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

**------------------------------------------------------------------------------------**

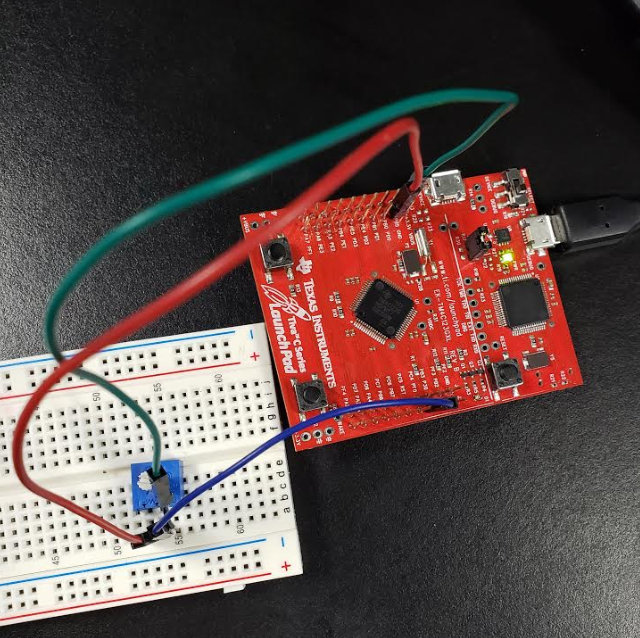
**TIVAC RTOS Assignment**

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.

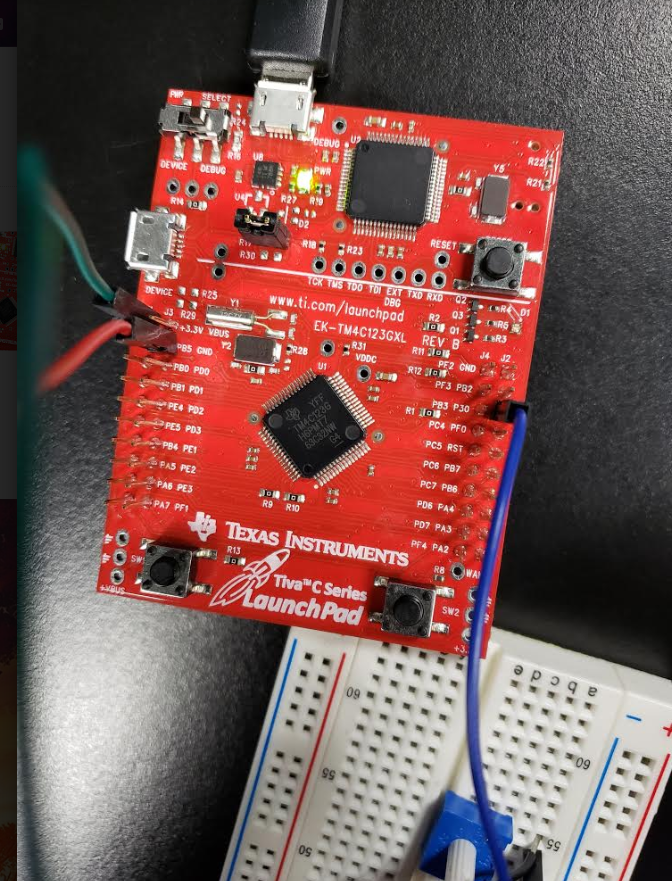
Youtube Link:

<https://www.youtube.com/watch?v=SWxu3coGIns>

**Modified Schematic:**

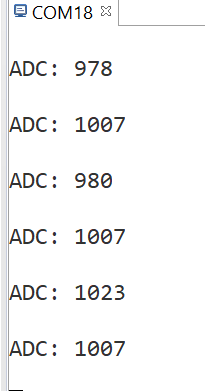


**This image shows the connections that I did**

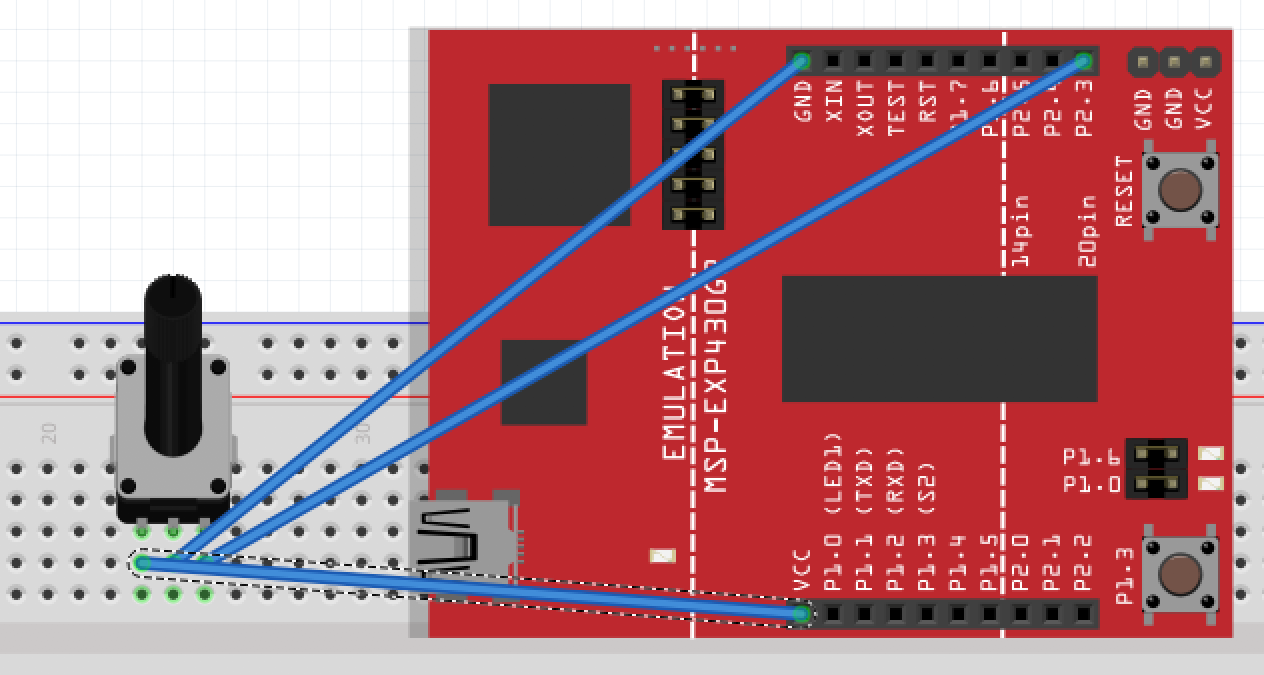


**A closeup picture of connections**

**The output of the ADC reading on UART:**

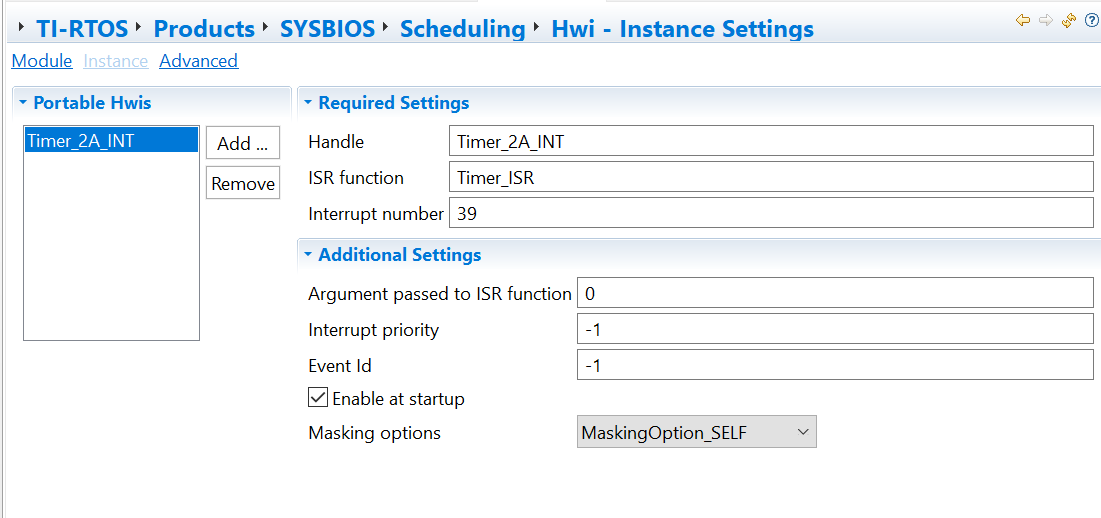


**Schematic of Connections**

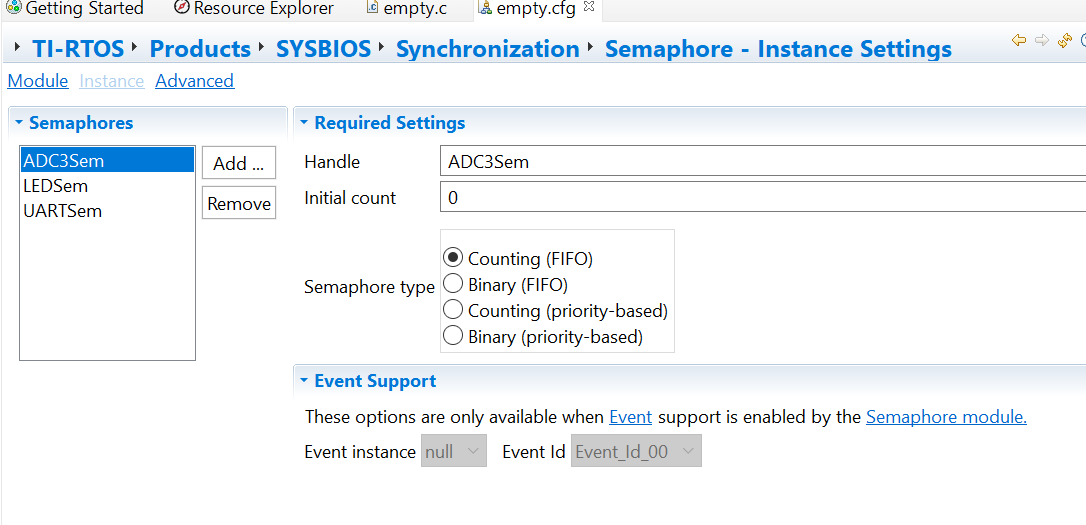


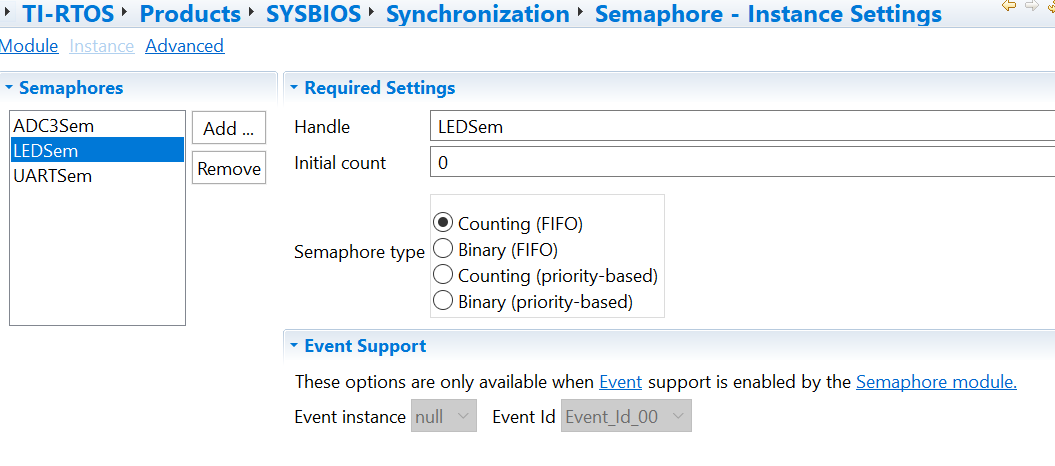
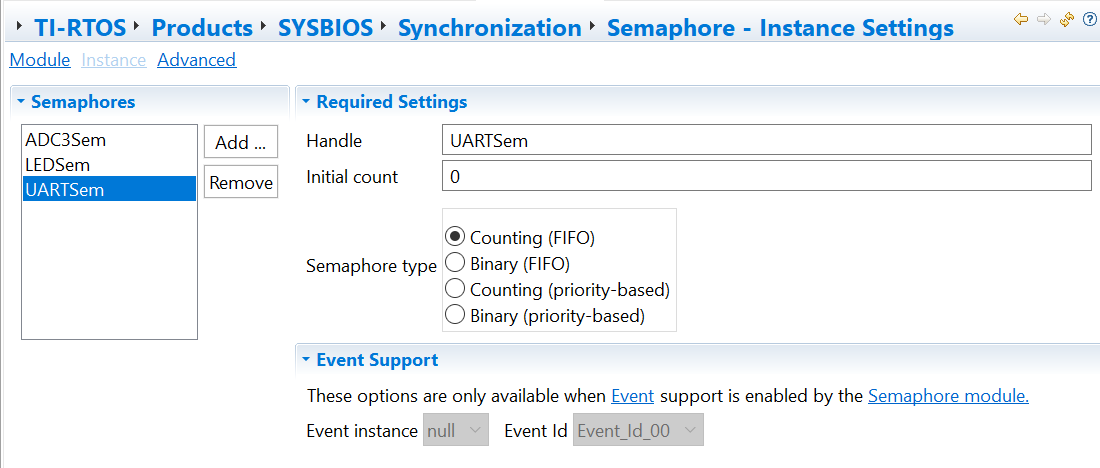
**Configurations in the Config file:**

**HWI:**

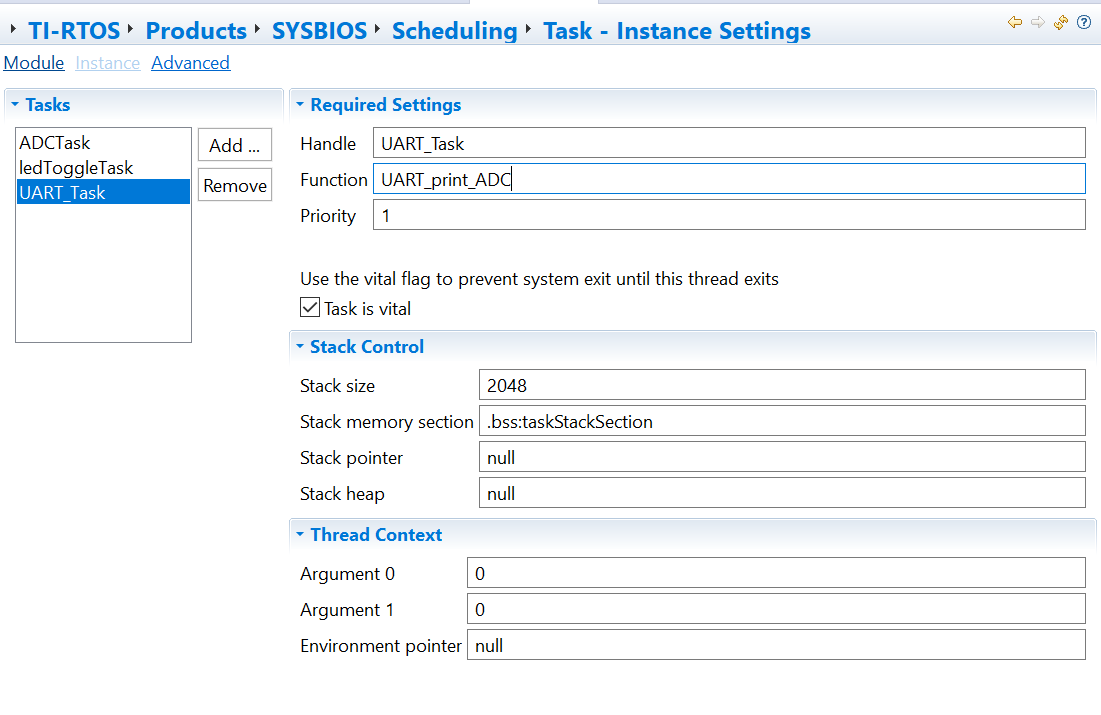


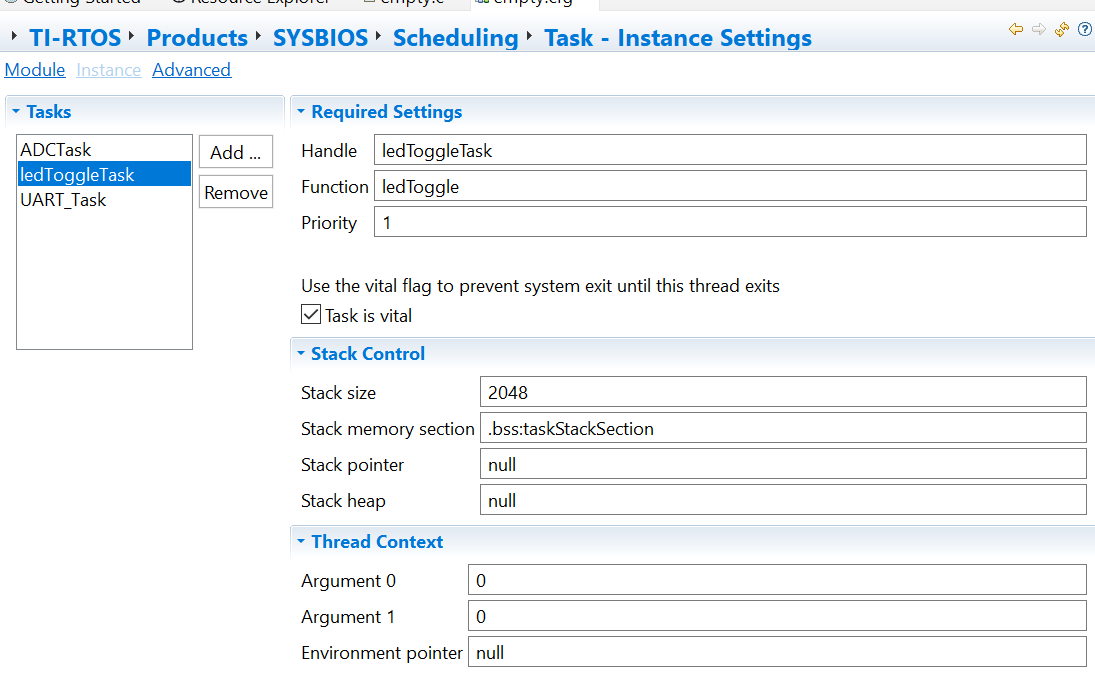
**Semaphores:**

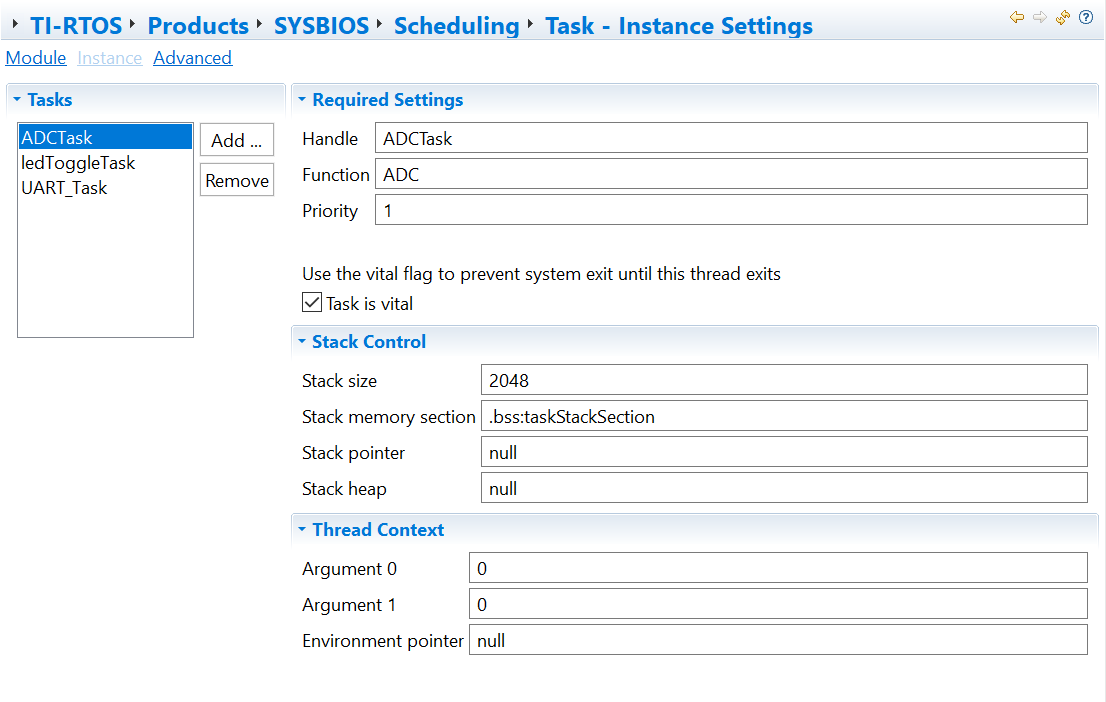




**Tasks:**

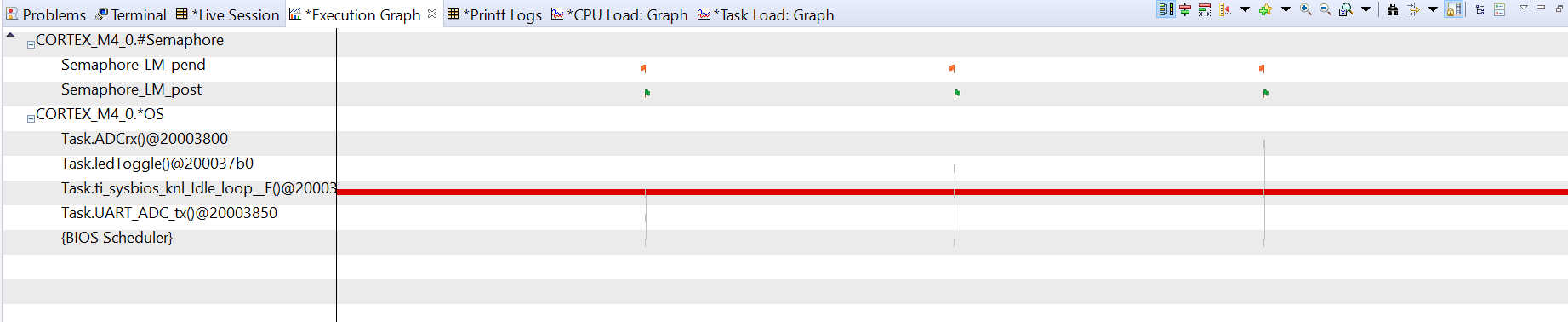




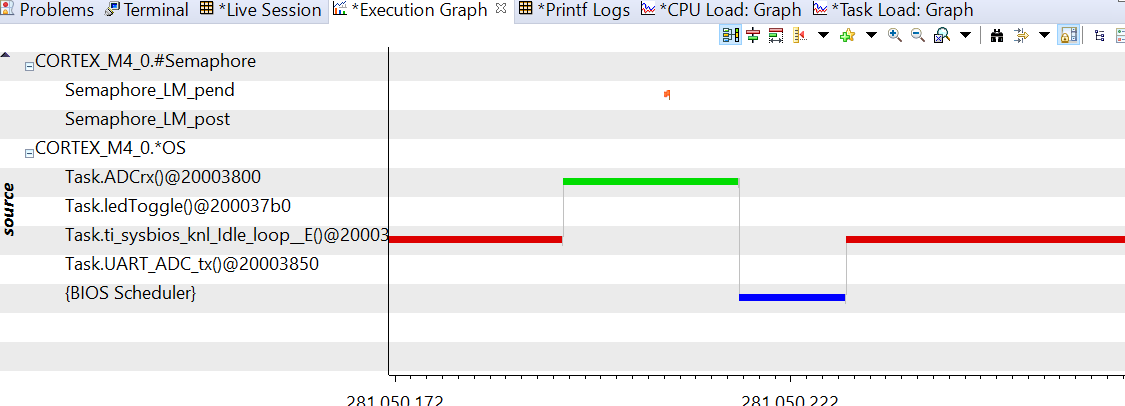


**Results on Execution Graph:**

Can see the 3 tasks below being triggered by HWI:



**Up close of one of the tasks:**



**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);