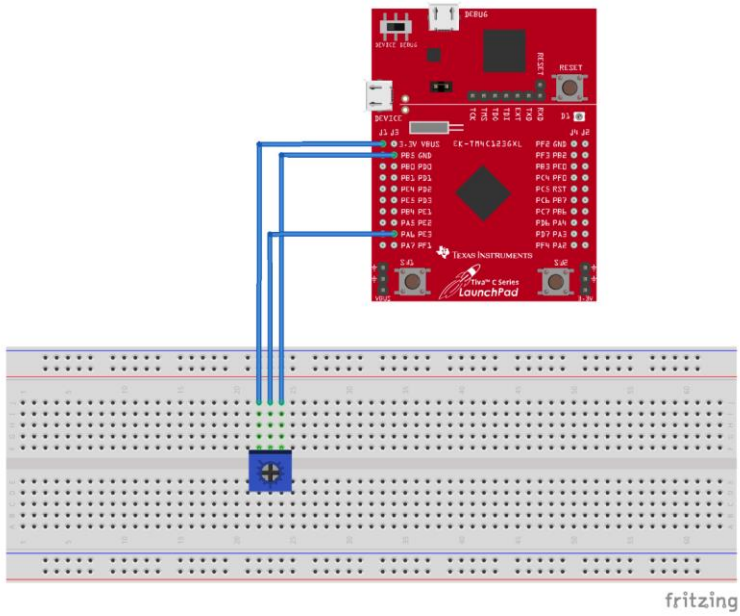
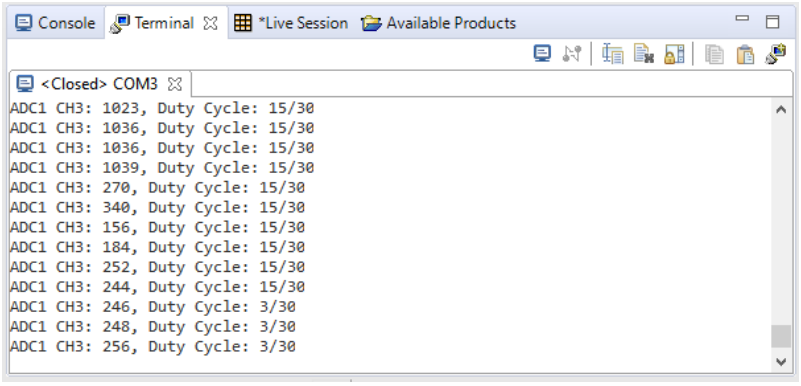
Chris Barr
Github root directory: https://github.com/BarrChris/submissions_da_F19

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

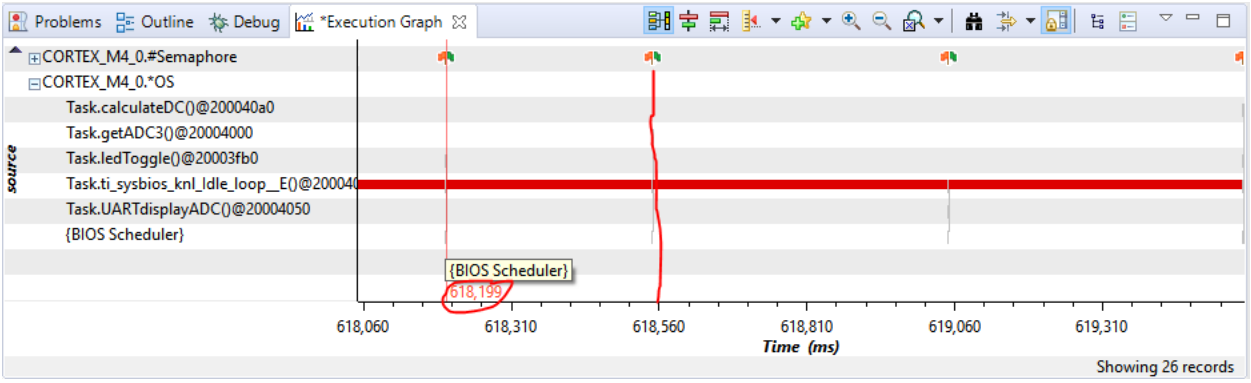
Modified Schematic (if applicable):



Schematic Drawn



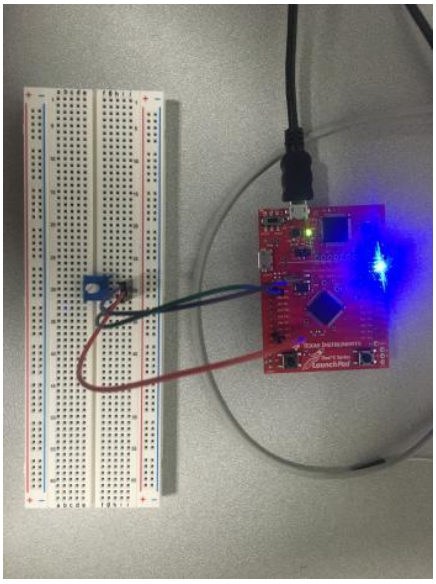
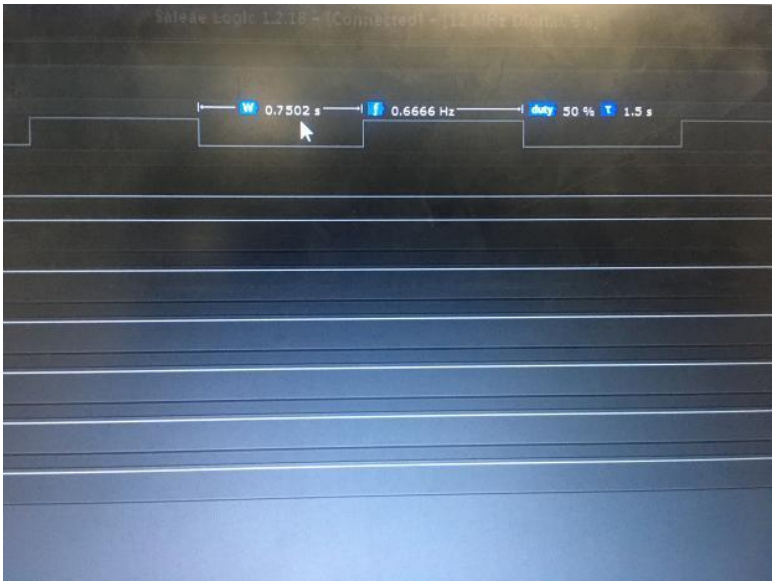
UART and ADC Conversion



Execution Graph showing the delay between the semaphores which shows here it's way above 30ms

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

Chris Barr
Github root directory: https://github.com/BarrChris/submissions_da_F19

Board Setup for configuration (left), and 50% DC waveform (right)

Task 01: ADC Task

Modified Code:

```
// Initializes ADC1
void initADC() {

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);
    SysCtlDelay(3);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlDelay(3);

    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);    //Configures pin to PE3
for ADC1


    //
    // Enable sample sequence 3 with a processor signal trigger.  Sequence 3
    // will do a single sample when the processor sends a singal to start the
    // conversion.  Each ADC module has 4 programmable sequences, sequence 0
    // to sequence 3.  This example is arbitrarily using sequence 3.
    //
    ADCSequenceConfigure(ADC1_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

    //
    // Configure step 0 on sequence 3.  Sample the ADC CHANNEL 3
    // (PE0) and configure the interrupt flag (ADC_CTL_IE) to be set
    // when the sample is done.  Tell the ADC logic that this is the last
```

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

```
    // conversion on sequence 3 (ADC_CTL_END).  Sequence 3 has only one
    // programmable step.  Sequence 1 and 2 have 4 steps, and sequence 0 has
    // 8 programmable steps.  Since we are only doing a single conversion
using
    // sequence 3 we will only configure step 0.  For more information on the
    // ADC sequences and steps, reference the datasheet.
    //
    ADCSequenceStepConfigure(ADC1_BASE, 3, 0, ADC_CTL_CH3  | ADC_CTL_IE |
ADC_CTL_END);

    //
    // Since sample sequence 3 is now configured, it must be enabled.
    //
    ADCSequenceEnable(ADC1_BASE, 3);

    //
    // Clear the interrupt status flag.  This is done to make sure the
    // interrupt flag is cleared before we sample.
    //
    ADCIntClear(ADC1_BASE, 3);
}

//*****************************************************************************
// ADC1 from CH3
//
// Converts and grabs values for the ADC
//*****************************************************************************
void getADC3(void) {

    while(1) {
        Semaphore_pend(ADC3Sem, BIOS_WAIT_FOREVER);
        //
        // Trigger the ADC conversion.
        //
        ADCProcessorTrigger(ADC1_BASE, 3);

        //
        // Wait for conversion to be completed.
        //
        while(!ADCIntStatus(ADC1_BASE, 3, false))
        {
        }

        //
        // Clear the ADC interrupt flag.
        //
        ADCIntClear(ADC1_BASE, 3);

        //
        // Read ADC Value.
        //
        ADCSequenceDataGet(ADC1_BASE, 3, ADCValues);
        ADC3out = ADCValues[0];
    }

}
```

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

Chris Barr
Github root directory: https://github.com/BarrChris/submissions_da_F19

Task 02: UART Display Task

Modified Code:

```
// initializes Console
void InitConsole(void)
{
    //
    // Enable GPIO port A which is used for UART0 pins.
    // TODO: change this to whichever GPIO port you are using.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Configure the pin muxing for UART0 functions on port A0 and A1.
    // This step is not necessary if your part does not support pin muxing.
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);

    //
    // Enable UART0 so that we can configure the clock.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

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

    //
    // Select the alternate (UART) function for these pins.
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

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

//*****************************************************************************
// UART
//
// Displays the ADC as projected from the potentiometer
//*****************************************************************************
void UARTdisplayADC(void)
{
    while(1)
    {
        Semaphore_pend(UARTSem, BIOS_WAIT_FOREVER);
        UARTprintf("ADC1 CH3: %d, Duty Cycle: %d/30\n", ADC3out, DC);
```

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

```
    }
}
```

Task 03: Switch Read Task

Modified Code:

```
//*****************************************************************************
// Read Switch
//
// Grabs the value of the ADC and switches the PWM
//*****************************************************************************
void calculateDC(void)
{
    while(1)
    {
        Semaphore_pend(SW_ReadSem, BIOS_WAIT_FOREVER);

        if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00)
        {
            if(ADC3out < 200)
                DC = 0;
            else if (ADC3out > 2000)
                DC = 30;
            else
                DC = 30 * ((float)ADC3out/2000.0);
        }
    }
}
```

Full Code:

tivac_tirtos.c

```
//*****************************************
// 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 hardware_init(void);
void ledToggle(void);
void Timer_ISR(void);
void initADC();
void getADC3(void);
void InitConsole(void);
void UARTdisplayADC(void);


//*****************************************
// Globals
//*****************************************
volatile int16_t i16ToggleCount = 0;
volatile int16_t i16InstanceCount = 0;
volatile int16_t DC = 30;
// This array is used for storing the data read from the ADC FIFO. It
// must be as large as the FIFO for the sequencer in use.  This example
// uses sequence 3 which has a FIFO depth of 1.  If another sequence
// was used with a deeper FIFO, then the array size must be changed.
//
uint32_t ADCValues[1];

//
// This variable is used to store the output of the ADC Channel 3
//
uint32_t ADC3out;



//*****************************************************************************
// main()
//*****************************************************************************
void main(void)
{

   hardware_init();
   initADC();
   InitConsole();
```

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

```
   BIOS_start();

}


//---------------------------------------------------------------------------
// hardware_init()
//
// inits GPIO pins for toggling the LED
//---------------------------------------------------------------------------
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|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);
      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);

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

      //Pushbutton setup
      GPIODirModeSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_4,
GPIO_DIR_MODE_IN);
      GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_4,
GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

      // Timer 2 setup code
      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);               // enable
Timer2A
      TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);            // periodic
configuration

      ui32Period = (SysCtlClockGet() / 20);                             //
period = 50ms
      TimerLoadSet(TIMER2_BASE, TIMER_A, ui32Period);             // sets
Timer2A period

      TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);            // enables
Timer2A int

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

}

// initializes Console
void InitConsole(void)
{
```

```
    //
    // Enable GPIO port A which is used for UART0 pins.
    // TODO: change this to whichever GPIO port you are using.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Configure the pin muxing for UART0 functions on port A0 and A1.
    // This step is not necessary if your part does not support pin muxing.
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);

    //
    // Enable UART0 so that we can configure the clock.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

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

    //
    // Select the alternate (UART) function for these pins.
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

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

// Initializes ADC1
void initADC() {

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);
    SysCtlDelay(3);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlDelay(3);

    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);    //Configures pin to PE3
for ADC1


    //
    // Enable sample sequence 3 with a processor signal trigger.  Sequence 3
    // will do a single sample when the processor sends a singal to start the
    // conversion.  Each ADC module has 4 programmable sequences, sequence 0
    // to sequence 3.  This example is arbitrarily using sequence 3.
    //
    ADCSequenceConfigure(ADC1_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
```

```
    //
    // Configure step 0 on sequence 3.  Sample the ADC CHANNEL 3
    // (PE0) and configure the interrupt flag (ADC_CTL_IE) to be set
    // when the sample is done.  Tell the ADC logic that this is the last
    // conversion on sequence 3 (ADC_CTL_END).  Sequence 3 has only one
    // programmable step.  Sequence 1 and 2 have 4 steps, and sequence 0 has
    // 8 programmable steps.  Since we are only doing a single conversion
using
    // sequence 3 we will only configure step 0.  For more information on the
    // ADC sequences and steps, reference the datasheet.
    //
    ADCSequenceStepConfigure(ADC1_BASE, 3, 0, ADC_CTL_CH3  | ADC_CTL_IE |
ADC_CTL_END);

    //
    // Since sample sequence 3 is now configured, it must be enabled.
    //
    ADCSequenceEnable(ADC1_BASE, 3);

    //
    // Clear the interrupt status flag.  This is done to make sure the
    // interrupt flag is cleared before we sample.
    //
    ADCIntClear(ADC1_BASE, 3);
}

//*****************************************************************************
// ledToggle()
//
// toggles LED on Tiva-C LaunchPad
//*****************************************************************************
void ledToggle(void)
{
      while(1)
      {
          Semaphore_pend(LEDSem, BIOS_WAIT_FOREVER);

          // LED values - 2=RED, 4=BLUE, 8=GREEN
              if (DC == 0)
                  GPIOPinWrite(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
              else 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);
              }

              i16ToggleCount += 1;                                    //
toggle counter

              Log_info1("LED TOGGLED [%u] TIMES",i16ToggleCount);     // logs
toggles
```

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

```
      }

}


//-------------------------------------------------------------------------
// Timer ISR - called by BIOS Hwi (see app.cfg)
//
// Posts Swi (or later a Semaphore) to toggle the LED
//-------------------------------------------------------------------------
void Timer_ISR(void)
{
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);               // clears
timer
    if (i16InstanceCount == DC) {
            Semaphore_post(LEDSem);
    }
    if(i16InstanceCount == 10) {
        Semaphore_post(ADC3Sem);
    }

    else if (i16InstanceCount == 20) {
        Semaphore_post(UARTSem);
    }

    else if(i16InstanceCount == 30) {
        Semaphore_post(SW_ReadSem);
        Semaphore_post(LEDSem);
        i16InstanceCount = 0;
    }

    i16InstanceCount++;
}

//-------------------------------------------------------------------------
// Read Switch
//
// Grabs the value of the ADC and switches the PWM
//-------------------------------------------------------------------------
void calculateDC(void)
{
    while(1)
    {
        Semaphore_pend(SW_ReadSem, BIOS_WAIT_FOREVER);

        if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00)
        {
            if(ADC3out < 200)
                DC = 0;
            else if (ADC3out > 2000)
                DC = 30;
            else
                DC = 30 * ((float)ADC3out/2000.0);
        }
    }
}
```

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

```
//*****************************************************************************
// ADC1 from CH3
//
// Converts and grabs values for the ADC
//*****************************************************************************
void getADC3(void) {

    while(1) {
        Semaphore_pend(ADC3Sem, BIOS_WAIT_FOREVER);
        //
        // Trigger the ADC conversion.
        //
        ADCProcessorTrigger(ADC1_BASE, 3);

        //
        // Wait for conversion to be completed.
        //
        while(!ADCIntStatus(ADC1_BASE, 3, false))
        {
        }

        //
        // Clear the ADC interrupt flag.
        //
        ADCIntClear(ADC1_BASE, 3);

        //
        // Read ADC Value.
        //
        ADCSequenceDataGet(ADC1_BASE, 3, ADCValues);
        ADC3out = ADCValues[0];
    }

}

//*****************************************************************************
// UART
//
// Displays the ADC as projected from the potentiometer
//*****************************************************************************
void UARTdisplayADC(void)
{
    while(1)
    {
        Semaphore_pend(UARTSem, BIOS_WAIT_FOREVER);
        UARTprintf("ADC1 CH3: %d, Duty Cycle: %d/30\n", ADC3out, DC);
    }
}
```

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

tivac_tirtos.cfg

```
/*
 * 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;
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 = "getADC3Task";
Program.global.getADC3Task = Task.create("&getADC3", 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 = "UARTdisplayADCTask";
Program.global.UARTdisplayADCTask = Task.create("&UARTdisplayADC",
task2Params);
var hwi1Params = new Hwi.Params();
hwi1Params.instance.name = "Timer_2A_int";
Program.global.Timer_2A_int = Hwi.create(39, "&Timer_ISR", hwi1Params);
var task3Params = new Task.Params();
task3Params.instance.name = "SW_Read";
Program.global.SW_Read = Task.create("&calculateDC", task3Params);
var semaphore3Params = new Semaphore.Params();
semaphore3Params.instance.name = "SW_ReadSem";
Program.global.SW_ReadSem = Semaphore.create(null, semaphore3Params);
```

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