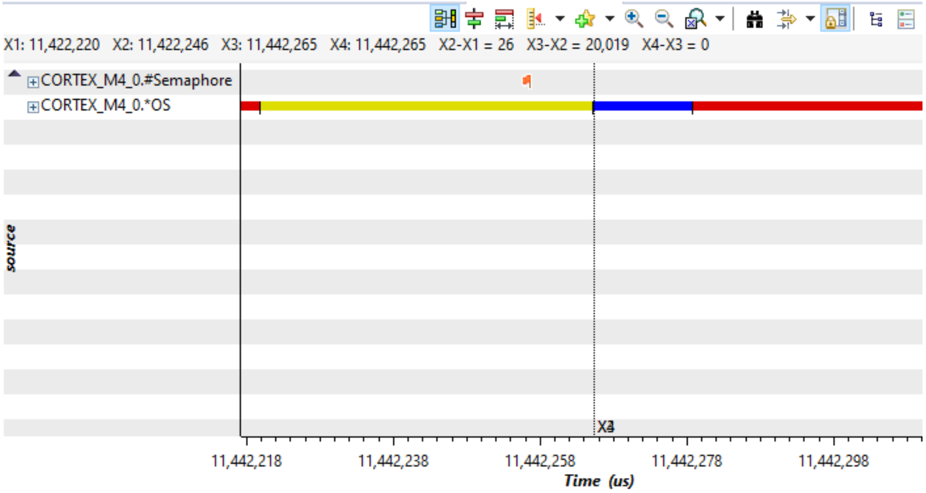
Jett Guerrero
Github root directory: https://github.com/guerrj1/Advanced-Embedded-Systems

Date Submitted: 12/13/2019

Youtube Link: https://youtu.be/DixUOEtXQ3w



X1: 11,422,220   X2: 11,422,246   X3: 11,442,265   X4: 11,442,265   X2-X1 = 26   X3-X2 = 20,019   X4-X3 = 0

Execution Graph

Modified Code:

```
//TIRTOS Assignment

//----------------------------------------
// 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 "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "driverlib/pwm.h"
```

```
#define PWM_FREQUENCY 55     //frequency for PWM

//----------------------------------------
// Prototypes
//----------------------------------------
void HARDWAREinit(void);
void Timer_ISR(void);
void ADCinit();
void ADCvalue (void);
void CONSOLEinit(void);
void ADCuart (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[4];

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


void main(void)
{
       HARDWAREinit();
       ADCinit();
       CONSOLEinit();
       BIOS_start();

}

//---------------------------------------------------------------------------
// HARDWAREinit()
//
// inits GPIO pins for toggling the LED
//---------------------------------------------------------------------------
void HARDWAREinit(void)
{
       uint32_t ui32Period;
       uint32_t ui32Load;
       uint32_t ui32PWMClock;

       //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);
       SysCtlPWMClockSet(SYSCTL_PWMDIV_64);

       //PWM initialization
```

```
	SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

	ui32PWMClock = SysCtlClockGet() / 64;
	ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
	GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
	GPIOPinConfigure(GPIO_PD0_M1PWM0);
	PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
	PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);
	PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
	PWMGenEnable(PWM1_BASE, PWM_GEN_0);

	// 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);

	//led config
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4);

	//buttion config
	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 config
	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
	TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
	ui32Period = (SysCtlClockGet() / 500);
	TimerLoadSet(TIMER2_BASE, TIMER_A, ui32Period);
	TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
	TimerEnable(TIMER2_BASE, TIMER_A);

}

void CONSOLEinit(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);
}

void ADCinit()
{
    //peripheral settings
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    //GPIO initialization
    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);    //sets PE3 for ADC

    //
    // 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(ADC0_BASE, 1, 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(ADC0_BASE, 1, 0, ADC_CTL_CH3);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH3);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH3);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_CH3 | ADC_CTL_IE | ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);

}

//---------------------------------------------------------------------------
// 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);
```

```
        //when button is pressed activate the led
        if (GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_0))
        {
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4);
        }

        //else led off
        else{
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);

        }

        //tasks calling
        if(i16InstanceCount == 10)
        {
                Semaphore_post(ADC3);
        }

        else if (i16InstanceCount == 20)
        {
                Semaphore_post(UART2);
        }

        else if(i16InstanceCount == 30)
        {
                Semaphore_post(SWRead1);
                i16InstanceCount = 0;
        }

        i16InstanceCount++;
}

//---------------------------------------------------------------------------
// Read Switch
//
// Grabs the value of the ADC and switches the PWM
//---------------------------------------------------------------------------
void SW_read(void)
{
        while(1)
        {
                if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)==0x00) {
                        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ADCoutput);
                }

                Semaphore_pend(SWRead1, BIOS_WAIT_FOREVER);
        }
}

//---------------------------------------------------------------------------
// ADC1 from CH3
//
// Converts and grabs values for the ADC
//---------------------------------------------------------------------------
void ADCvalue (void) {

        while(1) {
```

```
        ADCIntClear(ADC0_BASE, 1);
        //
        // Trigger the ADC conversion.
        //
        ADCProcessorTrigger(ADC0_BASE, 1);

        //
        // Wait for conversion to be completed.
        //
        while(!ADCIntStatus(ADC0_BASE, 1, false))
        {
        }

        //
        // Read ADC Value.
        //
        ADCSequenceDataGet(ADC0_BASE, 1, ADCValues);
        ADCoutput = (ADCValues[0] + ADCValues[1] + ADCValues[2] + ADCValues[3])/4;
        Semaphore_pend(ADC3, BIOS_WAIT_FOREVER);
    }

}

//---------------------------------------------------------------------------
// UART
//
// Displays the ADC as projected from the potentiometer
//---------------------------------------------------------------------------
void ADCuart (void)
{
    while(1)
    {
        UARTprintf("ADC Value: %d\n", ADCoutput);
        Semaphore_pend(UART2, BIOS_WAIT_FOREVER);
    }
}
```

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