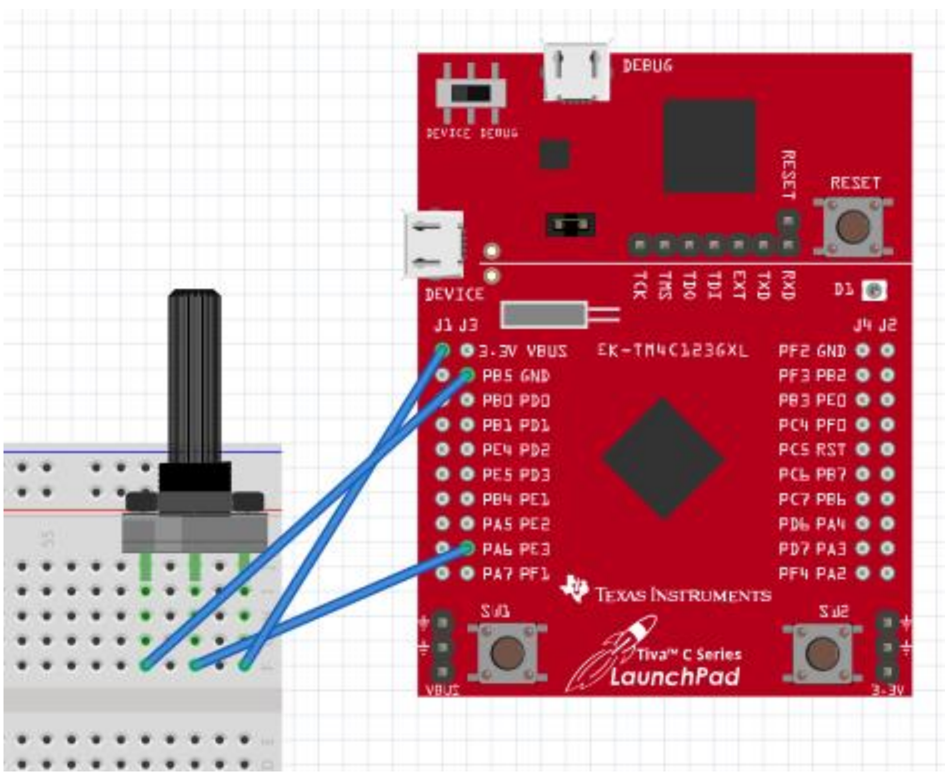
TITLE: TIVAC-TIRTOS-ASSIGNMENT

Midterm by Ron Joshua Recrio

GOAL:

- Get values using ADC on a potentiometer
- Print ADC values to UART
- LED on a PWM
- Switch read task to change duty cycle of PWM

Schematics:



DETAILED IMPLEMENTATION:

The first thing implemented is initializing the clock and all of the hardware components such as LED, switch, pwm, and the timer. Then the ADC and UART are initialized. Once those are made there are multiple ways to implement the following steps. The way chosen was to start with the configuration file and use the GUI to create a timer hardware interrupt. This will allows us to precisely determine which instance we are at in order to do the tasks 10 instances apart. The next thing is to create the tasks. The tasks are, read ADC, UART the adc values, and read the switch. Each of these tasks will be using a semaphore to synchronize. After setting the configuration, each task is made. The tasks are made to be as simple as possible (LED toggle only toggles LED). In the Timer interrupt, we turn on the LED based off of the duty cycle global variable. This variable is changed when the switch is pressed. This will read the ADC value and create a duty cycle value for the PWM. In this case we are simply using the timer as a toggle for our presumed PWM.

CODE:

```
//----------------------------------------
// 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"
```

```c
#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"
#include "driverlib/pwm.h"

// Define
#define PWM_FREQUENCY 55

//----------------------------------------
// Prototypes
//----------------------------------------
void hardware_init(void);
void Timer_ISR(void);
void initADC( );
void getADC3(void);
void InitConsole(void);
void UARTdisplayADC(void);


//---------------------------------------
// Globals
//---------------------------------------
volatile int16_t i16InstanceCount = 0;
volatile int16_t dutyCycle = 0;
// 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 adcValue ;



//---------------------------------------------------------------------------
// 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, ui32PWMClock, ui32Load;

	//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);
	SysCtlPWMClockSet(SYSCTL_PWMDIV_64);


	// Enable PWM and GPIOF
	SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
	// Configure PWM to PF1
	GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
	GPIOPinConfigure(GPIO_PD0_M1PWM0);

	// Configure SW1
	GPIODirModeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_DIR_MODE_IN);
	GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

	// Initialize
	ui32PWMClock = SysCtlClockGet() / 64;
	ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
	PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
	PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);

	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, dutyCycle);
	PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
	PWMGenEnable(PWM1_BASE, PWM_GEN_0);


	// 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() / 500);					//
period = CPU clk div 500 (1ms)
	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

}


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


//---------------------------------------------------------------------------
// 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); // reset

    if (GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_0))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 4);
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
    }

    // Task 1
    if (i16InstanceCount == 10) {
        Semaphore_post(ADC3Sem);
    }
    // Task 2
    else if (i16InstanceCount == 20) {
        Semaphore_post(UARTSem);
    }
    // Task 3
    else if(i16InstanceCount == 30) {
        Semaphore_post(SW_ReadSem);
        i16InstanceCount = 0;
    }

    i16InstanceCount++;
}

void changeDutyCycle(void)
{
    while(1)
    {
        Semaphore_pend(SW_ReadSem, BIOS_WAIT_FOREVER);

        // From experimentation the max is ~2200 and min is ~150
        // Cut off at 2000 and 200 for clean easy access to extreme values.
```

```
        // Calculate a new duty cycle given adcValue
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4) == 0x00)
        {
            dutyCycle = adcValue;
        }
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, dutyCycle);
    }
}


//
// Initializes ADC1 for use
//
void initADC() {
    // The ADC3 peripheral must be enabled for use.
                //
                SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);
                SysCtlDelay(3);
                SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
                SysCtlDelay(3);

                GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);    //Configure ADC pin:
PE0



                //
                // 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);

}

//
// Gets value of ADC1 CH 3
//
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);
        adcValue = ADCValues[0];
    }

}

void InitConsole(void)
{
    //
    // Enable GPIO port A which is used for UART0 pins.
    //
    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.
    //
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
```

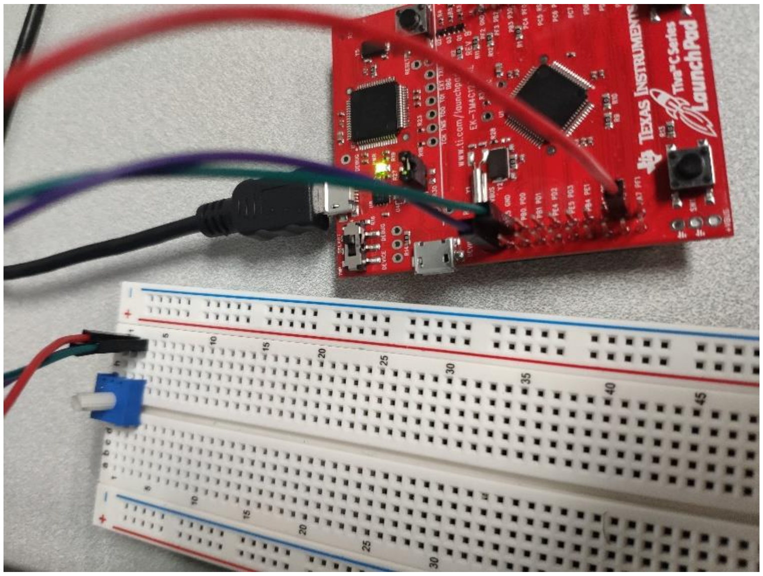
```
    //
    // Enable UART0 so that we can configure the clock.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

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

    //
    // Select the alternate (UART) function for these pins.
    //
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

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

//
// Displays ADC values
//
void UARTdisplayADC(void)
{
    while(1)
    {
        Semaphore_pend(UARTSem, BIOS_WAIT_FOREVER);
        UARTprintf("ADC: %d  Duty Cycle: %d%% \n", adcValue, (int)(((float)dutyCycle
/ 2500.0)*100.0));
    }
}

}
```

Video Link:

https://youtu.be/Zjq5c8u9RKY

Screenshots:

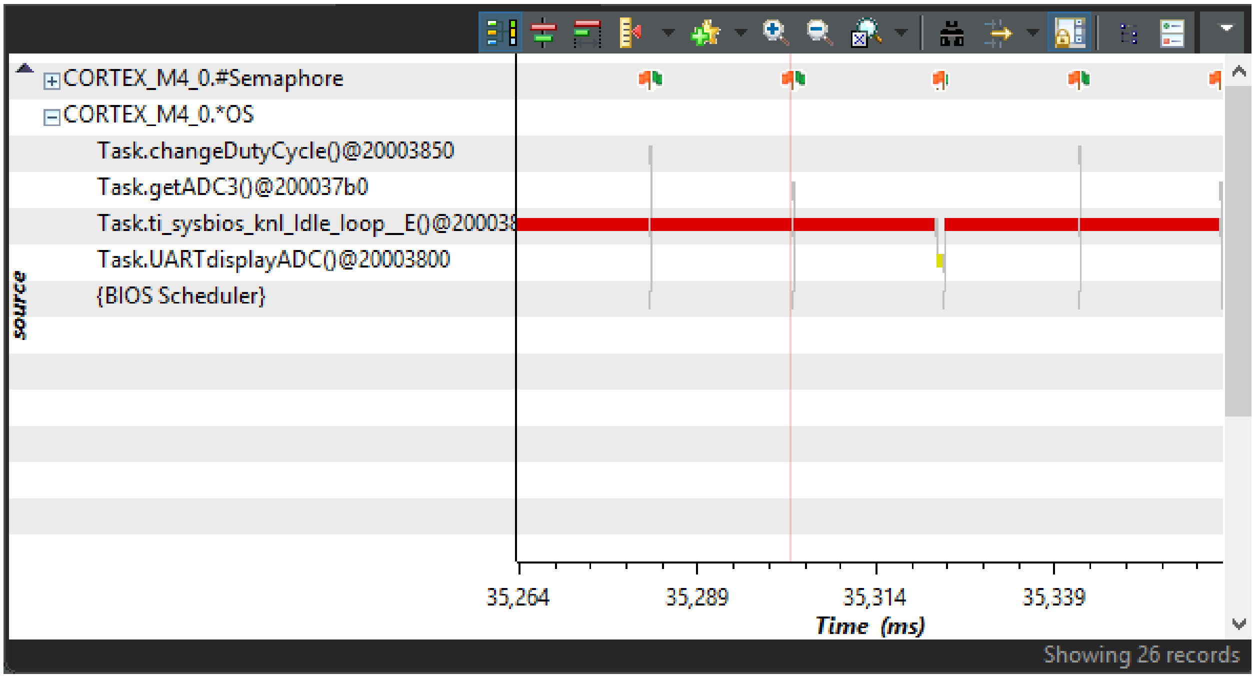
Circuit Implementation:



Output:

```
ADC: 2112  Duty Cycle: 83%
ADC: 2106  Duty Cycle: 83%
ADC: 2098  Duty Cycle: 83%
ADC: 2104  Duty Cycle: 83%
ADC: 2106  Duty Cycle: 83%
ADC: 2098  Duty Cycle: 83%
ADC: 2077  Duty Cycle: 83%
```

RTOS Analyzer:



Conclusions:

The project was successful. The three tasks were implemented accordingly. The only issue may be that the board may be running too slowly and could be increased by decreasing the ui32period. I am still a little confused on how to calculate the specific duty cycles for the PWM. I do not understand how to calculate the specific values to determine the high's and low's of the duty cycle.