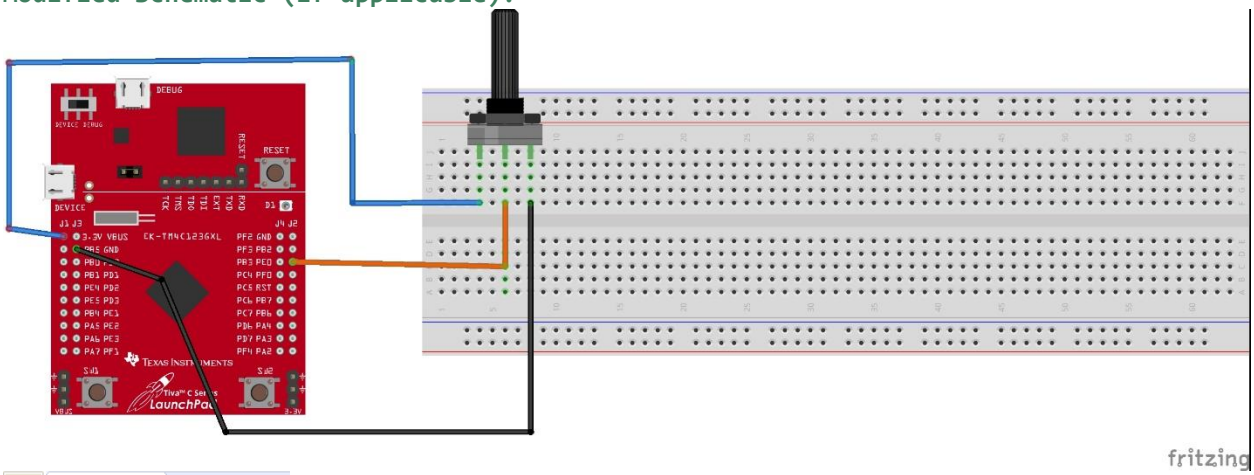
Ricky Perez
Github root directory: https://github.com/RickyPerez79/da_submissions_2019

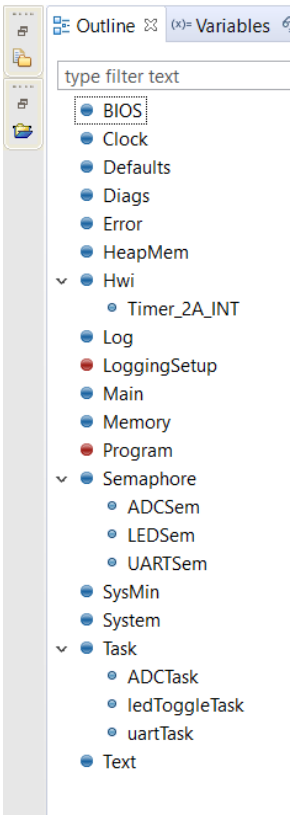
Date Submitted: **12/13/19**

RTOS Assignment:

Youtube Link: https://youtu.be/MKfrTColR1k

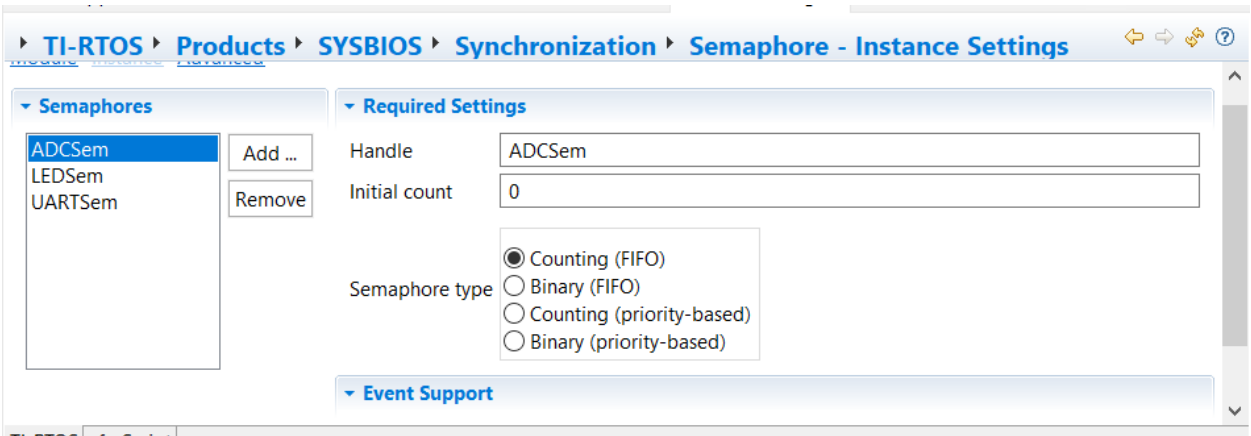
Modified Schematic (if applicable):

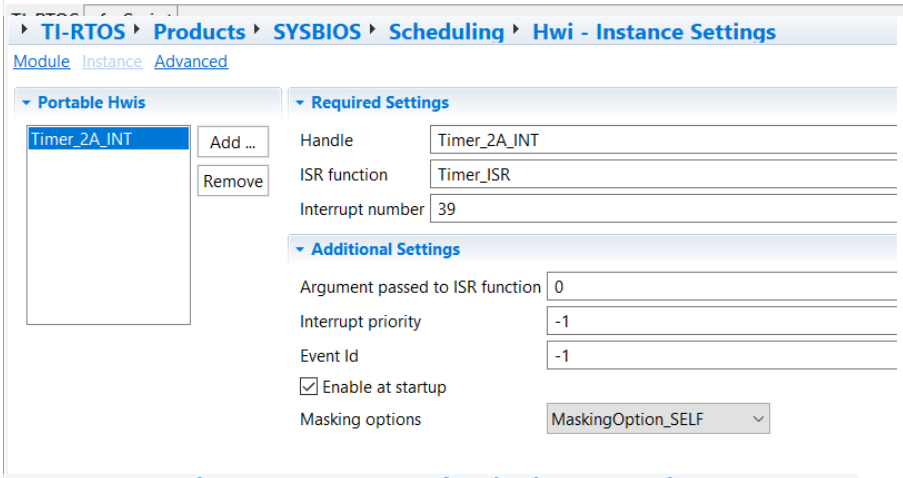


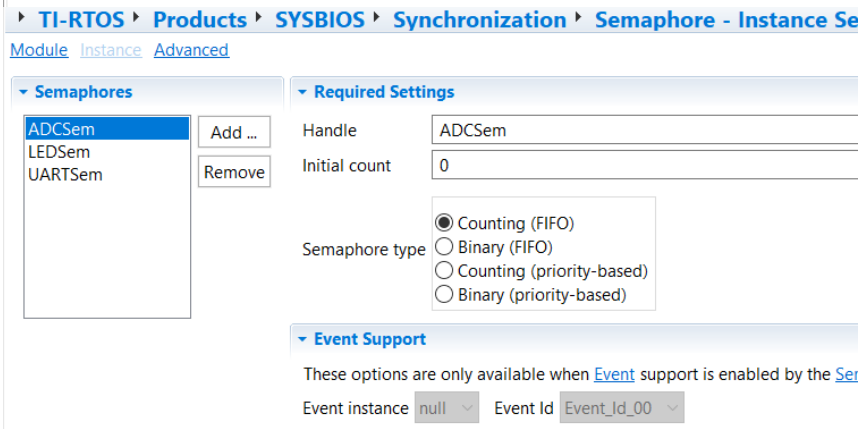


Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Ricky Perez
Github root directory: https://github.com/RickyPerez79/da_submissions_2019



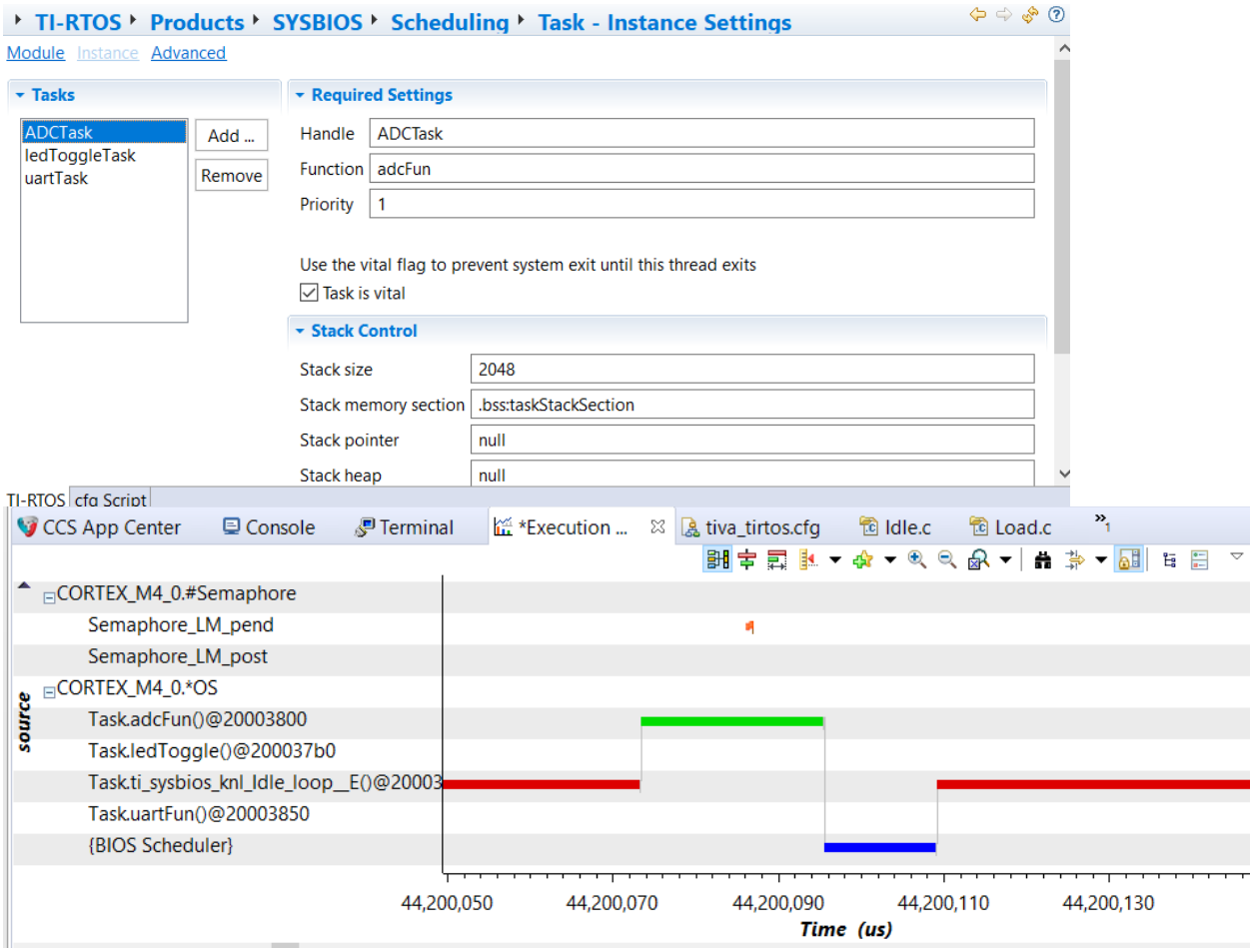




ADCTask, ledToggleTask, and uartTask have the same priority value of 1

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

Ricky Perez
Github root directory: https://github.com/RickyPerez79/da_submissions_2019



Modified Code:

```
// Insert code here
//-------------------------------------------------------------------------
-------------
// 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.
//
```

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

```
// 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 ledToggle(void);
void hardware_init(void);
void config_init(void);

void Timer_ISR(void);
void adcFun(void);
void uartFun(void);

//---------------------------------------
// Globals volatile
//---------------------------------------

volatile int16_t tCount = 0; // counter for toggle
volatile int16_t iCounter = 0; // Instance counter

// stores the ADC values from the TIvaC
uint32_t ADCValues[1];



uint32_t ADC_value ; // store the ADC output to UART

void main(void)
{

    hardware_init();

    config_init();

    BIOS_start();
}

//---------------------------------------------------
//                    HWI_init()
//---------------------------------------------------
void hardware_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,
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

    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_1))
        {
```

```
            GPIOPinWrite(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
        }
        else
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
        }

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

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

}

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

    if(iCounter == 1) {Semaphore_post(ADCSem);}// 10ms


    else if (iCounter == 2) {Semaphore_post(UARTSem);} // 20ms

    else if(iCounter == 3)
    {Semaphore_post(LEDSem); iCounter = 0;} // 30ms, post LEDSwi

    iCounter++;
}

void config_init(void)
{
    ////////////////////////////////// ADC
//////////////////////////////
    // 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);

////////////////////////////////////////////////////////////////////
////

    ////////////////////////////// UART
/////////////////////////////////////////

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // Enable GPIO port A
to use with UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); // Enable UART0 so
that we can configure the clock.
    GPIOPinConfigure(GPIO_PA0_U0RX); // Configure the pin muxing for
UART0 functions on port A0
    GPIOPinConfigure(GPIO_PA1_U0TX); // Configure the pin muxing for
UART0 functions on port A1

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC); // Use the
internal 16MHz oscillator as the UART clock source
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); //
Select the alternate (UART) function for these pins.

    // BAUD RATE: 115200 Frequency:16MHz
    UARTStdioConfig(0, 115200, 16000000); // Initialize the UART for
console I/O.

////////////////////////////////////////////////////////////////////
////

}
```

```
void adcFun(void)
{

    while(1)
    {
        Semaphore_pend(ADCSem, BIOS_WAIT_FOREVER);
        ADCProcessorTrigger(ADC0_BASE, 3); // trigger the ADC0_BASE
        while(!ADCIntStatus(ADC0_BASE, 3, false))
        {
            // wait until done
        }

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

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

}

void uartFun(void)
{
    while(1)
    {
        Semaphore_pend(UARTSem, BIOS_WAIT_FOREVER);
        UARTprintf("ADC value: %d\n\n", ADC_value);
    }
}
```
