CPE403 – Advanced Embedded Systems

Design Assignment 02

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Rishawn Peppers Johnson

Email: Peppersj@unlv.nevada.edu

Github Repository link (root): https://github.com/PeppersJ/v4e0nk_i3

Youtube Playlist link (root): https://drive.google.com/drive/folders/1fJ029-AAWjTnN-QrRqNLd0iLwKGm6A08?usp=sharing

Follow the submission guideline to be awarded points for this Assignment.

Submit the following for all Assignments:

1. In the document, for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only.
2. Create a private Github repository with a random name (no CPE/403, Lastname, Firstname). Place all labs under the root folder TIVAC, sub-folder named Assignment1, with one document and one video link file for each lab, place modified c files named as asng_taskxx.c.
3. If multiple c files or other libraries are used, create a folder asng1_t01 and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup_ccs.c and other include files, c) text file with youtube video links (see template).
5. Submit the doc file in canvas before the due date. The root folder of the github assignment directory should have the documentation and the text file with youtube video links.
6. Organize your youtube videos as playlist under the name “cpe403”. The playlist should have the video sequence arranged as submission or due dates.
7. Only submit pdf documents. Do not forget to upload this document in the github repository and in the canvas submission portal.

1. Code for Tasks. for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only. Use separate page for each task.

Base Code

```
//*****************************************************************************
//
// temperature_tmp006.c - Example to use of the SensorLib with the TMP006
//
// Copyright (c) 2013-2017 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.4.178 of the EK-TM4C123GXL Firmware Package.
//
//*****************************************************************************

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "sensorlib/hw_tmp006.h"
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/tmp006.h"
#include "drivers/rgb.h"

//*****************************************************************************
//
//! \addtogroup example_list
//! <h1>Temperature Measurement with the TMP006 (temperature_tmp006)</h1>
//!
//! This example demonstrates the basic use of the Sensor Library, TM4C123G
//! LaunchPad and the SensHub BoosterPack to obtain ambient and object
//! temperature measurements with the Texas Instruments TMP006 sensor.
//!
//! Connect a serial terminal program to the LaunchPad's ICDI virtual serial
//! port at 115,200 baud.  Use eight bits per byte, no parity and one stop bit.
//! The raw sensor measurements are printed to the terminal.  The RGB LED
//! blinks at 1Hz once the initialization is complete and the example is
//! running.
//
//*****************************************************************************

//*****************************************************************************
//
// Define TMP006 I2C Address.
//
//*****************************************************************************
#define TMP006_I2C_ADDRESS      0x41

//*****************************************************************************
//
// Global instance structure for the I2C master driver.
//
//*****************************************************************************
tI2CMInstance g_sI2CInst;

//*****************************************************************************
//
// Global instance structure for the TMP006 sensor driver.
//
//*****************************************************************************
tTMP006 g_sTMP006Inst;

//*****************************************************************************
//
// Global new data flag to alert main that TMP006 data is ready.
//
//*****************************************************************************
volatile uint_fast8_t g_vui8DataFlag;

//*****************************************************************************
//
// Global new error flag to store the error condition if encountered.
//
//*****************************************************************************
volatile uint_fast8_t g_vui8ErrorFlag;

//*****************************************************************************
//
// Colors for the RGB LED.
//
//*****************************************************************************
uint32_t g_pui32Colors[3];

//*****************************************************************************
//
// Application function to capture ASSERT failures and other debug conditions.
```

```
//
//*****************************************************************************
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

//*****************************************************************************
//
// TMP006 Sensor callback function.  Called at the end of TMP006 sensor driver
// transactions. This is called from I2C interrupt context. Therefore, we just
// set a flag and let main do the bulk of the computations and display.
//
//*****************************************************************************
void
TMP006AppCallback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    //
    // If the transaction succeeded set the data flag to indicate to
    // application that this transaction is complete and data may be ready.
    //
    if(ui8Status == I2CM_STATUS_SUCCESS)
    {
        g_vui8DataFlag = 1;
    }

    //
    // Store the most recent status in case it was an error condition
    //
    g_vui8ErrorFlag = ui8Status;
}

//*****************************************************************************
//
// TMP006 Application error handler.
//
//*****************************************************************************
void
TMP006AppErrorHandler(char *pcFilename, uint_fast32_t ui32Line)
{
    //
    // Set terminal color to red and print error status and locations
    //
    UARTprintf("\033[31;1m");
    UARTprintf("Error: %d, File: %s, Line: %d\n"
               "See I2C status definitions in utils\\i2cm_drv.h\n",
               g_vui8ErrorFlag, pcFilename, ui32Line);

    //
    // Return terminal color to normal
    //
    UARTprintf("\033[0m");
    //
    // Set RGB Color to RED
    //
    g_pui32Colors[0] = 0xFFFF;
    g_pui32Colors[1] = 0;
    g_pui32Colors[2] = 0;
    RGBColorSet(g_pui32Colors);

    //
    // Increase blink rate to get attention
    //
    RGBBlinkRateSet(10.0f);

    //
    // Go to sleep wait for interventions.  A more robust application could
    // attempt corrective actions here.
    //
    while(1)
    {
        ROM_SysCtlSleep();
    }
}

//*****************************************************************************
//
// Called by the NVIC as a result of I2C3 Interrupt. I2C3 is the I2C connection
// to the TMP006.
//
//*****************************************************************************
void
TMP006I2CIntHandler(void)
{
    //
    // Pass through to the I2CM interrupt handler provided by sensor library.
    // This is required to be at application level so that I2CMIntHandler can
    // receive the instance structure pointer as an argument.
    //
    I2CMIntHandler(&g_sI2CInst);
}

//*****************************************************************************
//
// Called by the NVIC as a result of GPIO port E interrupt event. For this
// application GPIO port E pin 0 is the interrupt line for the TMP006
//
//*****************************************************************************
void
IntGPIOe(void)
{
    uint32_t ui32Status;

    ui32Status = GPIOIntStatus(GPIO_PORTE_BASE, true);

    //
    // Clear all the pin interrupts that are set
    //
    GPIOIntClear(GPIO_PORTE_BASE, ui32Status);

    if(ui32Status & GPIO_PIN_0)
    {
        //
        // This interrupt indicates a conversion is complete and ready to be
        // fetched.  So we start the process of getting the data.
        //
        TMP006DataRead(&g_sTMP006Inst, TMP006AppCallback, &g_sTMP006Inst);
    }
}

//*****************************************************************************
//
// Configure the UART and its pins.  This must be called before UARTprintf().
//
//*****************************************************************************
void
ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

//*****************************************************************************
//
// Main 'C' Language entry point.
//
//*****************************************************************************
int
```

```
main(void)
{
    float fAmbient, fObject;
    int_fast32_t i32IntegerPart;
    int_fast32_t i32FractionPart;

    //
    // Setup the system clock to run at 40 Mhz from PLL with crystal reference
    //
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
                       SYSCTL_OSC_MAIN);

    //
    // Enable the peripherals used by this example.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    //
    // Initialize the UART.
    //
    ConfigureUART();

    //
    // Print the welcome message to the terminal.
    //
    UARTprintf("\033[2J\033[1;1HTMP006 Example\n");

    //
    // Setup the color of the RGB LED.
    //
    g_pui32Colors[RED] = 0;
    g_pui32Colors[BLUE] = 0xFFFF;
    g_pui32Colors[GREEN] = 0;

    //
    // Initialize the RGB Driver and start RGB blink operation.
    //
    RGBInit(0);
    RGBColorSet(g_pui32Colors);
    RGBIntensitySet(0.5f);
    RGBEnable();

    //
    // The I2C3 peripheral must be enabled before use.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C3);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

    //
    // Configure the pin muxing for I2C3 functions on port D0 and D1.
    // This step is not necessary if your part does not support pin muxing.
    //
    ROM_GPIOPinConfigure(GPIO_PD0_I2C3SCL);
    ROM_GPIOPinConfigure(GPIO_PD1_I2C3SDA);
```

```
//
// Select the I2C function for these pins.  This function will also
// configure the GPIO pins pins for I2C operation, setting them to
// open-drain operation with weak pull-ups.  Consult the data sheet
// to see which functions are allocated per pin.
//
GPIOPinTypeI2CSCL(GPIO_PORTD_BASE, GPIO_PIN_0);
ROM_GPIOPinTypeI2C(GPIO_PORTD_BASE, GPIO_PIN_1);

//
// Configure and Enable the GPIO interrupt. Used for DRDY from the TMP006
//
ROM_GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_0);
GPIOIntEnable(GPIO_PORTE_BASE, GPIO_PIN_0);
ROM_GPIOIntTypeSet(GPIO_PORTE_BASE, GPIO_PIN_0, GPIO_FALLING_EDGE);
ROM_IntEnable(INT_GPIOE);

//
// Keep only some parts of the systems running while in sleep mode.
// GPIOE is for the TMP006 data ready interrupt.
// UART0 is the virtual serial port
// TIMER0, TIMER1 and WTIMER5 are used by the RGB driver
// I2C3 is the I2C interface to the TMP006
//
ROM_SysCtlPeripheralClockGating(true);
ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOE);
ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UART0);
ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER0);
ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER1);
ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_I2C3);
ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_WTIMER5);

//
// Enable interrupts to the processor.
//
ROM_IntMasterEnable();

//
// Initialize I2C3 peripheral.
//
I2CMInit(&g_sI2CInst, I2C3_BASE, INT_I2C3, 0xff, 0xff,
         SysCtlClockGet());

//
// Initialize the TMP006
//
TMP006Init(&g_sTMP006Inst, &g_sI2CInst, TMP006_I2C_ADDRESS,
           TMP006AppCallback, &g_sTMP006Inst);

//
// Put the processor to sleep while we wait for the I2C driver to
// indicate that the transaction is complete.
//
while((g_vui8DataFlag == 0) && (g_vui8ErrorFlag == 0))
{
        ROM_SysCtlSleep();
    }

    //
    // If an error occurred call the error handler immediately.
    //
    if(g_vui8ErrorFlag)
    {
        TMP006AppErrorHandler(__FILE__, __LINE__);
    }

    //
    // clear the data flag for next use.
    //
    g_vui8DataFlag = 0;

    //
    // Delay for 10 milliseconds for TMP006 reset to complete.
    // Not explicitly required. Datasheet does not say how long a reset takes.
    //
    ROM_SysCtlDelay(ROM_SysCtlClockGet() / (100 * 3));

    //
    // Enable the DRDY pin indication that a conversion is in progress.
    //
    TMP006ReadModifyWrite(&g_sTMP006Inst, TMP006_O_CONFIG,
                          ~TMP006_CONFIG_EN_DRDY_PIN_M,
                          TMP006_CONFIG_EN_DRDY_PIN, TMP006AppCallback,
                          &g_sTMP006Inst);

    //
    // Wait for the DRDY enable I2C transaction to complete.
    //
    while((g_vui8DataFlag == 0) && (g_vui8ErrorFlag == 0))
    {
        ROM_SysCtlSleep();
    }

    //
    // If an error occurred call the error handler immediately.
    //
    if(g_vui8ErrorFlag)
    {
        TMP006AppErrorHandler(__FILE__, __LINE__);
    }

    //
    // clear the data flag for next use.
    //
    g_vui8DataFlag = 0;

    //
    // Last thing before the loop start blinking to show we got this far and
    // the tmp006 is setup and ready for auto measure
    //
```

```
RGBBlinkRateSet(1.0f);

//
// Loop Forever
//
while(1)
{
    //
    // Put the processor to sleep while we wait for the TMP006 to
    // signal that data is ready.  Also continue to sleep while I2C
    // transactions get the raw data from the TMP006
    //
    while((g_vui8DataFlag == 0) && (g_vui8ErrorFlag == 0))
    {
        ROM_SysCtlSleep();
    }

    //
    // If an error occurred call the error handler immediately.
    //
    if(g_vui8ErrorFlag)
    {
        TMP006AppErrorHandler(__FILE__, __LINE__);
    }

    //
    // Reset the flag
    //
    g_vui8DataFlag = 0;

    //
    // Get a local copy of the latest data in float format.
    //
    TMP006DataTemperatureGetFloat(&g_sTMP006Inst, &fAmbient, &fObject);

    //
    // Convert the floating point ambient temperature  to an integer part
    // and fraction part for easy printing.
    //
    i32IntegerPart = (int32_t)fAmbient;
    i32FractionPart = (int32_t)(fAmbient * 1000.0f);
    i32FractionPart = i32FractionPart - (i32IntegerPart * 1000);
    if(i32FractionPart < 0)
    {
        i32FractionPart *= -1;
    }
    UARTprintf("Ambient %3d.%03d\t", i32IntegerPart, i32FractionPart);

    //
    // Convert the floating point ambient temperature  to an integer part
    // and fraction part for easy printing.
    //
    i32IntegerPart = (int32_t)fObject;
    i32FractionPart = (int32_t)(fObject * 1000.0f);
    i32FractionPart = i32FractionPart - (i32IntegerPart * 1000);
```

```
        if(i32FractionPart < 0)
        {
            i32FractionPart *= -1;
        }
        UARTprintf("Object %3d.%03d\n", i32IntegerPart, i32FractionPart);
    }
}
```

Modified Code

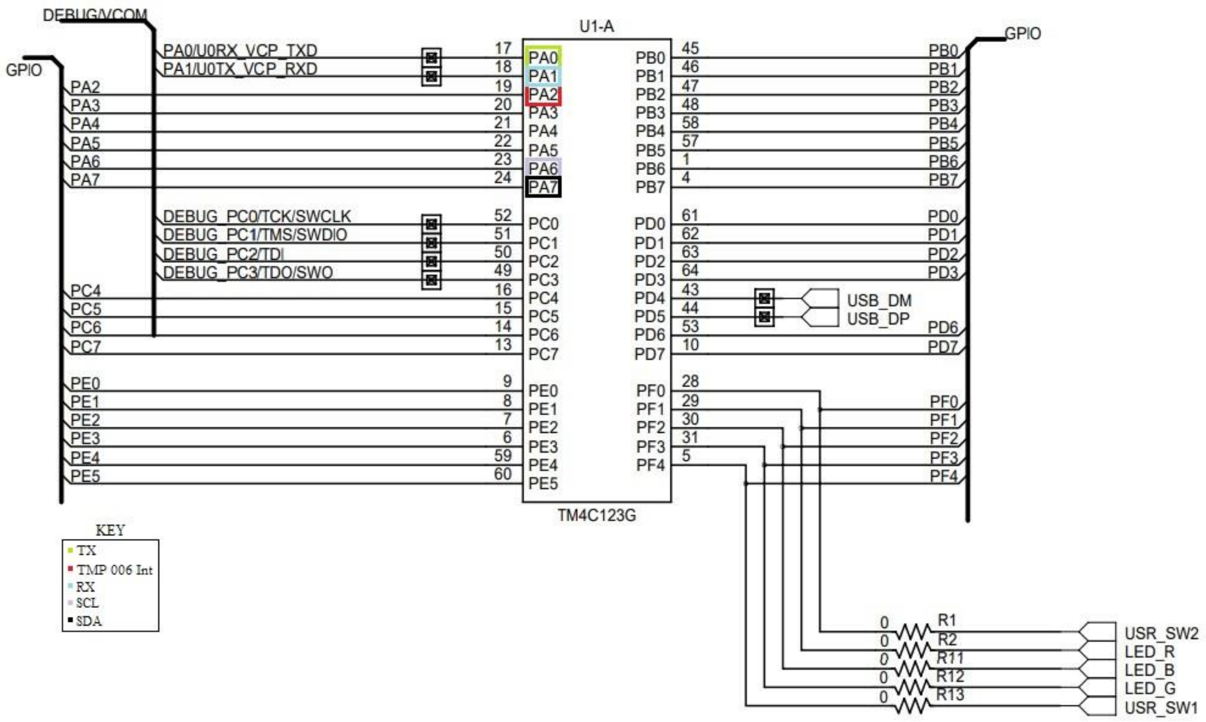
```
/*  Modified By: Rishawn Peppers Johnson
 *  Date Created: 20 October 2020
 *  Device: TivaC123GH6PM
 *  CpE 403 Assignment 02
 *
 *  Purpose: Interface the TivaC123GH6PM with the Educational BoosterPack MKII to
 *      read the ambient and object temperature from the MKII's tmp006.
 *
 *  Inputs: tmp006 - Ambient and object temperature.
 *  Outputs: Ambient and Object temperature through UART.
 *
 *  */
//*****************************************************************************
//
// temperature_tmp006.c - Example to use of the SensorLib with the TMP006
//
// Copyright (c) 2013-2017 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.4.178 of the EK-TM4C123GXL Firmware Package.
//
//*****************************************************************************

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "sensorlib/hw_tmp006.h"
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/tmp006.h"
#define GLOBAL_Q    20
#include"IQmath/IQmathLib.h"
```

```
//*****************************************************************************
//
//! \addtogroup example_list
//! <h1>Temperature Measurement with the TMP006 (temperature_tmp006)</h1>
//!
//! This example demonstrates the basic use of the Sensor Library, TM4C123G
//! LaunchPad and the SensHub BoosterPack to obtain ambient and object
//! temperature measurements with the Texas Instruments TMP006 sensor.
//!
//! Connect a serial terminal program to the LaunchPad's ICDI virtual serial
//! port at 115,200 baud.  Use eight bits per byte, no parity and one stop bit.
//! The raw sensor measurements are printed to the terminal.  The RGB LED
//! blinks at 1Hz once the initialization is complete and the example is
//! running.
//
//*****************************************************************************

//*****************************************************************************
//
// Define TMP006 I2C Address.
//
//*****************************************************************************
#define TMP006_I2C_ADDRESS      0x40

//*****************************************************************************
//
// Global instance structure for the I2C master driver.
//
//*****************************************************************************
tI2CMInstance g_sI2CInst;

//*****************************************************************************
//
// Global instance structure for the TMP006 sensor driver.
//
//*****************************************************************************
tTMP006 g_sTMP006Inst;

//*****************************************************************************
//
// Global new data flag to alert main that TMP006 data is ready.
//
//*****************************************************************************
volatile uint_fast8_t g_vui8DataFlag;

//*****************************************************************************
//
// Global new error flag to store the error condition if encountered.
//
//*****************************************************************************
volatile uint_fast8_t g_vui8ErrorFlag;

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

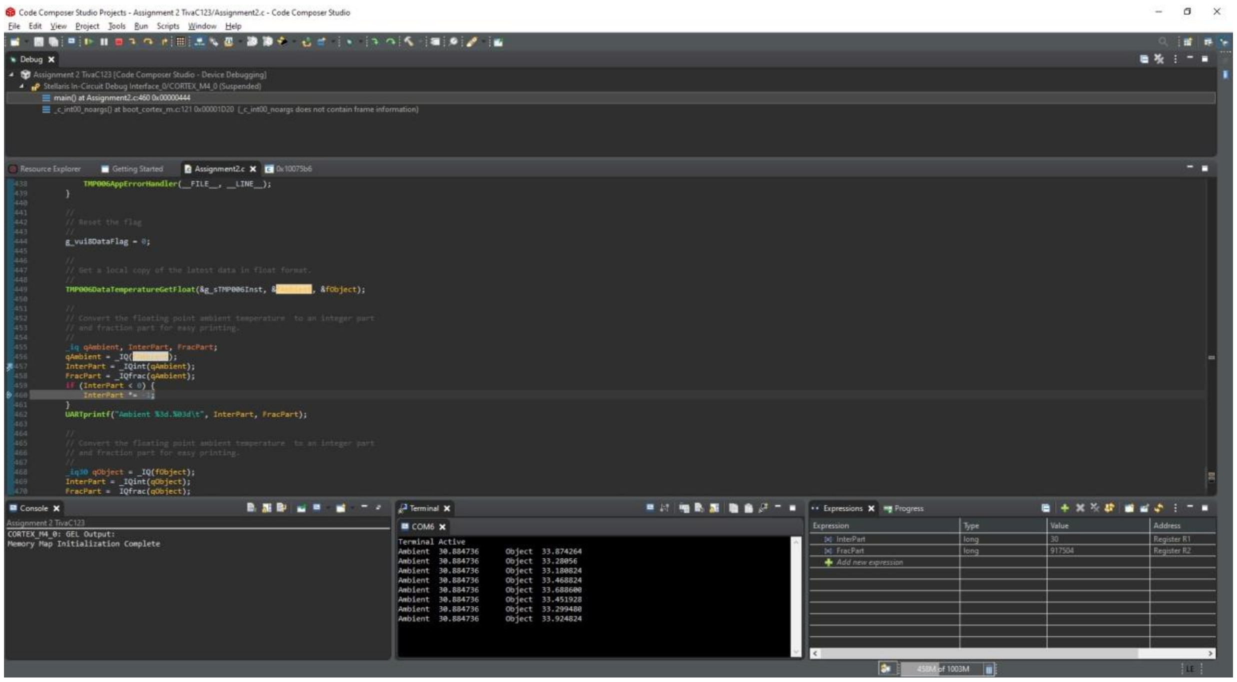
```
//*****************************************************************************
//
// Application function to capture ASSERT failures and other debug conditions.
//
//*****************************************************************************
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

//*****************************************************************************
//
// TMP006 Sensor callback function.  Called at the end of TMP006 sensor driver
// transactions. This is called from I2C interrupt context. Therefore, we just
// set a flag and let main do the bulk of the computations and display.
//
//*****************************************************************************
void
TMP006AppCallback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    //
    // If the transaction succeeded set the data flag to indicate to
    // application that this transaction is complete and data may be ready.
    //
    if(ui8Status == I2CM_STATUS_SUCCESS)
    {
        g_vui8DataFlag = 1;
    }

    //
    // Store the most recent status in case it was an error condition
    //
    g_vui8ErrorFlag = ui8Status;
}

//*****************************************************************************
//
// TMP006 Application error handler.
//
//*****************************************************************************
void
TMP006AppErrorHandler(char *pcFilename, uint_fast32_t ui32Line)
{
    //
    // Set terminal color to red and print error status and locations
    //
    UARTprintf("\033[31;1m");
    UARTprintf("Error: %d, File: %s, Line: %d\n"
               "See I2C status definitions in utils\\i2cm_drv.h\n",
               g_vui8ErrorFlag, pcFilename, ui32Line);

    //
    // Return terminal color to normal
    //
    UARTprintf("\033[0m");

    //
    // Go to sleep wait for interventions.  A more robust application could
    // attempt corrective actions here.
    //
    while(1)
    {
        ROM_SysCtlSleep();
    }
}

//*****************************************************************************
//
// Called by the NVIC as a result of I2C3 Interrupt. I2C3 is the I2C connection
// to the TMP006.
//
//*****************************************************************************
void
TMP006I2CIntHandler(void)
{
    //
    // Pass through to the I2CM interrupt handler provided by sensor library.
    // This is required to be at application level so that I2CMIntHandler can
    // receive the instance structure pointer as an argument.
    //
    I2CMIntHandler(&g_sI2CInst);
}

//*****************************************************************************
//
// Called by the NVIC as a result of GPIO port E interrupt event. For this
// application GPIO port E pin 0 is the interrupt line for the TMP006
//
//*****************************************************************************
void
IntGPIOa(void)
{
    uint32_t ui32Status;

    ui32Status = GPIOIntStatus(GPIO_PORTA_BASE, true);

    //
    // Clear all the pin interrupts that are set
    //
    GPIOIntClear(GPIO_PORTA_BASE, ui32Status);

    if(ui32Status & GPIO_PIN_2)
    {
        //
        // This interrupt indicates a conversion is complete and ready to be
        // fetched.  So we start the process of getting the data.
        //
        TMP006DataRead(&g_sTMP006Inst, TMP006AppCallback, &g_sTMP006Inst);
```

```c
    }
}

//*****************************************************************************
//
// Configure the UART and its pins.  This must be called before UARTprintf().
//
//*****************************************************************************
void
ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

//*****************************************************************************
//
// Main 'C' Language entry point.
//
//*****************************************************************************
int
main(void)
{
    float fAmbient, fObject;
    int_fast32_t i32IntegerPart;
    int_fast32_t i32FractionPart;

    //
    // Setup the system clock to run at 40 Mhz from PLL with crystal reference
    //
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);

    //
    // Enable the peripherals used by this example.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Initialize the UART.
    //
    ConfigureUART();

    //
    // Print the welcome message to the terminal.
    //
    UARTprintf("Terminal Active\n");

    //
    // The I2C3 peripheral must be enabled before use.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Configure the pin muxing for I2C3 functions on port D0 and D1.
    // This step is not necessary if your part does not support pin muxing.
    //
    ROM_GPIOPinConfigure(GPIO_PA6_I2C1SCL);
    ROM_GPIOPinConfigure(GPIO_PA7_I2C1SDA);

    //
    // Select the I2C function for these pins.  This function will also
    // configure the GPIO pins pins for I2C operation, setting them to
    // open-drain operation with weak pull-ups.  Consult the data sheet
    // to see which functions are allocated per pin.
    //
    GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);
    ROM_GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);

    //
    // Configure and Enable the GPIO interrupt. Used for DRDY from the TMP006
    //
    ROM_GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_2);
    GPIOIntEnable(GPIO_PORTA_BASE, GPIO_PIN_2);
    ROM_GPIOIntTypeSet(GPIO_PORTA_BASE, GPIO_PIN_2, GPIO_FALLING_EDGE);
    ROM_IntEnable(INT_GPIOA);

    //
    // Keep only some parts of the systems running while in sleep mode.
    // GPIOE is for the TMP006 data ready interrupt.
    // UART0 is the virtual serial port
    // TIMER0, TIMER1 and WTIMER5 are used by the RGB driver
    // I2C3 is the I2C interface to the TMP006
    //
    ROM_SysCtlPeripheralClockGating(true);
    ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UART0);
    ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_I2C1);

    //
    // Enable interrupts to the processor.
    //
    ROM_IntMasterEnable();

    //
    // Initialize I2C1 peripheral.
    //
    I2CMInit(&g_sI2CInst, I2C1_BASE, INT_I2C1, 0xff, 0xff,
             SysCtlClockGet());

    //
    // Initialize the TMP006
    //
    volatile uint_fast8_t temp;
    temp = TMP006Init(&g_sTMP006Inst, &g_sI2CInst, TMP006_I2C_ADDRESS,
               TMP006AppCallback, &g_sTMP006Inst);
    //
    // Put the processor to sleep while we wait for the I2C driver to
    // indicate that the transaction is complete.
    //
    while((g_vui8DataFlag == 0) && (g_vui8ErrorFlag == 0))
    {
        ROM_SysCtlSleep();
    }

    //
    // If an error occurred call the error handler immediately.
    //
    if(g_vui8ErrorFlag)
    {
        TMP006AppErrorHandler(__FILE__, __LINE__);
    }

    //
    // clear the data flag for next use.
    //
    g_vui8DataFlag = 0;

    //
    // Delay for 10 milliseconds for TMP006 reset to complete.
    // Not explicitly required. Datasheet does not say how long a reset takes.
    //
    ROM_SysCtlDelay(ROM_SysCtlClockGet() / (100 * 3));

    //
    // Enable the DRDY pin indication that a conversion is in progress.
    //
    TMP006ReadModifyWrite(&g_sTMP006Inst, TMP006_O_CONFIG,
                          ~TMP006_CONFIG_EN_DRDY_PIN_M,
                      TMP006_CONFIG_EN_DRDY_PIN, TMP006AppCallback,
                      &g_sTMP006Inst);

    //
    // Wait for the DRDY enable I2C transaction to complete.
    //
    while((g_vui8DataFlag == 0) && (g_vui8ErrorFlag == 0))
    {
        ROM_SysCtlSleep();
    }

    //
    // If an error occurred call the error handler immediately.
    //
    if(g_vui8ErrorFlag)
    {
        TMP006AppErrorHandler(__FILE__, __LINE__);
    }

    //
    // clear the data flag for next use.
    //
    g_vui8DataFlag = 0;

    //
    // Last thing before the loop start blinking to show we got this far and
    // the tmp006 is setup and ready for auto measure
    //
    //RGBBlinkRateSet(1.0f);

    //
    // Loop Forever
    //
    while(1)
    {
        //
        // Put the processor to sleep while we wait for the TMP006 to
        // signal that data is ready.  Also continue to sleep while I2C
        // transactions get the raw data from the TMP006
        //
        while((g_vui8DataFlag == 0) && (g_vui8ErrorFlag == 0))
        {
            ROM_SysCtlSleep();
        }

        //
        // If an error occurred call the error handler immediately.
        //
        if(g_vui8ErrorFlag)
        {
            TMP006AppErrorHandler(__FILE__, __LINE__);
        }

        //
        // Reset the flag
        //
        g_vui8DataFlag = 0;

        //
        // Get a local copy of the latest data in float format.
        //
        TMP006DataTemperatureGetFloat(&g_sTMP006Inst, &fAmbient, &fObject);

        //
        // Convert the floating point ambient temperature  to an integer part
        // and fraction part for easy printing.
        //
        _iq qAmbient, InterPart, FracPart;
        qAmbient = _IQ(fAmbient);
        InterPart = _IQint(qAmbient);
        FracPart = _IQfrac(qAmbient);
        if (InterPart < 0) {
            InterPart *= -1;
        }
        UARTprintf("Ambient %3d.%03d\t", InterPart, FracPart);

        //
        // Convert the floating point ambient temperature  to an integer part
        // and fraction part for easy printing.
        //
        _iq30 qObject = _IQ(fObject);
        InterPart = _IQint(qObject);
        FracPart = _IQfrac(qObject);
        if (InterPart < 0) {
            InterPart *= -1;
        }
        UARTprintf("Object %3d.%03d\n", InterPart, FracPart);



    }
}
```

2. Block diagram and/or Schematics showing the components, pins used, and interface.



3. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.


Code Composer Studio Projects - Assignment 2 TivaC123/Assignment2.c - Code Composer Studio
File Edit View Project Tools Run Scripts Window Help
Debug
Resource Explorer
Getting Started
Assignment2.c
0x10075b6
Console
Terminal
COM6
Expressions
Progress
458M of 1003M

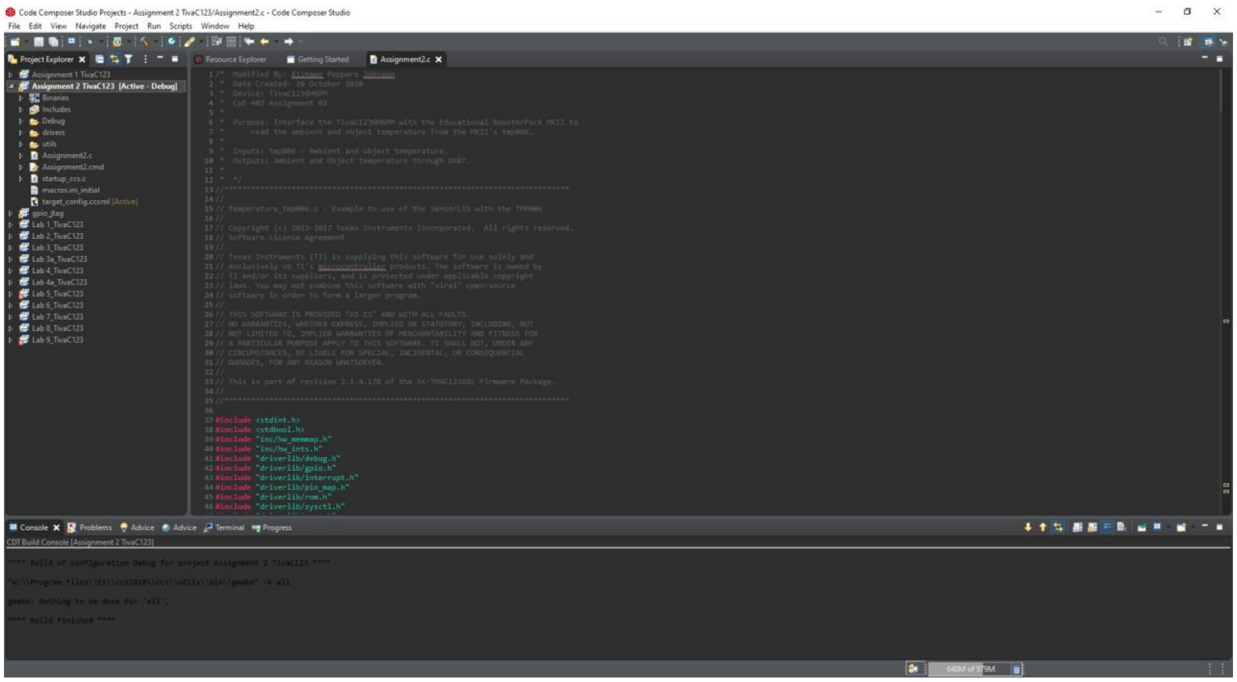
```
Assignment 2 TivaC123 [Code Composer Studio - Device Debugging]
  Stellaris In-Circuit Debug Interface_0/CORTEX_M4_0 (Suspended)
    main() at Assignment2.c:460 0x00000444
    _c_int00_noargs() at boot_cortex_m.c:121 0x00001D20  (_c_int00_noargs does not contain frame information)
```

```c
            TMP006AppErrorHandler(__FILE__, __LINE__);
        }

        //
        // Reset the flag
        //
        g_vui8DataFlag = 0;

        //
        // Get a local copy of the latest data in float format.
        //
        TMP006DataTemperatureGetFloat(&g_sTMP006Inst, &fAmbient, &fObject);

        //
        // Convert the floating point ambient temperature  to an integer part
        // and fraction part for easy printing.
        //
        _iq qAmbient, InterPart, FracPart;
        qAmbient = _IQ(fAmbient);
        InterPart = _IQint(qAmbient);
        FracPart = _IQfrac(qAmbient);
        if (InterPart < 0) {
            InterPart *= -1;
        }
        UARTprintf("Ambient %3d.%03d\t", InterPart, FracPart);

        //
        // Convert the floating point ambient temperature  to an integer part
        // and fraction part for easy printing.
        //
        _iq30 qObject = _IQ(fObject);
        InterPart = _IQint(qObject);
        FracPart =  IQfrac(qObject);
```

```
Assignment 2 TivaC123
CORTEX_M4_0: GEL Output:
Memory Map Initialization Complete
```

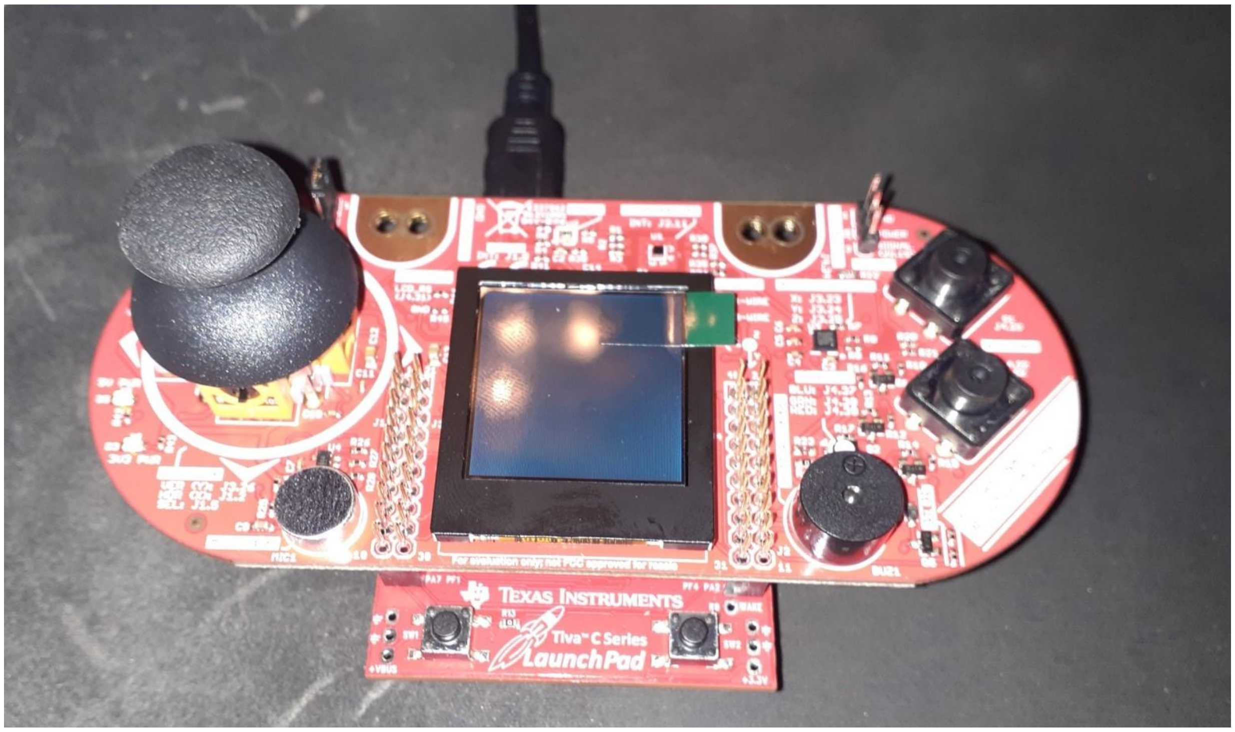
```
Terminal Active
Ambient  30.884736      Object  33.874264
Ambient  30.884736      Object  33.28056
Ambient  30.884736      Object  33.180824
Ambient  30.884736      Object  33.468824
Ambient  30.884736      Object  33.688600
Ambient  30.884736      Object  33.451928
Ambient  30.884736      Object  33.299480
Ambient  30.884736      Object  33.924824
```

Expression	Type	Value	Address
InterPart	long	30	Register R1
FracPart	long	917504	Register R2
Add new expression			



```c
/*  Modified By: [illegible] Peppers Johnson
 *  Date Created: 20 October 2020
 *  Device: TivaC123GH6PM
 *  CpE 403 Assignment 02
 *
 *  Purpose: Interface the TivaC123GH6PM with the Educational BoosterPack MKII to
 *       read the ambient and object temperature from the MKII's tmp006.
 *
 *  Inputs: tmp006 - Ambient and object temperature.
 *  Outputs: Ambient and Object temperature through UART.
 *
 *  */
//*****************************************************************************
//
// temperature_tmp006.c - Example to use of the SensorLib with the TMP006
//
// Copyright (c) 2013-2017 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.4.178 of the EK-TM4C123GXL Firmware Package.
//
//*****************************************************************************

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
```

```
CDT Build Console [Assignment 2 TivaC123]

**** Build of configuration Debug for project Assignment 2 TivaC123 ****

"W:\\Program Files\\ti\\ccs1010\\ccs\\utils\\bin\\gmake" -k all

gmake: Nothing to be done for 'all'.

**** Build Finished ****
```


Texas Instruments
Tiva C Series
LaunchPad

4. Declaration
   I understand the Student Academic Misconduct Policy -
   http://studentconduct.unlv.edu/misconduct/policy.html

“This assignment submission is my own, original work”.
Rishawn Peppers Johnson