TITLE: TI-RTOS Assignment

GOAL: The goal of this assignment is to create 3 tasks: 1. ADC task, 2. UART task, and 3. a switch read task. Each task will be executed in order every 30 ms.

- A HWI timer runs every 1 ms. The timer is incremented every 1 ms and will then be reset back to 0 once it reaches 30 ms (or the 30th instance).
- At every **10th instance**, the ADC values are updated and read (ADC task).
- At every **20th instance**, the current ADC value read from the previous task is displayed onto the terminal (UART task)
- At every **30th instance**, the state of a switch is read (either SW1 or SW2):
  - if the switch **was pressed**, update and set a new pulse width value (duty cycle).
  - if the switch **was not pressed**, do not update/change the pulse width value (duty cycle).

DELIVERABLES:

Tasks (all are completed):

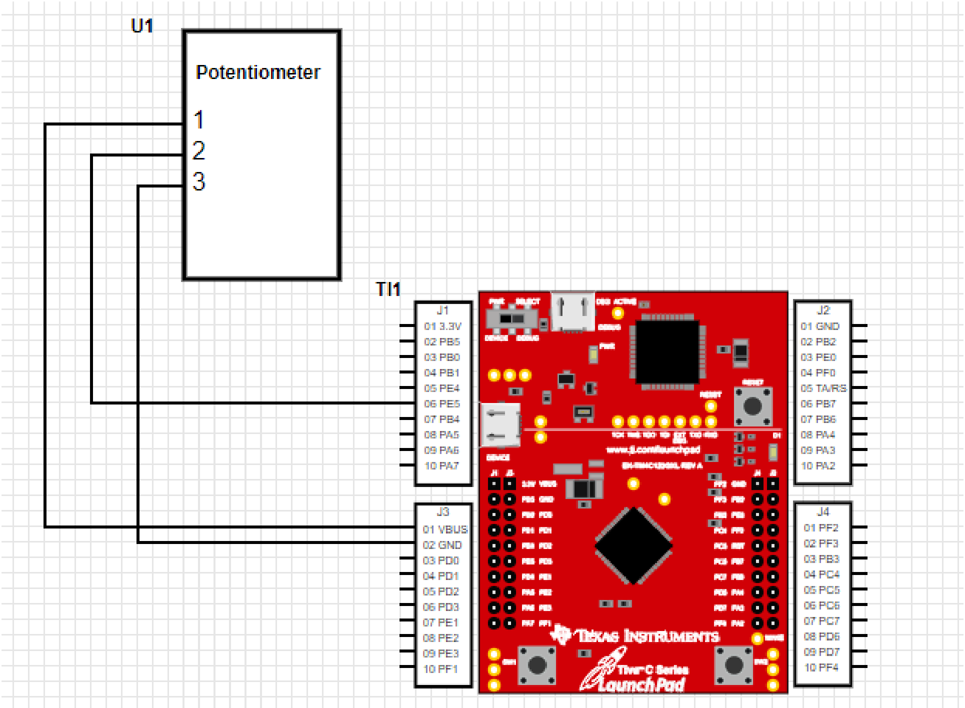
- *ADC Task* – read the value of the ADC coming from the output of the potentiometer.
- *UART Task* – display the recently obtained ADC value onto a local terminal
- *Switch Read Task* – read the state of a switch. If the switch was pressed, update the duty cycle of the PWM signal. Else, do not update the duty cycle of the PWM.

COMPONENTS:

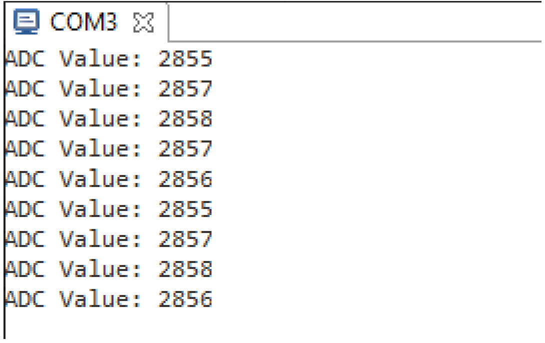
- TIVAC TM4C (TI-RTOS)
- Potentiometer: used to read the ADC values from
- Breadboard and jumper wires

SCHEMATICS and PICTURES:

Schematic Overview:

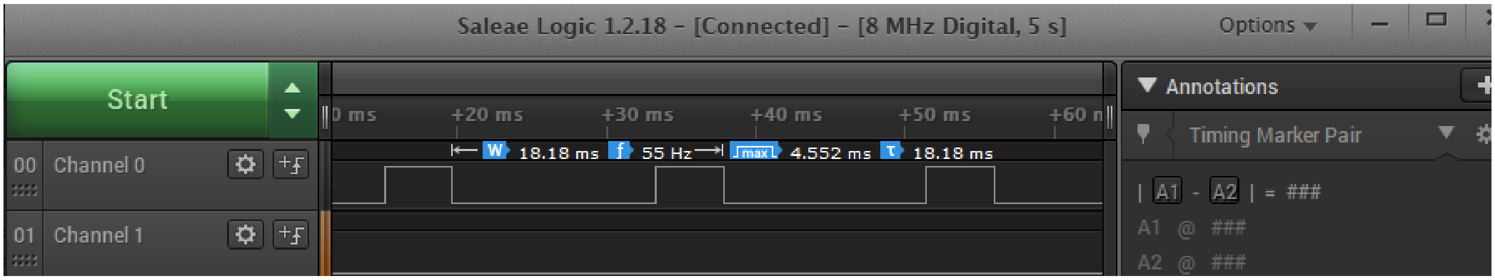


Samples of ADC Values (read from potentiometer):



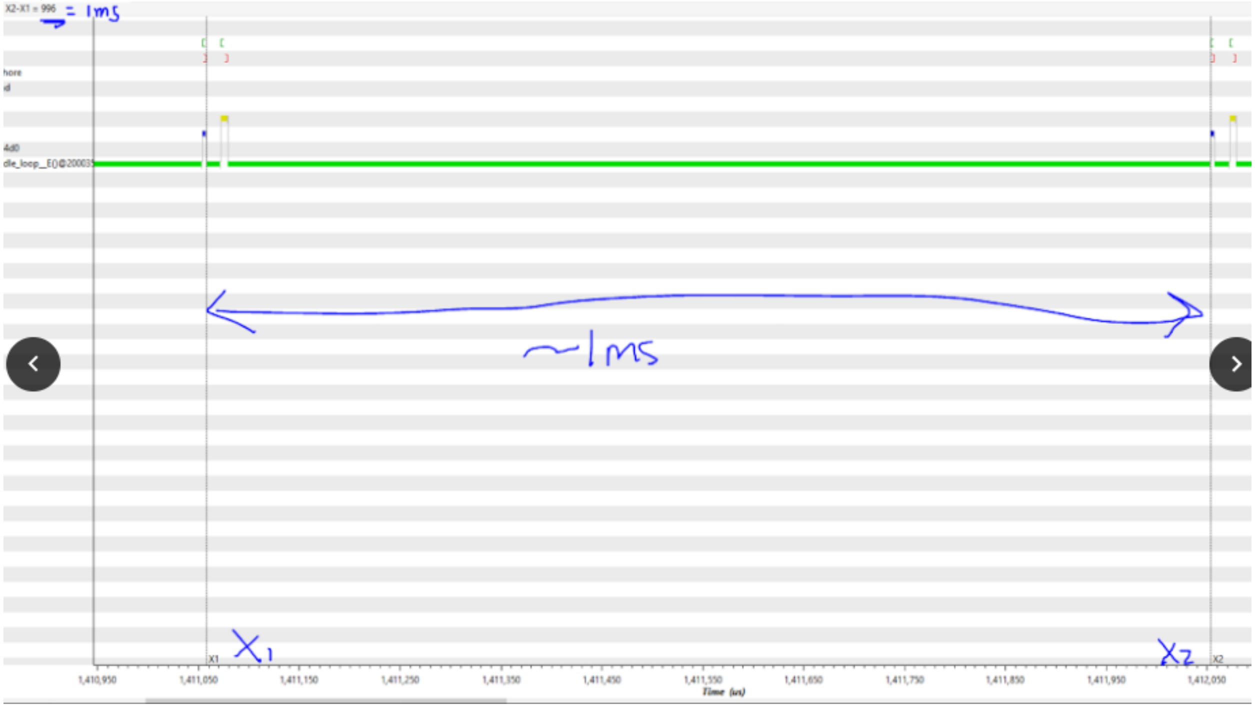
```
ADC Value: 2855
ADC Value: 2857
ADC Value: 2858
ADC Value: 2857
ADC Value: 2856
ADC Value: 2855
ADC Value: 2857
ADC Value: 2858
ADC Value: 2856
```

Sampled PWM Signal Showing the Pulse Width and Duty Cycle:



W 18.18 ms f 55 Hz ⌐max 4.552 ms τ 18.18 ms

Execution Graph (showing the HWI timer and the switch read task, note: timer executes every 1 ms):



Graph is showing it in microseconds. The value above (~996us is around 1ms)

IIMPLEMENTATION:

<u>CODE</u>:

```
/*
 * Copyright (c) 2015, 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.
// */
//



//----------------------------------------
// 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 "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "driverlib/pwm.h"

//----------------------------------------
// Function Prototypes
//----------------------------------------
void hardware_init(void);
void HWI_Timer(void);
void adcTaskFxn(void);
void swReadTaskFxn(void);
void uartTaskFxn(void);
void InitConsole(void);

//---------------------------------------
```

```
// Define stmts and Global Variables
//-------------------------------------

#define PWM_FREQUENCY 55    // PWM frequency set to 55Hz

volatile int16_t counter;
uint32_t ui32ADC0Value[4];
uint32_t ui32ADCAvg;
uint32_t ui32Adjust;
volatile uint32_t ui32Load;
volatile uint32_t ui32PWMClock;




/*main function*/

int main(void)
{

   hardware_init(); // call function to initialize the hardware

   /* Start BIOS */
   BIOS_start();

}


void InitConsole(void){
   //Enable GPIO port A for UART pins
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

   //Configure UART pins for Rx and Tx
   GPIOPinConfigure(GPIO_PA0_U0RX);
   GPIOPinConfigure(GPIO_PA1_U0TX);

   //Enable UART0.
   SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

   //Use the internal 16MHz oscillator
   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
   UARTStdioConfig(0, 115200, 16000000);
}


//-----------------------------------------------------------------------
// hardware_init()
//
// inits GPIO pins for toggling the LED
//-----------------------------------------------------------------------
```

```
void hardware_init(void)
{
   uint32_t ui32Period;

   counter = 0; // initialize counter to 0



   //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);

   SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

   // ADD Tiva-C GPIO setup - enables port, sets pins 1-3 (RGB) pins for output
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); // enable analog input 3 (PE0)

   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
   GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
   GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0); // use PE0 (AIN3 - channel 3) for
potentiometer

   GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);


   //initialize PWM
   ui32PWMClock = SysCtlClockGet() / 64;
   ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;

   GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0); //PD0 PWM pin
   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);


   //initialize ADC
   SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
   ADCHardwareOversampleConfigure(ADC0_BASE, 64);
   ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

   // using channel 3 for the ADC samples
   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);

    // Initialize Timer 2 for the HWI
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
    TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);

    ui32Period = (SysCtlClockGet() / 500);  // period is around 1ms
    TimerLoadSet(TIMER2_BASE, TIMER_A, ui32Period);

    TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);

    TimerEnable(TIMER2_BASE, TIMER_A);


    // call function to initialize UART
    InitConsole();

}



void adcTaskFxn(void){
// read ADC value, store into variable,
// set the pulse width according to the ADC value
    while(1){

        ADCIntClear(ADC0_BASE, 1);
        ADCProcessorTrigger(ADC0_BASE, 1);

        while (!ADCIntStatus(ADC0_BASE, 1, false)) {}

        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

        ui32ADCAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
        ui32Adjust = ui32ADCAvg; // store ADC avg value into the ui32Adjust variable

        Semaphore_pend (sem_ADC, BIOS_WAIT_FOREVER);
    }
}


void uartTaskFxn(void){
// display the current ADC value on terminal
    while(1){

        UARTprintf("ADC Value: %d\n", ui32Adjust);
        Semaphore_pend (sem_UART, BIOS_WAIT_FOREVER);
    }
}


void swReadTaskFxn(void){
// when the switch is pressed, the duty cycle of the PWM
// changes according to the ADC value
    while(1){
        // if switch 1 is pressed down...
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)==0x00)
        {
            // set and adjust the width of the PWM using the ui32Adjust value
            PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui32Adjust);
        }
        Semaphore_pend (sem_swRead, BIOS_WAIT_FOREVER);

    }
}


void HWI_Timer(void){
// HWI executes every 1ms
// at every 10th instance, ADC task is executed
// at every 20th instance, UART task is executed
// at every 30th instance, swRead task is executed and reset the counter
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT); // clear Timer interrupt
    counter++; // increment counter every time HWI occurs


    // every time the pulse is high, turn on LED, else, turn off LED
    // the duration of the time that is high depends on the pulse width value
    // that was set in the sw read function
    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_2, 0);
    }


    // execute ADC
    if (counter == 10){
        Semaphore_post (sem_ADC);
    }

    // execute UART and display current ADC value
    else if (counter == 20){
        Semaphore_post (sem_UART);
    }

    // execute sw Read task and read if the switch is pressed or not
    // if pressed, change the pwm pulse width according to the ADC value
    else if (counter == 30){
        Semaphore_post (sem_swRead);
        counter = 0; // reset counter
    }

}
```