report temperature and pressure*  *\* readings.*  *\**  *\* Copyright (C) 2018 by Kaiser Mittenburg and Ben Sokol.*  *\* All Rights Reserved.*  *\*/*  *#include "inc/hw\_ints.h"*  *#include "inc/hw\_memmap.h"*  *#include "inc/hw\_sysctl.h"*  *#include "inc/hw\_types.h"*  *#include "inc/hw\_uart.h"*  *#include <math.h>*  *#include <stdarg.h>*  *#include <stdbool.h>*  *#include <stddef.h>*  *#include <stdint.h>*  *#include <stdio.h>*  *#include <stdlib.h>*  *#include "Drivers/I2C7\_Handler.h"*  *#include "Drivers/UARTStdio\_Initialization.h"*  *#include "Drivers/uartstdio.h"*  *#include "sensorlib/bmp180.h"*  *#include "sensorlib/hw\_bmp180.h"*  *#include "sensorlib/i2cm\_drv.h"*  *#include "driverlib/gpio.h"*  *#include "driverlib/interrupt.h"*  *#include "driverlib/pin\_map.h"*  *#include "driverlib/sysctl.h"*  *#include "driverlib/timer.h"*  *#include "Tasks/Task\_ReportData.h"*  *#include "FreeRTOS.h"*  *#include "semphr.h"*  *#include "task.h"*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* External variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// Access to current SysTick*  **extern** **volatile** **long** **int** xPortSysTickCount**;**  *// SysTickClock Frequency*  *#define SysTickFrequency configTICK\_RATE\_HZ*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local constant variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// The I2C Address of the BMP180*  **const** **int** BMP180\_ADDRESS **=** **0x77;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// The BMP180 control block*  tBMP180 sBMP180**;**  *// A boolean that is set when an I2C transaction is completed.*  **volatile** **bool** BMP180SimpleDone **=** false**;**  *// The number of BMP180 callbacks taken.*  **uint32\_t** BMP180\_Callbacks\_Nbr **=** **0;**  *// Semaphore to indicate completion of the callback operation*  xSemaphoreHandle BMP180\_Semaphore**;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task function declarations*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** BMP180SimpleCallback**(void\*** pvData**,** **uint\_fast8\_t** ui8Status**);**  **extern** **void** Task\_BMP180\_Handler**(void\*** pvParameters**);**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task function definitions*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: BMP180SimpleCallback*  *\* Description: Callback Routine for the BMP180 for when*  *\* transactions have completed*  *\* Parameters: void\* pvData*  *\* uint\_fast8\_t ui8Status*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **void** BMP180SimpleCallback**(void\*** pvData**,** **uint\_fast8\_t** ui8Status**)** **{**  portBASE\_TYPE xHigherPriorityTaskWoken **=** pdFALSE**;**  BMP180\_Callbacks\_Nbr**++;**  **if** **(**ui8Status **!=** I2CM\_STATUS\_SUCCESS**)** **{**  *// An error occurred*  UARTprintf**(**">>>>BMP180 Error: %02X\n"**,** ui8Status**);**  **}**  *// Indicate that the I2C transaction has completed.*  BMP180SimpleDone **=** true**;**  *// "Give" the BMP180\_Semaphore*  xSemaphoreGiveFromISR**(**BMP180\_Semaphore**,** **&**xHigherPriorityTaskWoken**);**  portYIELD\_FROM\_ISR**(**xHigherPriorityTaskWoken**);**  **}**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: Task\_BMP180\_Handler*  *\* Description: Task to report Temperature and Pressure from*  *\* BMP180 sensor*  *\* Parameters: void\* pvParameters;*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** Task\_BMP180\_Handler**(void\*** pvParameters**)** **{**  *// Initialize UART*  UARTStdio\_Initialization**();**  *// Initialize I2C7*  I2C7\_Initialization**();**  *// Initialize BMP180\_Semaphore*  vSemaphoreCreateBinary**(**BMP180\_Semaphore**);**  *// Initialize the BMP180.*  BMP180SimpleDone **=** false**;**  BMP180Init**(&**sBMP180**,** I2C7\_Instance\_Ref**,** BMP180\_ADDRESS**,**  BMP180SimpleCallback**,** **0);**  xSemaphoreTake**(**BMP180\_Semaphore**,** portMAX\_DELAY**);**  UARTprintf**(**">>>>BMP180: Initialized!\n"**);**  *// Loop forever reading and reporting data from the BMP180.*  **while** **(1)** **{**  **float** fTemperature **=** **0.0;**  **float** fPressure **=** **0.0;**  *// Request a reading from the BMP180.*  BMP180DataRead**(&**sBMP180**,** BMP180SimpleCallback**,** **0);**  xSemaphoreTake**(**BMP180\_Semaphore**,** portMAX\_DELAY**);**  *// Get the new pressure and temperature reading.*  BMP180DataPressureGetFloat**(&**sBMP180**,** **&**fPressure**);**  BMP180DataTemperatureGetFloat**(&**sBMP180**,** **&**fTemperature**);**  ReportData\_Item pressureItem**;**  pressureItem**.**TimeStamp **=** xPortSysTickCount**;**  pressureItem**.**ReportName **=** **0002;**  pressureItem**.**ReportValueType\_Flg **=** **0**b0001**;**  pressureItem**.**ReportValue\_0 **=** **\*(int32\_t\*)&**fPressure**;**  pressureItem**.**ReportValue\_1 **=** **0;**  pressureItem**.**ReportValue\_2 **=** **0;**  pressureItem**.**ReportValue\_3 **=** **0;**  ReportData\_Item tempItem**;**  tempItem**.**TimeStamp **=** xPortSysTickCount**;**  tempItem**.**ReportName **=** **0003;**  tempItem**.**ReportValueType\_Flg **=** **0**b0001**;**  tempItem**.**ReportValue\_0 **=** **\*(int32\_t\*)&**fTemperature**;**  tempItem**.**ReportValue\_1 **=** **0;**  tempItem**.**ReportValue\_2 **=** **0;**  tempItem**.**ReportValue\_3 **=** **0;**  *// Send ReportData\_Items to queue to print*  xQueueSend**(**ReportData\_Queue**,** **&**pressureItem**,** **0);**  xQueueSend**(**ReportData\_Queue**,** **&**tempItem**,** **0);**  *// Delay*  vTaskDelay**((**SysTickFrequency **\*** **1000)** **/** **1000);**  **}**  **}** |