Itzel Becerril
Github root directory: https://github.com/HadidBuilds/TivaC_project_labs

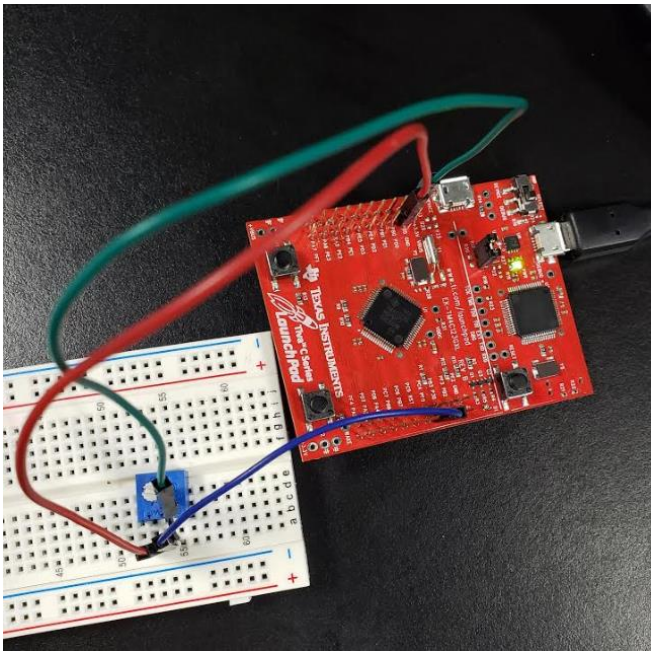
Date Submitted: **12/13/19**

TIVAC RTOS Assignment

In this assignment, I incorporated the concepts that I learned in the TIVAC RTOS labs. The goal of this assignment was to implement three tasks in our RTOS project. The three tasks consisted of ADC, UART, and a switch read. The way I implemented this project was starting with the Lab 8 code from the RTOS labs as a template. In lab 8 I learned to create one task that ran off a HWI using a semaphore. In this project, I had to create 3 tasks, 3 semaphores, and one HWI. I didn't change the HWI name, but I changed the timer to 30. Each task that was created had its own function. For each function task I created a semaphore. The reason why is because when the HWI clicks for 10ms it will trigger the semaphore in the ISR function that will then run the first task. At 20ms the second task will run. On 30ms the third task will run. Each task triggered by its respected semaphore. In the first task I had the board read the ADC value that was controlled by a potentiometer every 10ms. The second task was printing the ADC values, using UART, every 20ms. The third task updated the PWM if a switch was read. This task occurred every 30ms.

Youtube Link:
https://www.youtube.com/watch?v=SWxu3coGIns

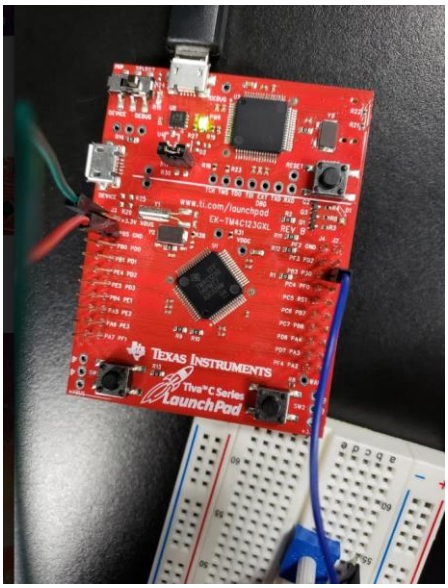
Modified Schematic:



This image shows the connections that I did

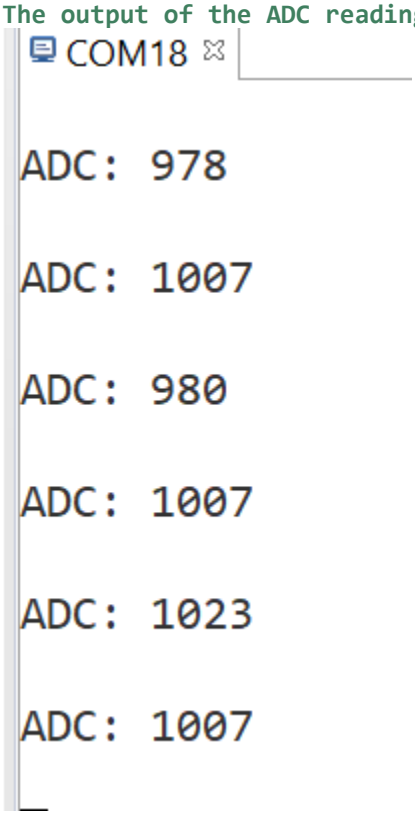
Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Itzel Becerril
Github root directory: https://github.com/HadidBuilds/TivaC_project_labs



A closeup picture of connections

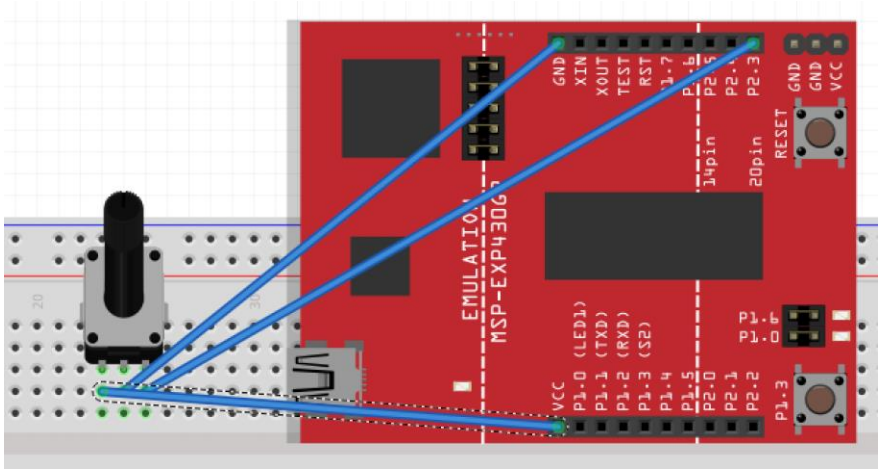
The output of the ADC reading on UART:



Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

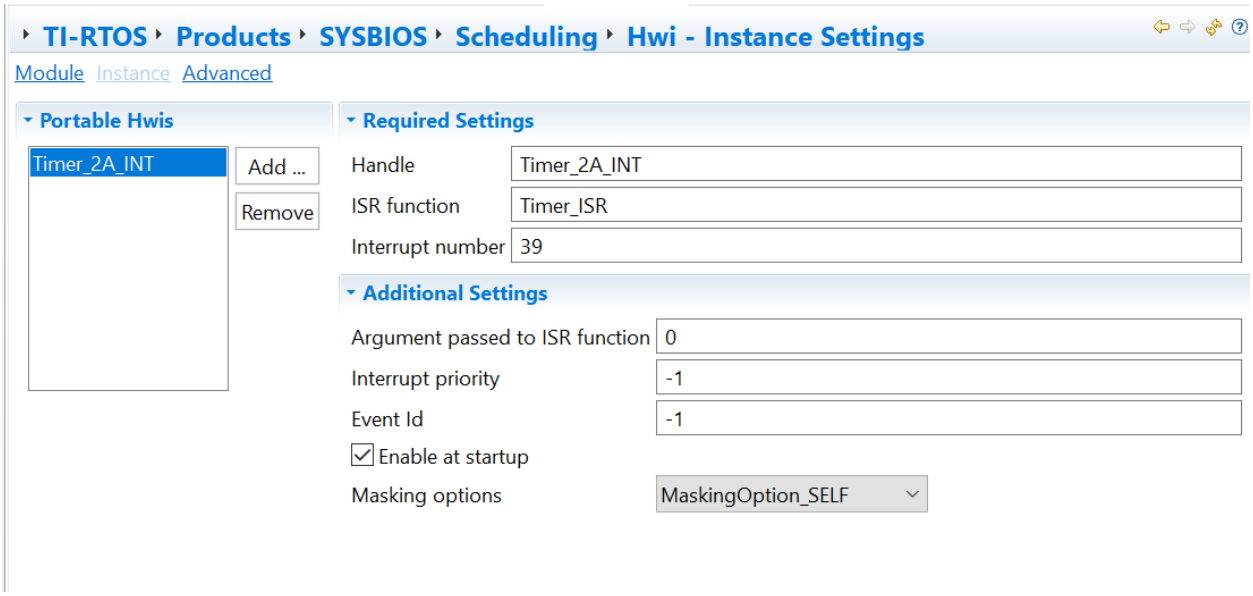
Itzel Becerril
Github root directory: https://github.com/HadidBuilds/TivaC_project_labs

Schematic of Connections



Configurations in the Config file:

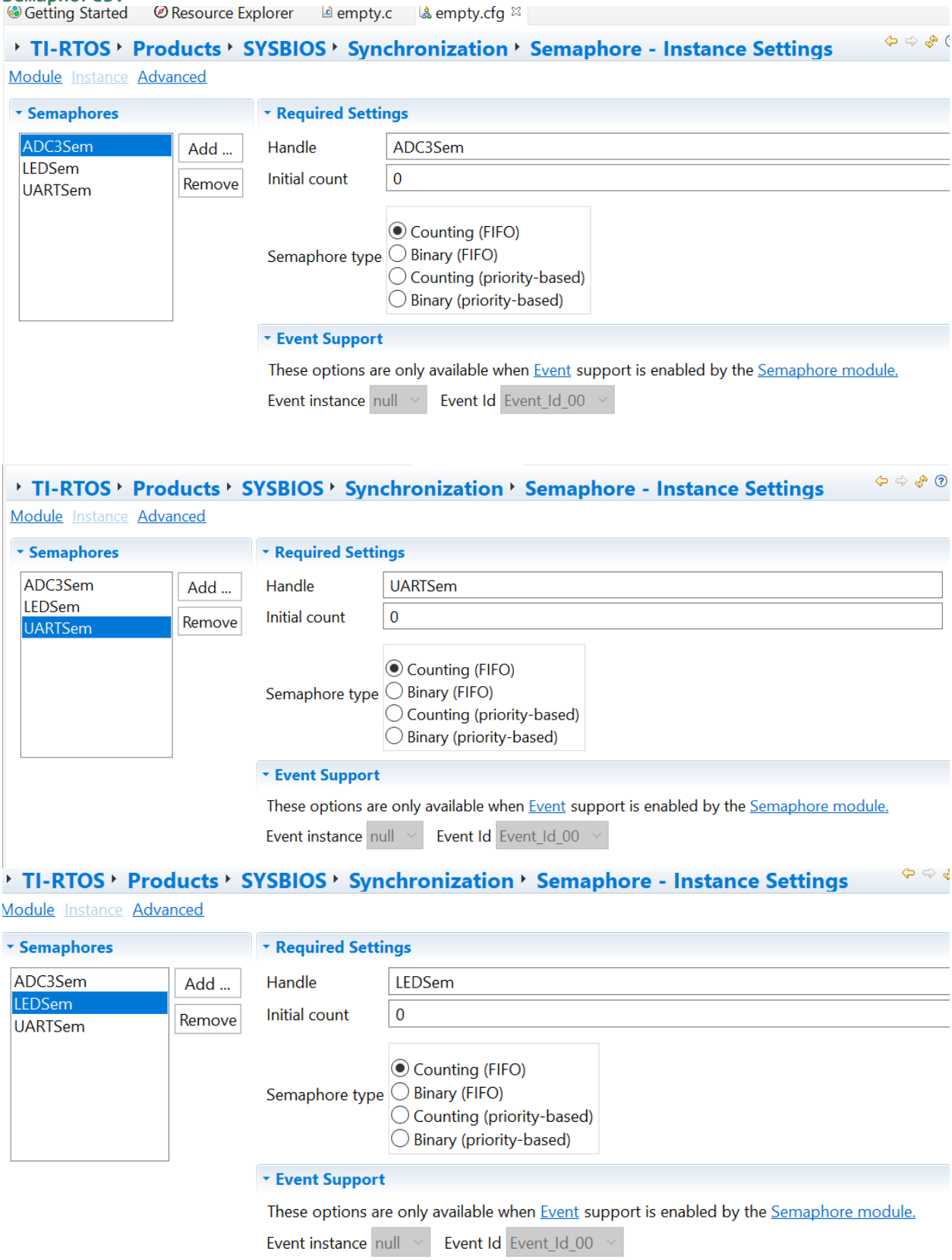
HWI:



Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Itzel Becerril
Github root directory: https://github.com/HadidBuilds/TivaC_project_labs

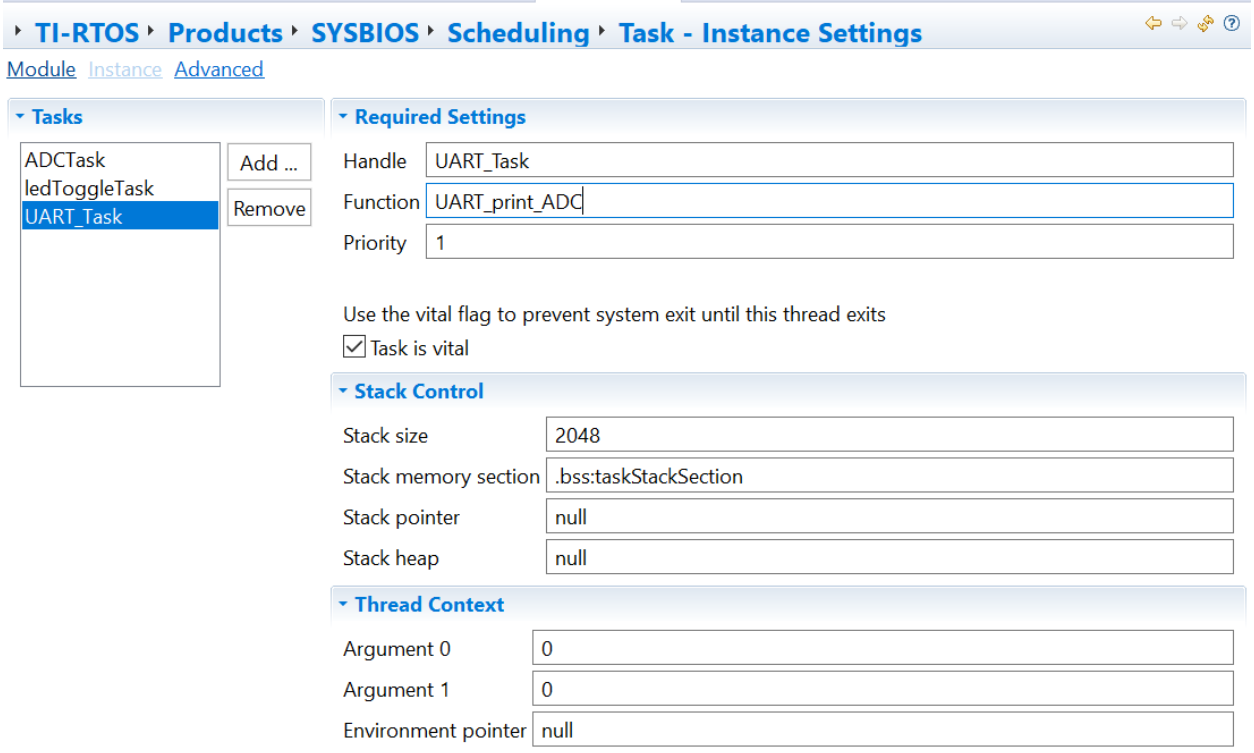
Semaphores:

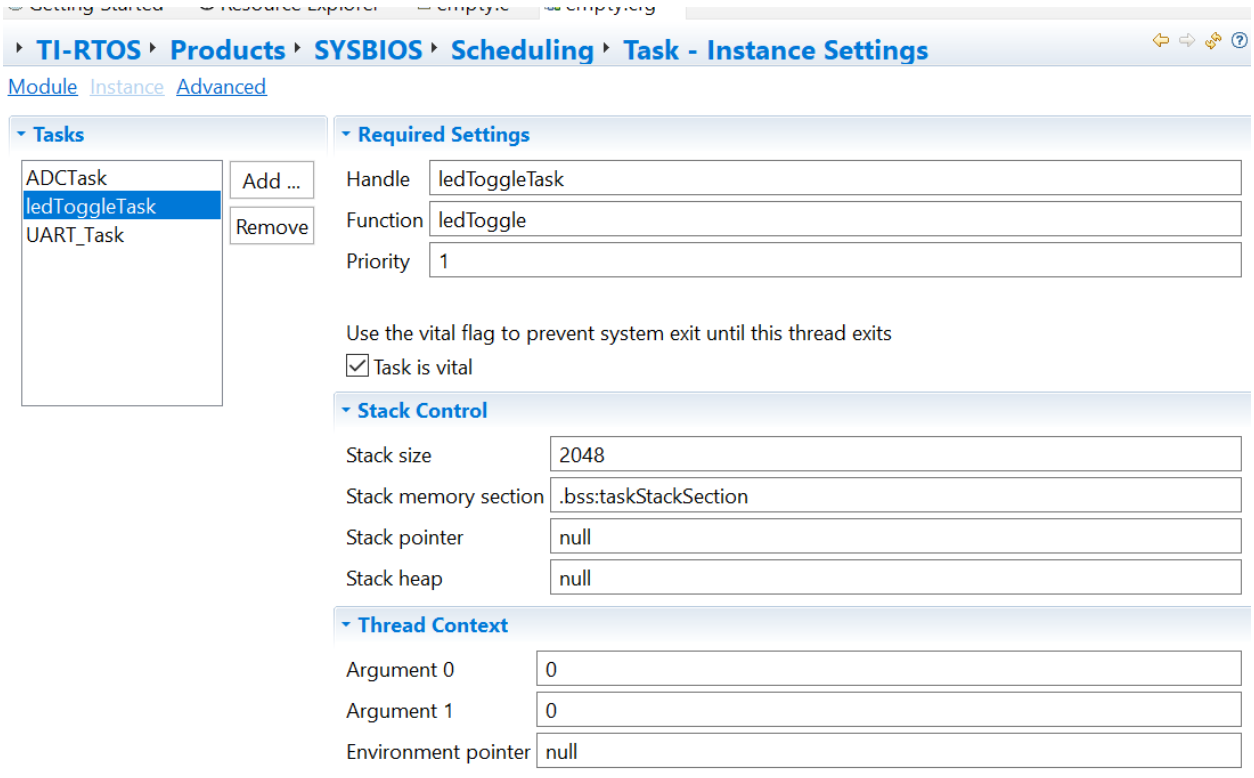


Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Itzel Becerril
Github root directory: https://github.com/HadidBuilds/TivaC_project_labs

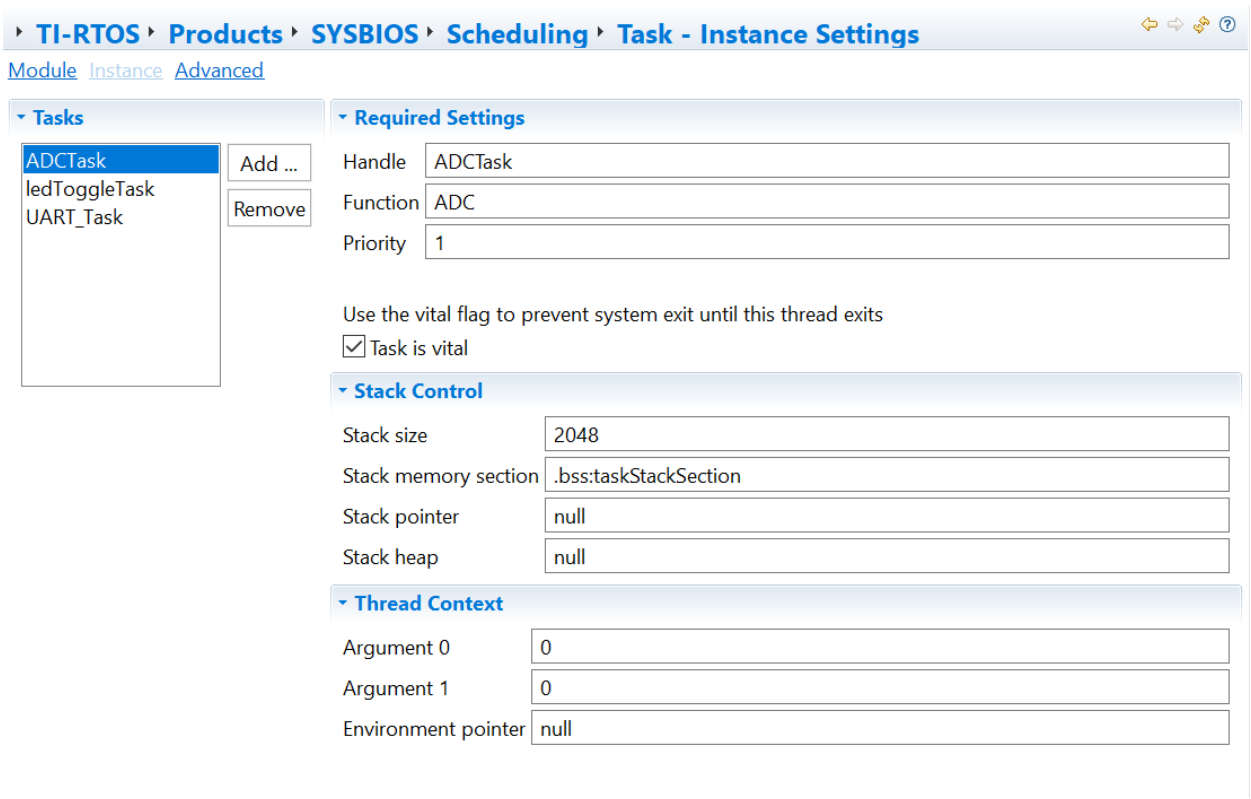
Tasks:





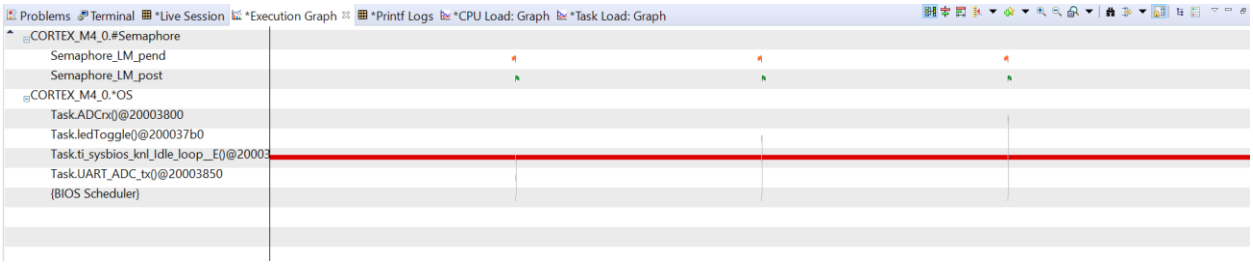
Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Itzel Becerril
Github root directory: https://github.com/HadidBuilds/TivaC_project_labs

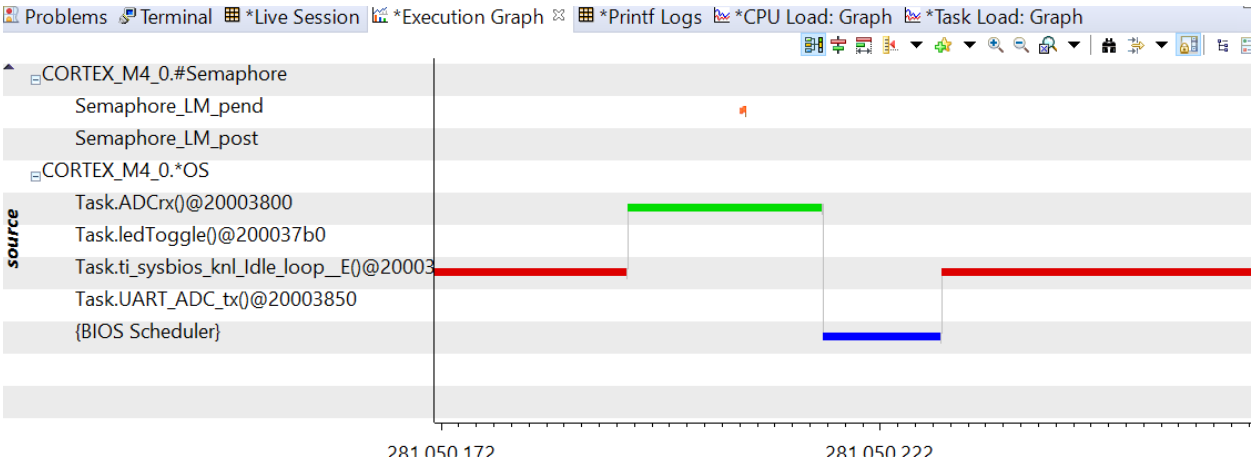


Results on Execution Graph:

Can see the 3 tasks below being triggered by HWI:



Up close of one of the tasks:



Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Modified Code:

EMPTY.c FILE

```
//*****************************************************************************
// Project: Blink TM4C BIOS Using Swi (SOLUTION)
// Author: Eric Wilbur
// Date: June 2014
//
// Note: The function call TimerIntClear(TIMER2_BASE,
TIMER_TIMA_TIMEOUT) HAS
//       to be in the ISR. This fxn clears the TIMER's interrupt flag
coming
//       from the peripheral - it does NOT clear the CPU interrupt
flag - that
//       is done by hardware. The author struggled figuring this part
out - hence
//       the note. And, in the Swi lab, this fxn must be placed in the
//       Timer_ISR fxn because it will be the new ISR.
//
// Follow these steps to create this project in CCSv6.0:
// 1. Project -> New CCS Project
// 2. Select Template:
//    - TI-RTOS for Tiva-C -> Driver Examples -> EK-TM4C123 LP ->
Example Projects ->
//      Empty Project
//    - Empty Project contains full instrumentation (UIA, RTOS
Analyzer) and
//      paths set up for the TI-RTOS version of MSP430Ware
// 3. Delete the following files:
//    - Board.h, empty.c, EK_TM4C123GXL.c/h, empty_readme.txt
// 4. Add main.c from TI-RTOS Workshop Solution file for this lab
// 5. Edit empty.cfg as needed (to add/subtract) BIOS services, delete
given Task
// 6. Build, load, run...
//*****************************************************************************


//---------------------------------------
// BIOS header files
//---------------------------------------
#include <xdc/std.h>                        //mandatory - have to
include first, for BIOS types
```

```
#include <ti/sysbios/BIOS.h>                //mandatory - if you call
APIs like BIOS_start()
#include <xdc/runtime/Log.h>                //needed for any
Log_info() call
#include <xdc/cfg/global.h>                 //header file for
statically defined objects/handles


//*****************************************
// TivaWare Header Files
//*****************************************
#include <stdint.h>
#include <stdbool.h>

#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/adc.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"


//*****************************************
// Prototypes
//*****************************************
void HWI_init(void);
void ledToggle(void);
void Timer_ISR(void);
void ADC_init();
void ADC(void);
void Cons_init(void);
void UART_print_ADC(void);


//*****************************************
// Globals have to be declared as volatile
//*****************************************
volatile int16_t Tog_Count = 0;
volatile int16_t Inst_Count = 0;
```

```
// ADCValues stores the ADC values from the TIvaC and the size has to
match the
// FIFO depth
uint32_t ADCValues[1];


// ADCval is used to store the ADC output var to UART
uint32_t ADCval ;

//---------------------------------------------------------------------
-------
// main()
//---------------------------------------------------------------------
-------
 void main(void){
    HWI_init();
    ADC_init();
    Cons_init();
    BIOS_start();
}

//---------------------------------------------------------------------
-------
// HWI_init()
//---------------------------------------------------------------------
-------
void HWI_init(void){
    uint32_t Period;

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

SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL
_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);

    // Turn on the LED
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3,
4);

    // Timer 2 setup code
```

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

    Period = (SysCtlClockGet() / 20);                   // period =
CPU clk div 20 (50ms)
    TimerLoadSet(TIMER2_BASE, TIMER_A, Period);         // set Timer 2
period

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

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

//-------------------------------------------------------------------
-------
// ledToggle()
//-------------------------------------------------------------------
-------
void ledToggle(void){
    while(1){
        Semaphore_pend(LEDSem, BIOS_WAIT_FOREVER);

        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2)){
            GPIOPinWrite(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
        }
        else{
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
        }

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

        Log_info1("LED TOGGLED [%u] TIMES",Tog_Count);     // send
toggle count to UIA
    }

}

//-------------------------------------------------------------------
-------
// Timer ISR - called by BIOS Hwi (see app.cfg)
```

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

    if(Inst_Count == 1) {
        Semaphore_post(ADC3Sem);
    }

    else if (Inst_Count == 2) {
        Semaphore_post(UARTSem);
    }

    else if(Inst_Count == 3) {
        Semaphore_post(LEDSem);                                  //
post LEDSwi
        Inst_Count = 0;
    }
    Inst_Count++;
}

//-------------------------------------------------------------------
-------
//ADC_init
//-------------------------------------------------------------------
-------
void ADC_init() {

    // The PE0 peripheral must be enabled for use.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlDelay(3);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlDelay(3);

    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0);    //Configure ADC
pin: PE0

    // Sample from ADC0_BASE using sequencer 3
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

    // Here sequence 3 is configured to be zero steps when it samples
from adc
    // channel 3 with the interrupt flag set when done sampling and
set the
    // ADC_CLT_END.
```

```
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH3  |
ADC_CTL_IE | ADC_CTL_END);

    ADCSequenceEnable(ADC0_BASE, 3);

    // clear any previous flags
    ADCIntClear(ADC0_BASE, 3);
}

//-------------------------------------------------------------------------
-------
//ADC
//-------------------------------------------------------------------------
-------
void ADC(void) {

    while(1) {
        Semaphore_pend(ADC3Sem, BIOS_WAIT_FOREVER);

        // tell processor to trigger the ADC0_BASE
        ADCProcessorTrigger(ADC0_BASE, 3);

        // wait for completion
        while(!ADCIntStatus(ADC0_BASE, 3, false)){}

        // Clear ADC flag
        ADCIntClear(ADC0_BASE, 3);

        // store ADC0_BASE value into ADCValues
        ADCSequenceDataGet(ADC0_BASE, 3, ADCValues);
        ADCval = ADCValues[0];
    }

}

//-------------------------------------------------------------------------
-------
//Cons_init
//-------------------------------------------------------------------------
-------
void Cons_init(void){

    // Enable GPIO port A to use with UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
```

```
    // Enable UART0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    // set default 16MHz frequency
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // config UART0 at BAUD: 115200 F:16MHz
    UARTStdioConfig(0, 115200, 16000000);
}

//-------------------------------------------------------------------------
-------
//UART_print_ADC
//-------------------------------------------------------------------------
-------
void UART_print_ADC(void) {
    while(1) {
        Semaphore_pend(UARTSem, BIOS_WAIT_FOREVER);
        UARTprintf("ADC: %d\n\n", ADCval);
    }
}
```

EMPTY.cfg FILE

```
/*
 * Copyright (c) 2013, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above
copyright
 *    notice, this list of conditions and the following disclaimer in
the
 *    documentation and/or other materials provided with the
distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names
of
```

```
 *    its contributors may be used to endorse or promote products
derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO,
 * 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.
 */

/*
 *  ======== empty.cfg ========
 */

/* ================ General configuration ================ */
var Defaults = xdc.useModule('xdc.runtime.Defaults');
var Diags = xdc.useModule('xdc.runtime.Diags');
var Error = xdc.useModule('xdc.runtime.Error');
var Log = xdc.useModule('xdc.runtime.Log');
var Main = xdc.useModule('xdc.runtime.Main');
var Memory = xdc.useModule('xdc.runtime.Memory');
var System = xdc.useModule('xdc.runtime.System');
var Text = xdc.useModule('xdc.runtime.Text');

var BIOS = xdc.useModule('ti.sysbios.BIOS');
var Clock = xdc.useModule('ti.sysbios.knl.Clock');
var Semaphore = xdc.useModule('ti.sysbios.knl.Semaphore');
var Hwi = xdc.useModule('ti.sysbios.hal.Hwi');
var HeapMem = xdc.useModule('ti.sysbios.heaps.HeapMem');
//var FatFS = xdc.useModule('ti.sysbios.fatfs.FatFS');

/* ================ System configuration ================ */
```

```
var SysMin = xdc.useModule('xdc.runtime.SysMin');
var Task = xdc.useModule('ti.sysbios.knl.Task');
System.SupportProxy = SysMin;

/* ================ Logging configuration ================ */
var LoggingSetup = xdc.useModule('ti.uia.sysbios.LoggingSetup');

/* ================ Kernel configuration ================ */
/* Use Custom library */
var BIOS = xdc.useModule('ti.sysbios.BIOS');
BIOS.libType = BIOS.LibType_Custom;
BIOS.logsEnabled = true;
BIOS.assertsEnabled = true;
var hwi0Params = new Hwi.Params();
hwi0Params.instance.name = "Timer_2A_INT";
Program.global.Timer_2A_INT = Hwi.create(39, "&Timer_ISR",
hwi0Params);
Program.stack = 1024;
BIOS.heapSize = 0;
BIOS.cpuFreq.lo = 40000000;
LoggingSetup.sysbiosSwiLogging = false;
var task0Params = new Task.Params();
task0Params.instance.name = "ledToggleTask";
Program.global.ledToggleTask = Task.create("&ledToggle", task0Params);
var semaphore0Params = new Semaphore.Params();
semaphore0Params.instance.name = "LEDSem";
Program.global.LEDSem = Semaphore.create(null, semaphore0Params);
LoggingSetup.loadTaskLogging = true;
LoggingSetup.sysbiosSemaphoreLogging = true;
var semaphore1Params = new Semaphore.Params();
semaphore1Params.instance.name = "ADC3Sem";
Program.global.ADC3Sem = Semaphore.create(null, semaphore1Params);
var task1Params = new Task.Params();
task1Params.instance.name = "ADCTask";
Program.global.ADCTask = Task.create("&ADC", task1Params);
var semaphore2Params = new Semaphore.Params();
semaphore2Params.instance.name = "UARTSem";
Program.global.UARTSem = Semaphore.create(0, semaphore2Params);
var task2Params = new Task.Params();
task2Params.instance.name = "UART_Task";
Program.global.UART_Task = Task.create("&UART_print_ADC",
task2Params);
```