CPE403 – Advanced Embedded Systems

Design Assignment 3

Name: Elmer Mejia

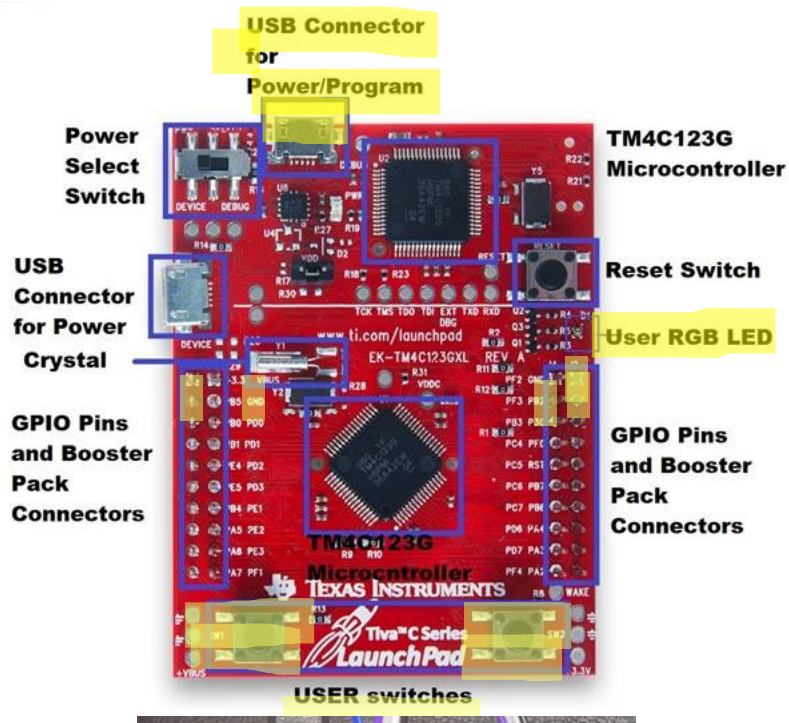
Email: mejiae4@unlv.nevada.edu

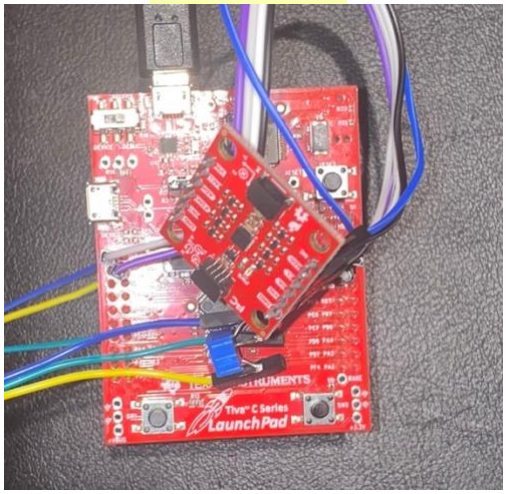
Github Repository link (root): assignments

Youtube Playlist link (root): Tiva_C

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

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



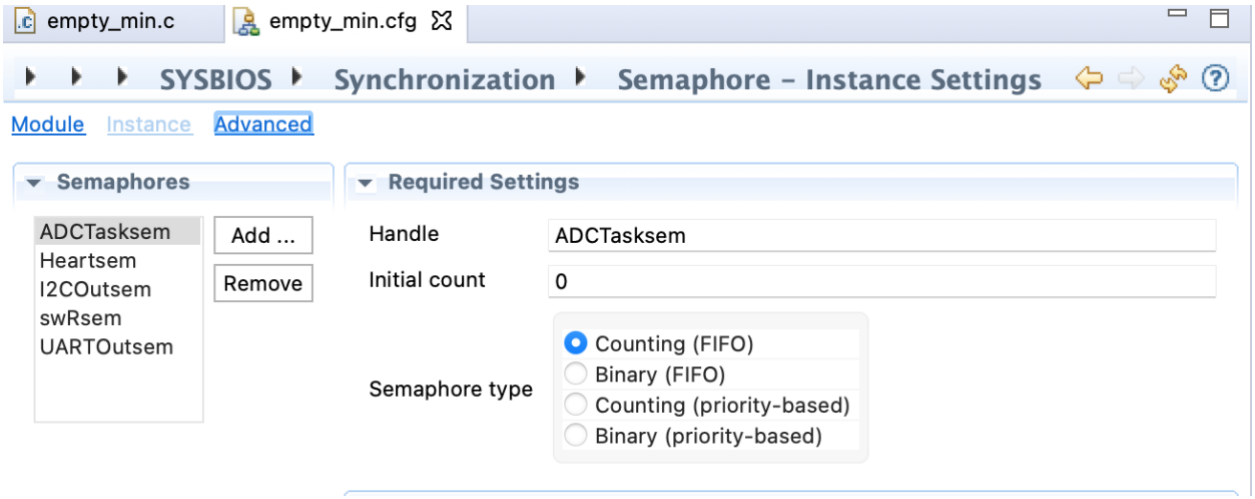


Idle and Tasks:

empty_min.c | empty_min.cfg

```
driversConfig.libType = driversConfig.LibType_Instrumented;
var semaphore3Params = new Semaphore.Params();
semaphore3Params.instance.name = "Heartsem";
Program.global.Heartsem = Semaphore.create(null, semaphore3Params);
var halHwi0Params = new halHwi.Params();
halHwi0Params.instance.name = "Timer_2A_INT";
Program.global.Timer_2A_INT = halHwi.create(39, "&Timer_ISR", halHwi0Params);
BIOS.rtsGateType = BIOS.GateMutex;
LoggingSetup.sysbiosHwiLogging = true;
LoggingSetup.loadHwiLogging = true;
var semaphore2Params = new Semaphore.Params();
semaphore2Params.instance.name = "ADCTasksem";
Program.global.ADCTasksem = Semaphore.create(null, semaphore2Params);
var task1Params = new Task.Params();
task1Params.instance.name = "ADCTask";
task1Params.stackSize = 2048;
Program.global.ADCTask = Task.create("&ADCtask", task1Params);
var semaphore2Params0 = new Semaphore.Params();
semaphore2Params0.instance.name = "UARTOutsem";
Program.global.UARTOutsem = Semaphore.create(null, semaphore2Params0);
var task2Params = new Task.Params();
task2Params.instance.name = "UARTTask";
task2Params.stackSize = 2048;
Program.global.UARTTask = Task.create("&UART_Out", task2Params);
var semaphore3Params0 = new Semaphore.Params();
semaphore3Params0.instance.name = "swRsem";
Program.global.swRsem = Semaphore.create(null, semaphore3Params0);
var task3Params = new Task.Params();
task3Params.instance.name = "switchReadTask";
Program.global.switchReadTask = Task.create("&switchRead", task3Params);
var semaphore4Params = new Semaphore.Params();
semaphore4Params.instance.name = "I2COutsem";
Program.global.I2COutsem = Semaphore.create(null, semaphore4Params);
var task4Params = new Task.Params();
task4Params.instance.name = "I2CTask";
Program.global.I2CTask = Task.create("&I2C_Out", task4Params);
Idle.idleFxns[0] = "&HeartBeatFx";
BIOS.swiEnabled = true;
```

Semaphores:



2. 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.

```
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include <xdc/runtime/Log.h>              //needed for any Log_info() call
#include <xdc/cfg/global.h>               //header file for statically defined objects/handles
#include <xdc/runtime/Diags.h>

/* BIOS module Headers */
#include <ti/sysbios/BIOS.h>
```

```
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

/* Board Header file */
#include "Board.h"

/* Include header files for adc and GPIO functions */
#include <ti/drivers/GPIO.h>
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/debug.h"
#include "driverlib/pin_map.h"
#include "driverlib/adc.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/i2c.h"
#include <time.h>
#include <math.h>
```

```
#include <inc/hw_gpio.h>
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include "icm20948_def.h"

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 16384.0
#define SAMPLE_RATE 0.01
#define RATIO (180/3.14159265359)

#define PWM_FREQUENCY 55

volatile int16_t i16ToggleCount1 = 0;
volatile int16_t i16ToggleCount2 = 0;
volatile uint32_t ui32Load = 0;
volatile uint32_t ui32PWMClock = 0;
volatile uint32_t pinVal1 = 0;  // variable to hold the pinRead
volatile uint32_t pinVal2 = 0;  // variable to hold the pinRead
volatile uint32_t PWMval = 0;
volatile int16_t i16ToggleCount = 0;
uint8_t HByte , LByte=0;
uint32_t val[1];
uint32_t outVal;
//---------------------------------------------------------------------------

float ACC_Data, ACC_Data2, ACC_Data3;
```

```
float GYRO_Data, GYRO_Data2, GYRO_Data3;          // raw values
float Ax, Ay, Az;

void initI2C0(void)
{
    // Turn on I2C0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
    SysCtlDelay(3);
    // Reset I2C0
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
    SysCtlDelay(3);
    // Enable GPIOB
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlDelay(3);

    // Configure GPIO SCL/SDA pins on PB2/PB3
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Set pins to I2C function
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    // Enable and master I2C
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);

    // Clear I2C FIFOs
```

```
    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
}



void I2C0Read(uint8_t slave_addr, uint8_t reg, uint8_t *data)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
    while(I2CMasterBusy(I2C0_BASE));
    *data = I2CMasterDataGet(I2C0_BASE);
}

// This function has not been tested - for using 16bit read you can
// also use the I2C0Read twice if this does not work
void I2C0Read16(uint8_t slave_addr, uint8_t reg, uint16_t *data)
{
    uint8_t HByte , LByte=0;
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);
```

```
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
    while(I2CMasterBusy(I2C0_BASE));
    HByte = I2CMasterDataGet(I2C0_BASE);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT);
    while(I2CMasterBusy(I2C0_BASE));
    LByte = I2CMasterDataGet(I2C0_BASE);
    *data = (LByte <<8 | HByte);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
    while(I2CMasterBusy(I2C0_BASE));
}


void I2C0Write(uint8_t slave_addr, uint8_t reg, uint8_t data)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterDataPut(I2C0_BASE, data);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
    while(I2CMasterBusy(I2C0_BASE));
}

/*reads the slave device*/
void ICM_get_whom_am_I()
{
    uint8_t WAI=0;
```

```
    I2C0Write(ICM20948_ADDRESS, ICM20948_REG_PWR_MGMT_1,
ICM20948_REG_LP_CONFIG);
    SysCtlDelay(3);
    I2C0Write(ICM20948_ADDRESS, ICM20948_REG_BANK_SEL,
ICM20948_BANK_0);
    SysCtlDelay(3);
    I2C0Read(ICM20948_ADDRESS, ICM20948_REG_WHO_AM_I, &WAI);
    if (WAI != ICM20948_DEVICE_ID)
    UARTprintf("Device Not Found\n");
    else
    UARTprintf("Device Found\n");
}

void ICM20948_config(void)
{
    I2C0Write(ICM20948_ADDRESS, ICM20948_REG_PWR_MGMT_1,
ICM20948_REG_LP_CONFIG); // power on
    SysCtlDelay(3);
    I2C0Write(ICM20948_ADDRESS, ICM20948_REG_BANK_SEL,
ICM20948_BANK_2); // Bank 2 select
    SysCtlDelay(3);
    I2C0Write(ICM20948_ADDRESS, ICM20948_REG_GYRO_CONFIG_1, 0x00); // gyro
config
    SysCtlDelay(3);
    I2C0Write(ICM20948_ADDRESS, ICM20948_SHIFT_GYRO_FS_SEL, 0x00); // gyro
config
    SysCtlDelay(3);
```

```
    I2C0Write(ICM20948_ADDRESS, ICM20948_REG_ACCEL_CONFIG, 0x00); // accel
config
    SysCtlDelay(3);
    I2C0Write(ICM20948_ADDRESS, ICM20948_ACCEL_FULLSCALE_4G, 0x00); //
accel config
    SysCtlDelay(3);
    I2C0Write(ICM20948_ADDRESS, ICM20948_REG_BANK_SEL,
ICM20948_BANK_0); // Bank 2 select
    SysCtlDelay(3);
}

void delay_simple(void)
{
     SysCtlDelay(6700000);      // creates ~500ms delay - TivaWare fxn

}

void hardware_init(void)
{
    uint32_t ui32Period;

    //Set CPU Clock to 40MHz. 400MHz PLL/2 = 200 DIV 5 = 40MHz

SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYS
CTL_OSC_MAIN);

    // ADD Tiva-C GPIO setup - enables port, sets pins 1-3 (RGB) pins for output
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;   // unlock
the GPIOCR register for port F
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = 0x01;      // Free up pin 0

    // PF4 as input. Connect PF0/PF4 to internal Pull-up resistors and set 2 mA as
current strength.
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0); // make F0 an input
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);

    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

    // PWM setup
    SysCtlPWMClockSet(SYSCTL_PWMDIV_64);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);

    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
    GPIOPinConfigure(GPIO_PF1_M1PWM5);

    ui32PWMClock = SysCtlClockGet() / 64;
```

```
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
    PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, ui32Load);

    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, PWMval * ui32Load / 1000);
    PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
    PWMGenEnable(PWM1_BASE, PWM_GEN_2);


    // Timer 2 setup code
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);          // enable Timer 2 periph
clks
    TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);       // cfg Timer 2 mode -
periodic

    ui32Period = (SysCtlClockGet() /20);                   // period = CPU clk div 2 (500ms)
    TimerLoadSet(TIMER2_BASE, TIMER_A, ui32Period);        // set Timer 2 period

    TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);       // enables Timer 2 to
interrupt CPU

    TimerEnable(TIMER2_BASE, TIMER_A);                     // enable Timer 2

}
void ADC_init(void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlDelay(3);
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlDelay(3);

    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0,
ADC_CTL_CH4|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntClear(ADC0_BASE, 3);
}

void config_UART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTStdioConfig(0, 115200, 16000000);

}


void ADCtask(void){
```

```
    while(1)
    {
    Semaphore_pend(ADCTasksem, BIOS_WAIT_FOREVER);

    ADCProcessorTrigger(ADC0_BASE, 3);
    while(!ADCIntStatus(ADC0_BASE, 3, false)){}
    ADCIntClear(ADC0_BASE, 3);
    ADCSequenceDataGet(ADC0_BASE, 3, val);
    outVal = val[0];
    }
}

void UART_Out(void){
    while (1){
        Semaphore_pend (UARTOutsem, BIOS_WAIT_FOREVER);
        UARTprintf("ADC Value: %d\n\n", outVal);
        UARTprintf("Acc. X: %d | Acc. Y: %d | Acc. Z: %d\n", (int)Ax, (int)Ay, (int)Az);
    }
}

void switchRead(void)
{
    while(1)
    {
    Semaphore_pend(swRsem, BIOS_WAIT_FOREVER);
    pinVal1= GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_0);    // read F0
    pinVal2= GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)/16;    // read F4
```

```
    if(!(pinVal1&pinVal2))
    {
        PWMval = (outVal/32);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, PWMval * ui32Load / 1000);
        if (PWMval < 1)
        {
            PWMval = 1;
            PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, PWMval * ui32Load / 1000);
        }
        else if (PWMval > 120)
        {
            PWMval = 120;
            PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, PWMval * ui32Load / 1000);
        }
    }


    delay_simple();                         // create a delay of ~1/2sec
    }
}

void I2C_Out(void){
    while (1){
        Semaphore_pend (I2COutsem, BIOS_WAIT_FOREVER);
```

```
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_ACCEL_XOUT_H_SH,
&HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_ACCEL_XOUT_L_SH,
&LByte);
        ACC_Data = (LByte <<8 | HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_ACCEL_YOUT_H_SH,
&HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_ACCEL_YOUT_L_SH,
&LByte);
        ACC_Data2 = (LByte <<8 | HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_ACCEL_ZOUT_H_SH,
&HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_ACCEL_ZOUT_L_SH,
&LByte);
        ACC_Data3 = (LByte <<8 | HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_GYRO_XOUT_H_SH,
&HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_GYRO_XOUT_L_SH, &LByte);
        GYRO_Data = (LByte <<8 | HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_GYRO_YOUT_H_SH,
&HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_GYRO_YOUT_L_SH, &LByte);
        GYRO_Data2 = (LByte <<8 | HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_GYRO_ZOUT_H_SH,
&HByte);
        I2C0Read(ICM20948_ADDRESS, ICM20948_REG_GYRO_ZOUT_L_SH, &LByte);
        GYRO_Data3 = (LByte <<8 | HByte);
```

```
        Ax = (atan2(ACC_Data, sqrt (ACC_Data2 * ACC_Data2 + ACC_Data3 *
ACC_Data3))*180.0)/3.14;
        Ay = (atan2(ACC_Data2, sqrt (ACC_Data * ACC_Data + ACC_Data3 *
ACC_Data3))*180.0)/3.14;
        Az = (atan2(ACC_Data3, sqrt (ACC_Data2 * ACC_Data2 + ACC_Data3 *
ACC_Data3))*180.0)/3.14;
    }
}



void HeartBeatFx(void)
{
    while(1)
    {
        // LED values - 2=RED, 4=BLUE, 8=GREEN
        Semaphore_pend (Heartsem, BIOS_WAIT_FOREVER);
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_1))
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3,
0);
        }
        else
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 8);
```

```
        }

        delay_simple();                         // create a delay of ~1/2sec

        i16ToggleCount += 1;                    // keep track of #toggles

        Log_info1("LED TOGGLED [%u] times", i16ToggleCount);    // send #toggles to
Log Display
        System_printf("Count: %d\n", i16ToggleCount);
        System_flush();
    }
}
```

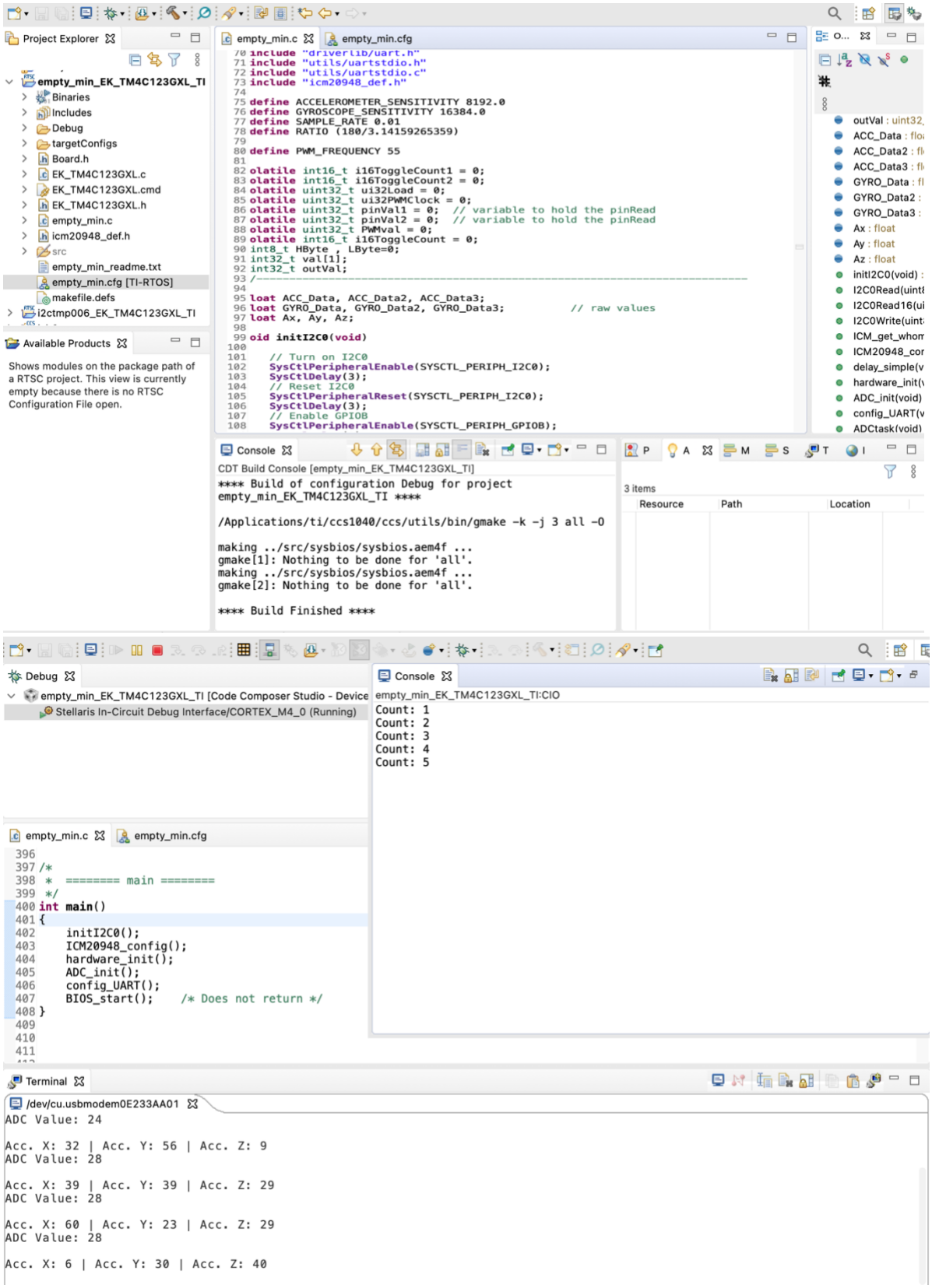
```
volatile int16_t tCount=0;


/*
 *  ======== main ========
 */
int main()
{
    initI2C0();
    ICM20948_config();
    hardware_init();
    ADC_init();
    config_UART();
    BIOS_start();    /* Does not return */
}
```

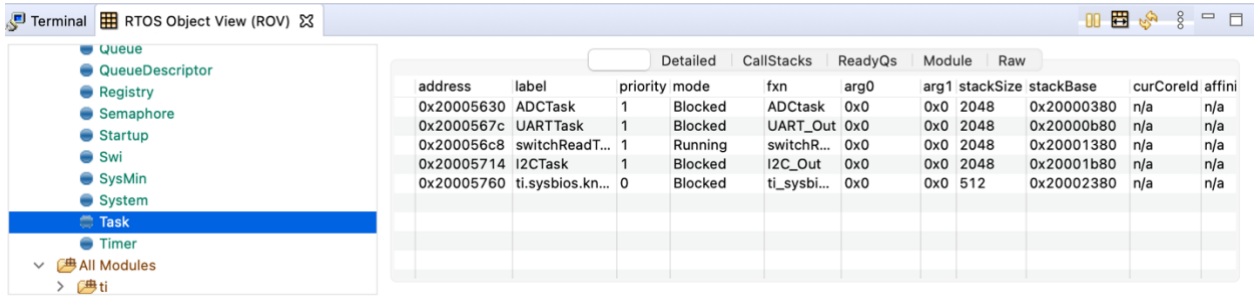
```
//---------------------------------------------------------------------------
// Timer ISR  to be called by BIOS Hwi
//
// Posts Semaphore for releasing tasks
//-------------------------------------------------------------
// Counting Semaphore
```

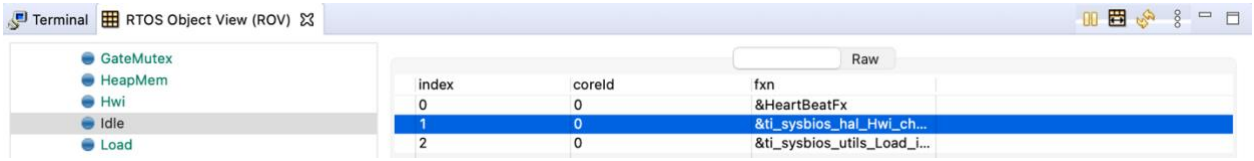
```
-------------
void Timer_ISR(void)
{
   TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);        // must clear timer flag
FROM timer

   if(tCount == 10)
   {
      Semaphore_post(ADCTasksem);
   }
   else if(tCount == 20)
   {
      Semaphore_post(UARTOutsem);
   }
   else if(tCount == 30)
   {
      Semaphore_post(swRsem);
   }
   else if(tCount == 40)
   {
      Semaphore_post(I2COutsem);
   }
   else if(tCount == 50)
   {
      Semaphore_post(Heartsem);
      tCount = 0;
   }
```

```
    tCount++;
}
```

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




Terminal
RTOS Object View (ROV)
Queue
QueueDescriptor
Registry
Semaphore
Startup
Swi
SysMin
System
Task
Timer
All Modules
ti
Detailed
CallStacks
ReadyQs
Module
Raw
address label priority mode fxn arg0 arg1 stackSize stackBase curCoreId affini
0x20005630 ADCTask 1 Blocked ADCtask 0x0 0x0 2048 0x20000380 n/a n/a
0x2000567c UARTTask 1 Blocked UART_Out 0x0 0x0 2048 0x20000b80 n/a n/a
0x200056c8 switchReadT... 1 Running switchR... 0x0 0x0 2048 0x20001380 n/a n/a
0x20005714 I2CTask 1 Blocked I2C_Out 0x0 0x0 2048 0x20001b80 n/a n/a
0x20005760 ti.sysbios.kn... 0 Blocked ti_sysbi... 0x0 0x0 512 0x20002380 n/a n/a


Terminal
RTOS Object View (ROV)
GateMutex
HeapMem
Hwi
Idle
Load
Raw
index coreId fxn
0 0 &HeartBeatFx
1 0 &ti_sysbios_hal_Hwi_ch...
2 0 &ti_sysbios_utils_Load_i...

4. Declaration

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

"This assignment submission is my own, original work".
-Elmer Mejia