CPE403 – Advanced Embedded Systems

Design Assignment 03

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Do Le

Email: led2@unlv.nevada.edu

Github Repository link (root): https://github.com/DoVietLe/AES

Youtube Playlist link (root):
https://www.youtube.com/playlist?list=PLFfzhLPj7fvOz1lm2Vd9DevkHetoyvRQ6

1. Code for Tasks. for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only. Use separate page for each task.

```
#include <file.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

/* XDCtools Header files */
#include <xdc/std.h>
#include <xdc/cfg/global.h>

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Swi.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/PWM.h>

/* TI-RTOS Header files */
#include <ti/drivers/GPIO.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SDSPI.h>
// #include <ti/drivers/SPI.h>
#include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>

/* Board Header file */
#include "Board.h"
```

```
/* Extra Header files*/
#include "driverlib/adc.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"
#include "UARTUtils.h"

#define TASKSTACKSIZE   512
#define BUTTON GPIO_PIN_4

Task_Struct task0Struct;
Task_Struct task1Struct;
Task_Struct task2Struct;
Char task0Stack[TASKSTACKSIZE];
Char task1Stack[TASKSTACKSIZE];
Char task2Stack[TASKSTACKSIZE];

uint8_t timerCount = 0;
uint32_t adcVal[1];

// Initializes ADC since TIRTOS doesn't have ADC APIs.
void initADC() {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_3);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH4|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3);
}


/*
 *  ======== heartBeatFxn ========
 *  Toggle the Board_LED1. The Task_sleep is determined by arg0 which
 *  is configured for the heartBeat Task instance.
 */
//*
Void heartBeatFxn(UArg arg0, UArg arg1)
{
    while (1) {
        Task_sleep((UInt)arg0);
        GPIO_toggle(Board_LED2);
    }
}
//*/
/*
 *  ======== UARTMon_taskFxn ========
 *
 *
 */
Void UARTMon_taskFxn(UArg arg0, UArg arg1)
{
    while (1) {
       Semaphore_pend(PostVal, BIOS_WAIT_FOREVER);
       printf("adcVal: %d\n\r", adcVal[0]);
    }
}

/*
 *  ======== PWMSet Task ========
 * This task sets up the PWM then continuously pends for a semaphore.
 * When semaphore PostPWM is > 0, it checks to see if button0 is pressed, and updates
PWM duty cycle if so.
```

```
 */
Void PWMSet(UArg arg0, UArg arg1)
{
    PWM_Handle pwm1;
    PWM_Params params;
    uint16_t   pwmPeriod = 1000;      // Period and duty in microseconds
    uint16_t   duty = 0;

    PWM_Params_init(&params);
    params.period = pwmPeriod;
    pwm1 = PWM_open(Board_PWM0, &params);
    PWM_setDuty(pwm1, duty);        // Initializes duty as 0.

    while (1) {
       Semaphore_pend(PostPWM, BIOS_WAIT_FOREVER);
       if (!GPIO_read(Board_BUTTON0)) {
           duty = (adcVal[0]*pwmPeriod)/4096;
           PWM_setDuty(pwm1, duty);
       }
    }
}

/*
 *  ======== ADC Task ========
 * This task triggers a read on the ADC and stores the value.
 *
 */
void ADCUpdate() {
    while (1) {
        Semaphore_pend(PostADC, BIOS_WAIT_FOREVER);
        ADCIntClear(ADC0_BASE, 3);
        ADCProcessorTrigger(ADC0_BASE, 3);
        while(!ADCIntStatus(ADC0_BASE, 3, false));
        ADCSequenceDataGet(ADC0_BASE, 3, adcVal);
    }
}

/*
 *  ======== ISR_timer0 ========
 * This ISR is used to control which task runs
 * every 5ms.
 */
//*
Void ISR_timer0()
{
    timerCount++;
    if (timerCount == 5) {
        Semaphore_post(PostADC);
    } else if (timerCount == 10) {
        Semaphore_post(PostVal);
    } else if (timerCount >= 15) {
        Semaphore_post(PostPWM);
        timerCount = 0;
    }
}


/*
 *  ======== main ========
 */
int main(void)
{
    Task_Params taskParams;
```

```
/* Call board init functions */
Board_initGeneral();
Board_initGPIO();
// Board_initI2C();
// Board_initSDSPI();
// Board_initSPI();
Board_initUART();
// Board_initUSB(Board_USBDEVICE);
// Board_initWatchdog();
// Board_initWiFi();
initADC();
Board_initPWM();

// PWM Task
Task_Params_init(&taskParams);
taskParams.arg0 = 1000;
taskParams.stackSize = TASKSTACKSIZE;
taskParams.stack = &task0Stack;
Task_construct(&task0Struct, (Task_FuncPtr)PWMSet, &taskParams, NULL);

// ADC Task
Task_Params_init(&taskParams);
taskParams.arg0 = 1000;
taskParams.stackSize = TASKSTACKSIZE;
taskParams.stack = &task1Stack;
Task_construct(&task1Struct, (Task_FuncPtr)ADCUpdate, &taskParams, NULL);

// Heartbeat Task
Task_Params_init(&taskParams);
taskParams.arg0 = 1000;
taskParams.stackSize = TASKSTACKSIZE;
taskParams.stack = &task2Stack;
Task_construct(&task2Struct, (Task_FuncPtr)heartBeatFxn, &taskParams, NULL);

/* Turn on user LED  */
GPIO_write(Board_LED2, Board_LED_ON);

/*
 *  Add the UART device to the system.
 *  All UART peripherals must be setup and the module must be initialized
 *  before opening.  This is done by Board_initUART().  The functions used
 *  are implemented in UARTUtils.c.
 */
add_device("UART", _MSA, UARTUtils_deviceopen,
           UARTUtils_deviceclose, UARTUtils_deviceread,
           UARTUtils_devicewrite, UARTUtils_devicelseek,
           UARTUtils_deviceunlink, UARTUtils_devicerename);

/* Open UART0 for writing to stdout and set buffer */
freopen("UART:0", "w", stdout);
setvbuf(stdout, NULL, _IOLBF, 128);

/*
 *  Initialize UART port 0 used by SysCallback.  This and other SysCallback
 *  UART functions are implemented in UARTUtils.c. Calls to System_printf
 *  will go to UART0, the same as printf.
 */
UARTUtils_systemInit(0);

/* Start BIOS */
BIOS_start();
```

```
    return (0);
}
```

2. Block diagram and/or Schematics showing the components, pins used, and interface.

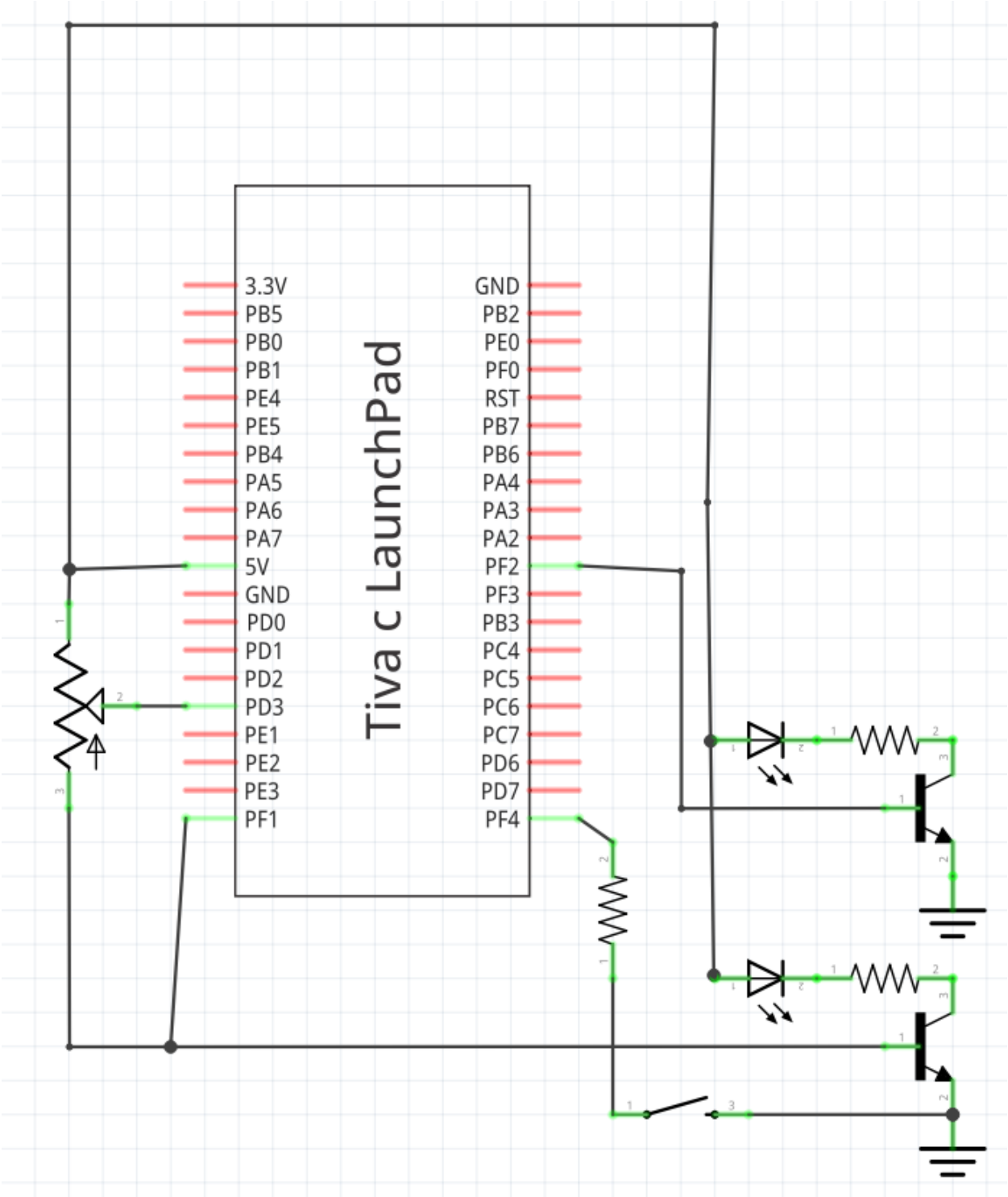

Figure 1: Block diagram

3. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.

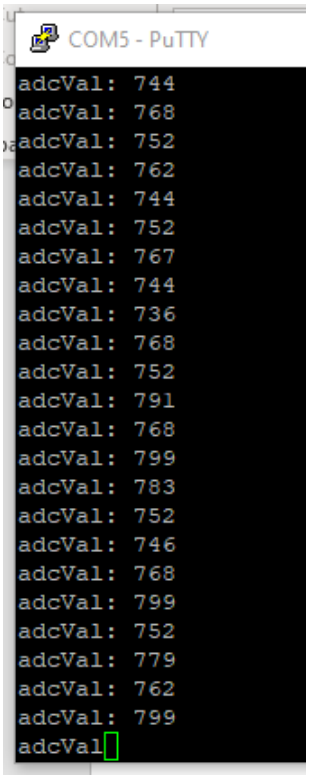

Figure 4: Read from potentiometer

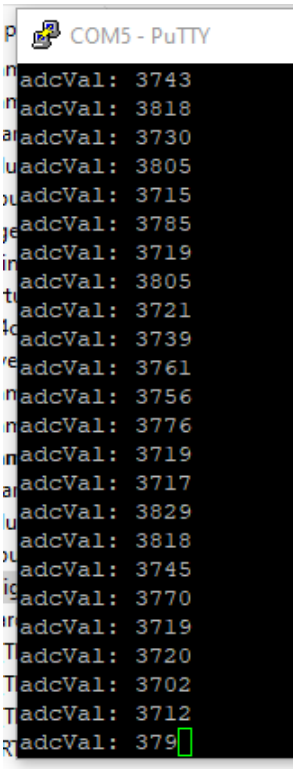

Figure 3: Another read from potentiometer

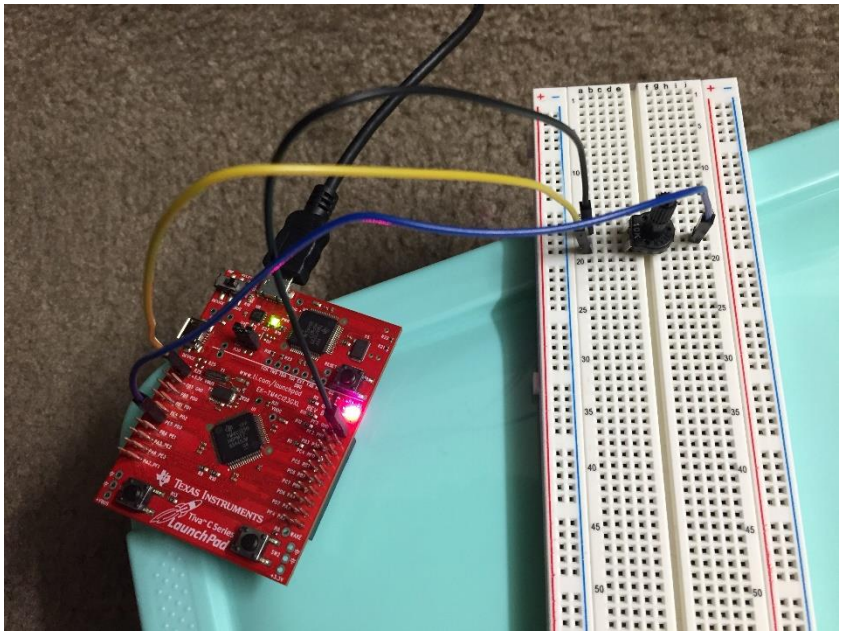

Figure 2: Image of setup

4. Declaration

I understand the Student Academic Misconduct Policy -
http://studentconduct.unlv.edu/misconduct/policy.html

"This assignment submission is my own, original work".
Do V. Le