CPE 403 ADV EMB SYS DES F 2019

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:

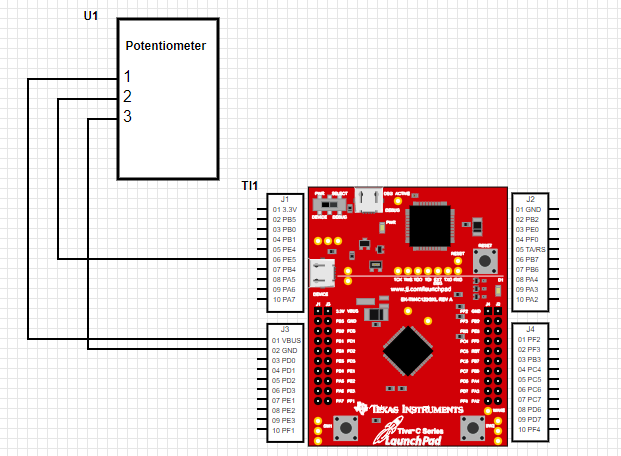
Tasks (all are completed):

* *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.

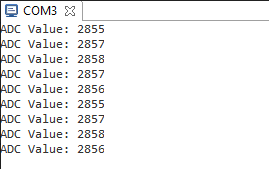
COMPONENTS:

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

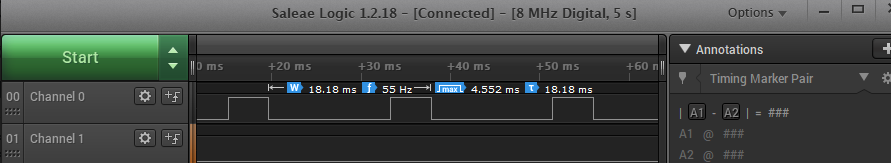
SCHEMATICS and PICTURES:

Schematic Overview:

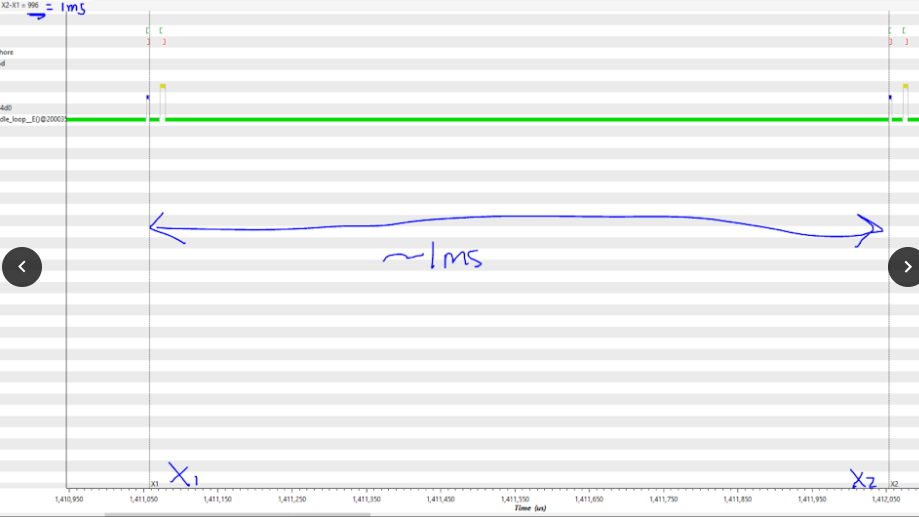
Samples of ADC Values (read from potentiometer):



Sampled PWM Signal Showing the Pulse Width and Duty Cycle:



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:

CODE:

/\*

\* 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

}

}