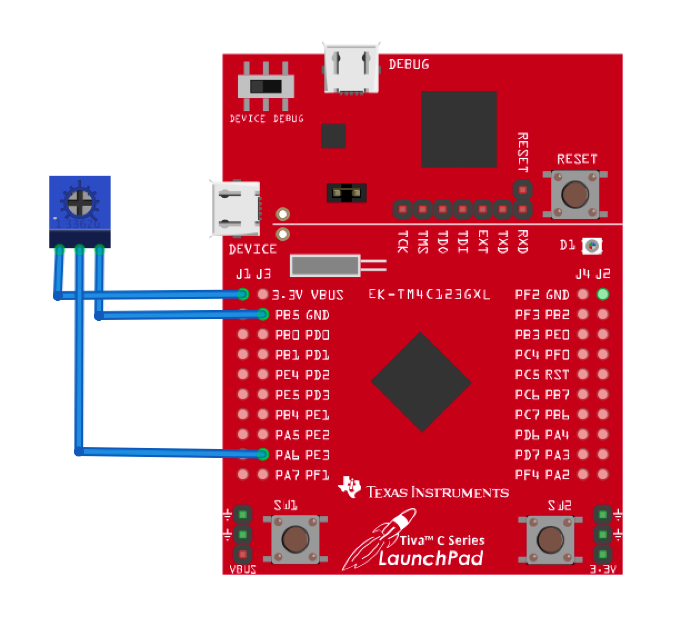
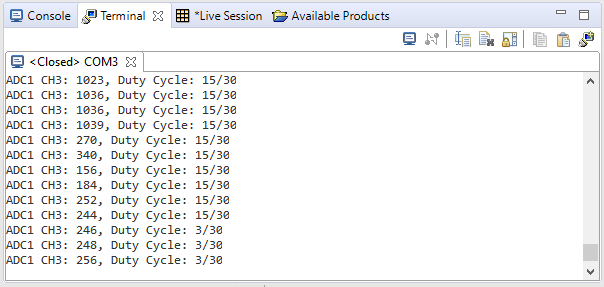
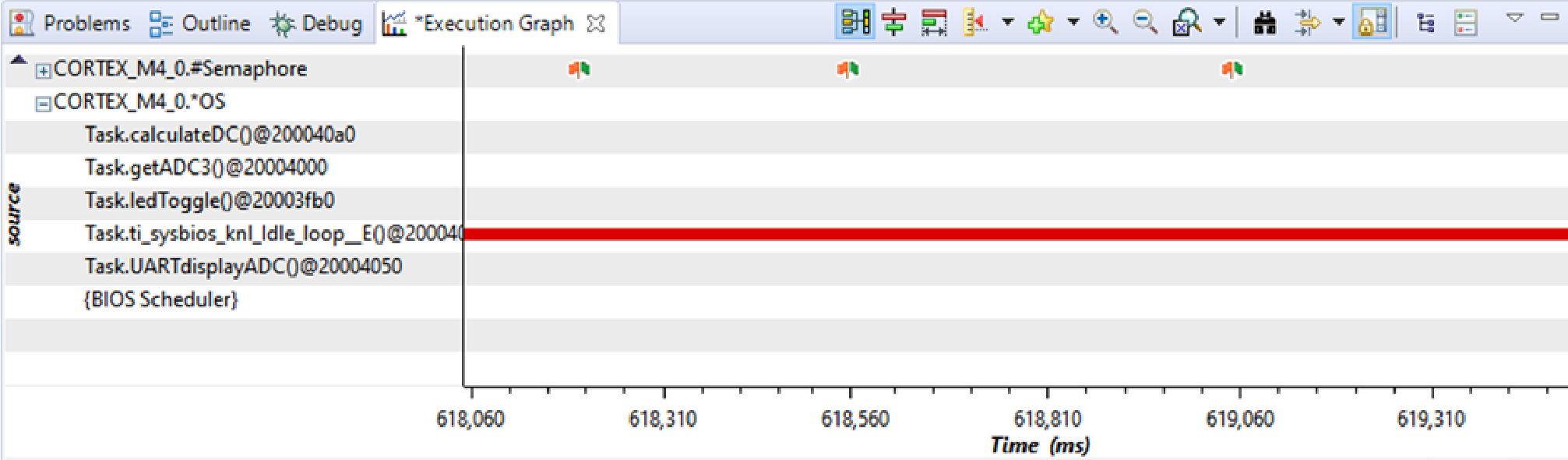
**Youtube Link:**

<https://youtu.be/gwN4If9qgz0>

**Modified Schematic:**

****

**UART: Execution Graph:**



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

**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

// 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];

}

}

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

**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);

}

}

**------------------------------------------------------------------------------------  
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);

}

}

}

**------------------------------------------------------------------------------------  
Complete Code:**

//----------------------------------------

// BIOS header files

//----------------------------------------

#include <xdc/std.h>

#include <ti/sysbios/BIOS.h>

#include <xdc/runtime/Log.h>

#include <xdc/cfg/global.h>

//------------------------------------------

// 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\_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

}

}

//---------------------------------------------------------------------------

// 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, Duty Cycle: %d/30\n", ADC3out, DC);

}

}

/\*

\* 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);