Tenniel Takenaka-Fuller 2000722457

TIRTOS Midterm

**GOAL:**

- Create an ADC task
- Create a UART display task
- Create a switch read task
- Execute these tasks every 30 ms

**VIDEO & GITHUB LINK:**

**https://youtu.be/mAs8rjGA6k8**

https://github.com/TennielTakenaka/psychic-invention

**DELIVERABLES**:

The main project deliverable is to create an assignment that will execute an ADC task, UART display task, and a switch read task – each done in 30ms intervals. I will use CCS and the TIVA TM4C123GH6PM TIRTOS features. I will be delivering a demo of the project in a YouTube video, documentation of code (in this document), screenshots, and schematics.

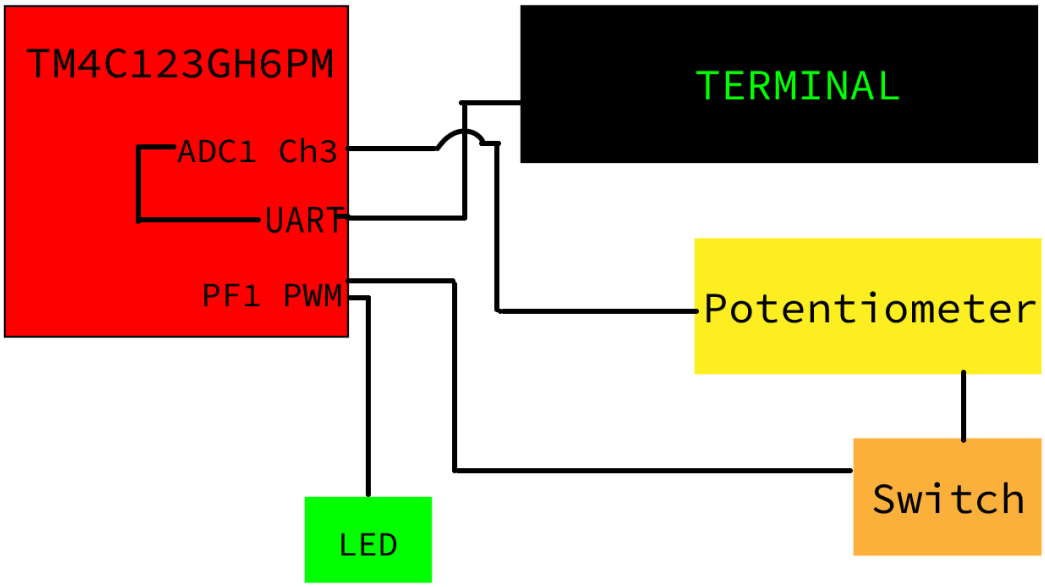
**COMPONENTS**:

- CCS: Code Composer Studio. It's an IDE that I will use to run the TIRTOS midterm on in conjunction with the Tiva C. It has the CCS compiler that I will use to compile the project.
- TIVA TM4C123GH6PM: This is the main device I am running the real time operating system on. I will be using its ADC, UART, PWM, and switch button features.
- ADC: I use and initialized ADC1 Channel 3 of the TIVA C. To interface this component, I would use the TIVA C ADC library functions to do so. ADC stands for analog to digital converter, which converts analog signals into digital ones. It will amplify a signals strength or vary frequency to add or take away data. Limitations of ADC include not being able to connect a raw voltage to it (voltages < 0v or > 3.3V) and ADC can only be used with one pin at a time.
- PWM: I initialize a PWM, or pulse width modulation frequency, to signal to an LED (PF1) to turn on and set the PWM initial value of the duty cycle to zero. The PWM duty cycle does not change unless the switch is pressed. To interface this, I use the TIVAC library functions to do so. Limitations of PWM include noisy signals and it can also damage DC and AC motors (if they were to be connected).
- UART: UART, a Universal Asynchronous Receiver-Transmitter, is used in this project to display the current ADC value in the terminal at the 20$^{th}$ instance of the HWI. The UART is interfaced and initialized through driverlib/uart library functions and will print to the console using UARTprintf. Limitations of UART include that the size of the data in the frame is limited, the speed for data transfer is less compared to parallel communication, and the transmitter and receiver must agree to rules of transmission and appropriate baud rate must be selected (sometimes selecting the appropriate mutual connection may be hard to achieve).
- SWITCH BUTTON: The switch read task is used to check the status of SW1/SW2 to update the current value of duty cycle based on the ADC value. If the switch button is pressed, the duty cycle of the PWM will change. To interface the switch, I enable the GPIO Pin output, pin 4 so the switch can be read. I also do an interrupt handler so the switch functions can be carried out. A

limitation of the switch includes needing to debounce the switch so that a single press doesn't appear like multiple presses.

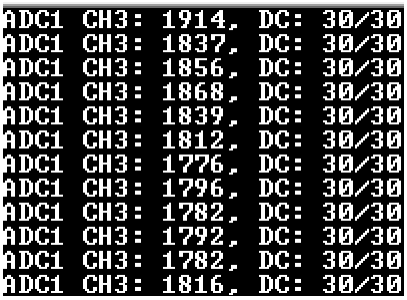
- POTENTIOMETER: I connect the potentiometer to the ADC pin (PE3) of the Tiva C. This will feed input and different values to the ADC pin. In turn, I would just need to interface the ADC to read these values, not the potentiometer. One limitation that I faced with the potentiometer is that it needs a large force to move its sliding contacts, causing damage to the potentiometer due to movement of the wiper.

**SCHEMATICS:**

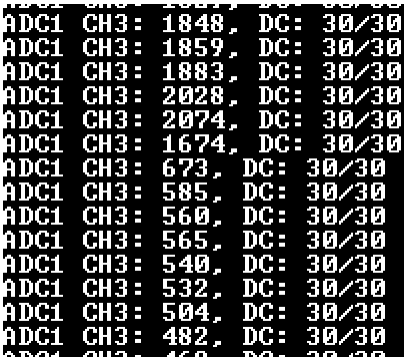



**SCREENSHOTS OF OUTPUT:**

1) Potentiometer turned all the way to the right when the board is first powered on:



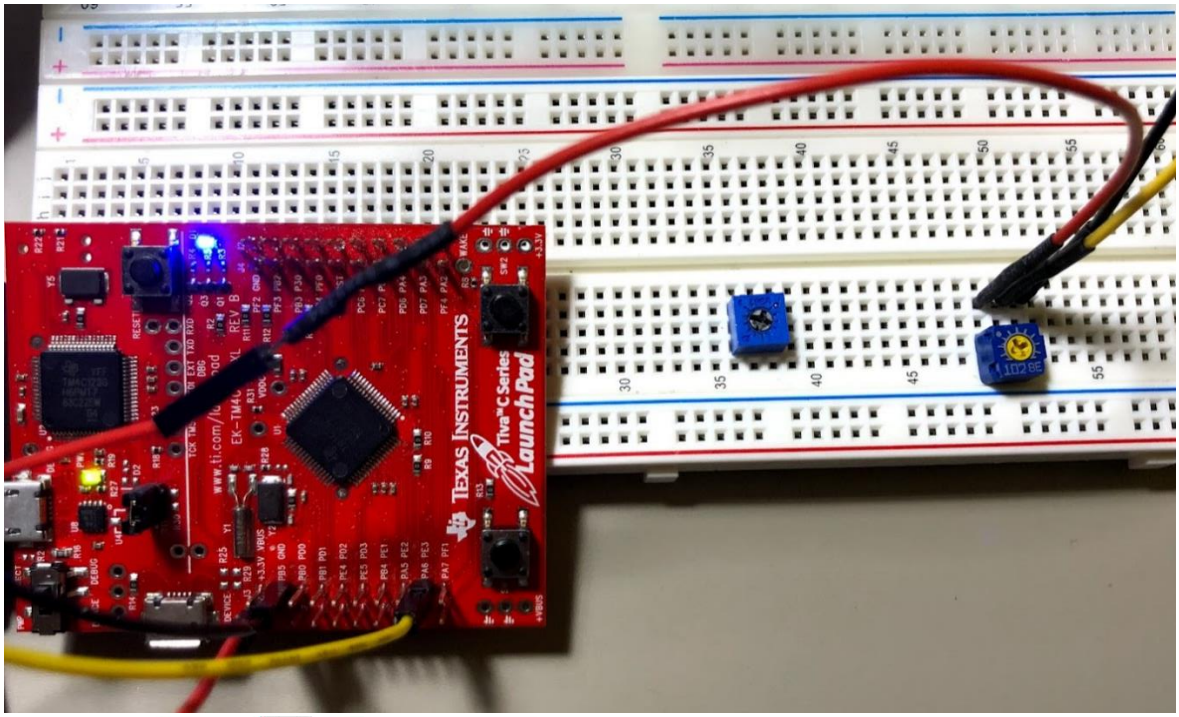

2) Potentiometer is turned to the left. We can see the transition happen because the ADC values drop significantly (but the DC values remain the same because the switch has not been pressed yet).

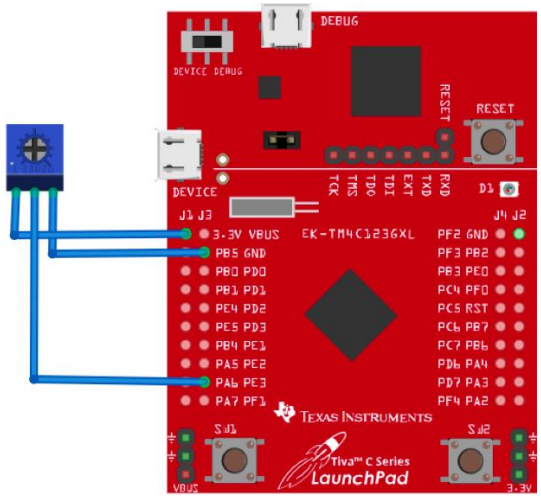


3) The switch has been pressed in this step. We can see the transition happen because the DC values adjust itself to match the potentiometer change that we did in step 2 above

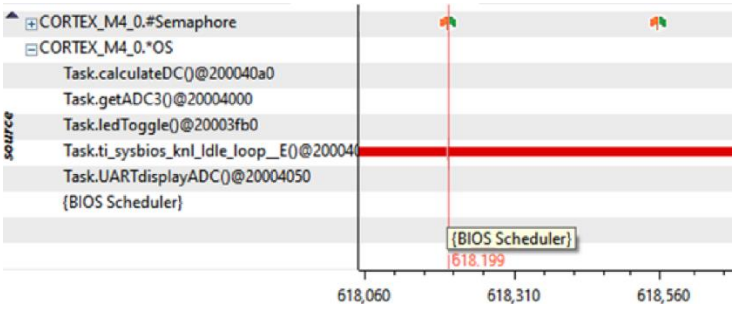
```
ADC1 CH3:  439,  DC:  30/30
ADC1 CH3:  432,  DC:  30/30
ADC1 CH3:  492,  DC:  30/30
ADC1 CH3:  494,  DC:  7/30
ADC1 CH3:  500,  DC:  7/30
ADC1 CH3:  460,  DC:  7/30
ADC1 CH3:  472,  DC:  7/30
ADC1 CH3:  476,  DC:  7/30
ADC1 CH3:  484,  DC:  7/30
ADC1 CH3:  504,  DC:  7/30
ADC1 CH3:  520,  DC:  7/30
ADC1 CH3:  536,  DC:  7/30
```

**PHOTOS OF THE BOARD:**



**EXECUTION ANALYSIS:**




**CONCLUSIONS:**

This midterm project was conducted on CCS with the TIVA C TM4C123GH6PM's real time operating system and a potentiometer. I used hardware interrupts to carry out the tasks of reading ADC, displaying to UART, and changing duty cycle values with the potentiometer & switch. Every 10$^{th}$ instance, ADC was read, every 20$^{th}$ instance, UART displayed the current ADC & duty cycle values, and every 30$^{th}$ instance, the switch was read. If I turned the potentiometer, the ADC values would change, however the duty cycle values would remain the same (as well as the LED light's flashing speed). Only when I hit switch 1 on the TIVA C board, would the duty cycle values update in real time on the UART display, and the LED changes its flashing speed to match the new duty cycle value.

**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();

   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_MAI
N);

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

	ui32Period = (SysCtlClockGet() / 20);				//
period = CPU clk div 20 (50ms)
	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

}

// 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;                                    // keep track
of #toggles

                Log_info1("LED TOGGLED [%u] TIMES",i16ToggleCount);     // send toggle
count to UIA
       }

}


//---------------------------------------------------------------------------
// 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);
        }
    }
}


//*****************************************************************************
// 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, DC: %d/30\n", ADC3out, DC);
    }
}
```

**TIVAC_TIRTOS.CFG**

```
/*
 *  ======== 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);
```