CPE403 – Advanced Embedded Systems

Design Assignment #4

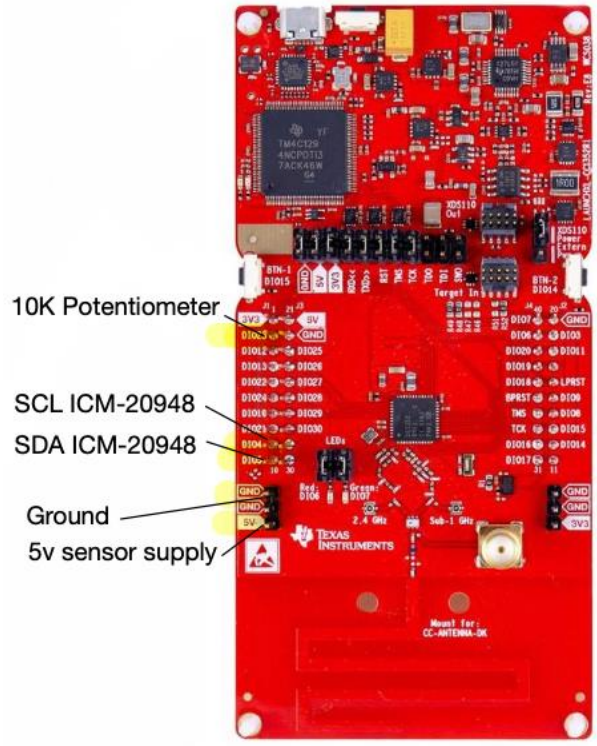
Name: Brian Wolak

Email: wolak@unlv.nevada.edu

GitHub Repository link (root): https://github.com/brianwolak/advanced_submissions

YouTube Playlist (root): https://youtube.com/playlist?list=PLl6a3M--0IcYnZIMGyucOLe8c-16wAClN

Goal of this assignment is to create five tasks, 1) ADC task (AD0), 2) UART display task, 3) Switch Read task, 4) I2C Sensor read task, and 5) Heartbeat function (RED LED/DIO6). The heartbeat function is performed throughout the execution of the program. Each task will be executed in order specified above every 20 ms. Connect a potentiometer to the ADC pin. Use ADC0 CH4. Also initialize a PWM signal to a LED (DIO7). Initial value of the PWM duty cycle is set to 0. Create a timer 0/1/2 timer Callback for every 1 ms, at 5th instance of timer Callback the task ADC is performed, at 10th instance of timer Callback the task UART displays the current value ADC in the terminal, at 15th instance of timer Callback the task Switch Read is performed to check the status of the SW1/SW2 to update the current value of duty cycle based on the ADC value, and at the 20th instance of timer Callback perform any I2C sensor read task. Note that the duty cycle of the PWM does not change unless the switch is pressed, even when the ADC value changes. However, the UART should display the dynamic and current active values of the ADC, and I2C sensor data. Use semaphores to switch between the tasks.



CC1352R1 Board Connections

C Code:

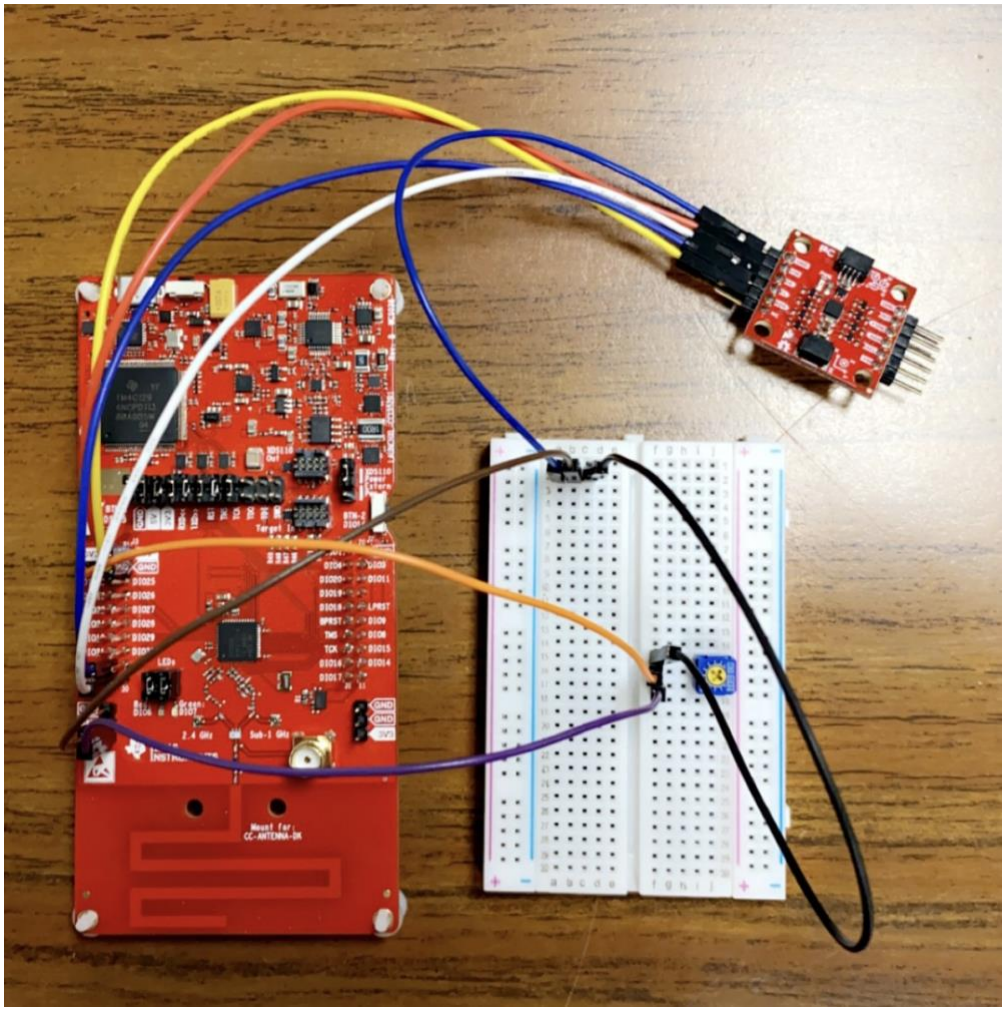
```
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include <stdint.h>
#include <stddef.h>
#include <unistd.h>
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/display/Display.h>
#include <ti/drivers/ADC.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/PWM.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/Board.h>
#include "ti_drivers_config.h"

#define stack1 512
#define stack2 640

// global variables
volatile uint32_t tickCount = 0;
volatile uint32_t sleepCount;
uint16_t ADCread;
uint16_t output;
uint16_t PWMperiod = 3000;
uint16_t dutyCycle = 0;
uint32_t PWMsave;
int button = 0;

Task_Struct task1Struct, task2Struct, task3Struct, task4Struct, task5Struct;
char t1Stack[stack1];
char t2Stack[stack1];
char t3Stack[stack1];
char t4Stack[stack2];
char t5Stack[stack1];
Semaphore_Struct semStruct;
Semaphore_Handle semHandle;

// push button 0 interrupt function to change PWM duty cycle
void pushButton(uint_least8_t index){
   button = 1;
}

// ACD read task function
void taskFunction1(UArg arg0, UArg arg1){
    ADC_Handle ADC;
    ADC_Params ADCparams;
    ADC_Params_init(&ADCparams);
    ADC = ADC_open(CONFIG_ADC_0, &ADCparams);

    while(1){
```

```
        System_printf("Task 1 executing..\n");

        if (Semaphore_getCount(semHandle) == 0) {
            System_printf("Task 1 denied access..\n");
        }

        /* Get access to resource */
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);

        /* Do work by waiting for 2 system ticks to pass */
        ADC_convert(ADC, &ADCread);
        ADCread = ADCread / 4;

        Semaphore_post(semHandle);
        Task_sleep(sleepCount);
    }
}

// ADC and I2C print function
void taskFunction2(UArg arg0, UArg arg1){
    Display_Handle hSerial = Display_open(Display_Type_UART, NULL);
    while(1){
        System_printf("Task 2 executing..\n");

        if (Semaphore_getCount(semHandle) == 0) {
            System_printf("Task 2 denied access..\n");
        }

        /* Get access to resource */
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);

        Display_printf(hSerial, 1, 0, "Potentiometer Value = %d", ADCread);
        Display_printf(hSerial, 1, 0, "PWM Value = %d", PWMsave);
        Display_printf(hSerial, 1, 0, "Temperature Value = %d\n", output);

        Semaphore_post(semHandle);

        Task_sleep(sleepCount);
    }
}

// PWM adjust task function
Void taskFunction3(UArg arg0, UArg arg1) {

    PWM_Handle pwm1 = NULL;
    PWM_Params params;
    PWM_init();

    PWM_Params_init(&params);
    params.dutyUnits = PWM_DUTY_US;
    params.dutyValue = 0;
    params.periodUnits = PWM_PERIOD_US;
    params.periodValue = PWMperiod;
    pwm1 = PWM_open(CONFIG_PWM_0, &params);
```

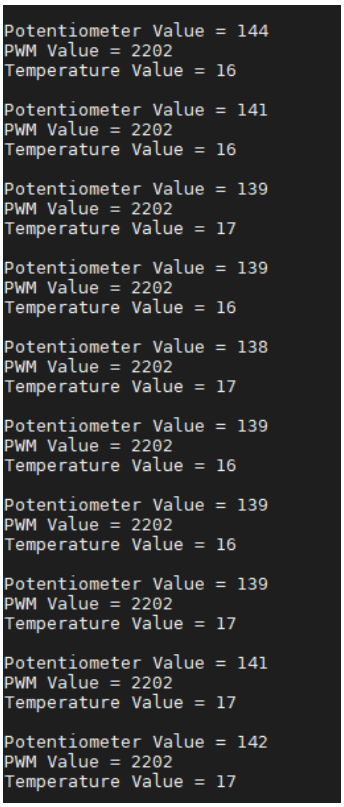
```
    PWM_start(pwm1);
    while(1){
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);
        if(button == 1){
            dutyCycle = (ADCread * 3000) / 1023;
            PWMsave = dutyCycle;
            PWM_setDuty(pwm1, dutyCycle);
            button = 0;
            Semaphore_post(semHandle);
            Task_sleep(sleepCount);
        }
        else    {
            Semaphore_post(semHandle);
            Task_sleep(sleepCount);
        }
    }
}

// ICM20948 task function to read 0x39 temp high value
Void taskFunction4(UArg arg0, UArg arg1){
    while(1){
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);
        uint8_t txBuffer[3] = {0x06, 0x01, 0x39};
        uint8_t rxBuffer[2];
        int i;
        I2C_Handle      i2c;
        I2C_Params      i2cParams;
        I2C_Transaction i2cTransaction;

        I2C_init();
        I2C_Params_init(&i2cParams);
        i2cParams.bitRate = I2C_400kHz;
        i2c = I2C_open(CONFIG_I2C_0, &i2cParams);

        i2cTransaction.slaveAddress = 0x68;
        i2cTransaction.writeBuf   = txBuffer;
        i2cTransaction.writeCount = 3;
        i2cTransaction.readBuf    = rxBuffer;
        i2cTransaction.readCount  = 2;

        for (i = 0; i < 1; i++){
            if (I2C_transfer(i2c, &i2cTransaction)) {

                output = (rxBuffer[0] << 6) | (rxBuffer[1] >> 2);


                if (rxBuffer[0] & 0x80) {
                    output |= 0xF000;
                }
                output /= 32;
                }
            else{
                System_printf("I2C Bus fault\n");
            }
        }
```

```
        I2C_close(i2c);
        Semaphore_post(semHandle);
        Task_sleep(sleepCount);
    }
}

// heart beat task function
Void taskFunction5(UArg arg0, UArg arg1){
    while(1){
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);
        GPIO_toggle(CONFIG_GPIO_LED_1);
        Semaphore_post(semHandle);
        Task_sleep(5000);
    }
}

int main()
{
    Task_Params taskParams;
    Semaphore_Params semParams;

    Board_init();
    Display_init();
    GPIO_init();
    ADC_init();

    // task parameter setup
    Task_Params_init(&taskParams);
    taskParams.stackSize = stack1;
    taskParams.stack = &t1Stack;
    taskParams.priority = 1;
    Task_construct(&task1Struct, (Task_FuncPtr)taskFunction1, &taskParams, NULL);

    taskParams.stack = &t2Stack;
    taskParams.priority = 1;
    Task_construct(&task2Struct, (Task_FuncPtr)taskFunction2, &taskParams, NULL);

    taskParams.stack = &t3Stack;
    taskParams.priority = 1;
    Task_construct(&task3Struct, (Task_FuncPtr)taskFunction3, &taskParams, NULL);

    taskParams.stack = &t4Stack;
    taskParams.priority = 1;
    Task_construct(&task4Struct, (Task_FuncPtr)taskFunction4, &taskParams, NULL);

    taskParams.stack = &t5Stack;
    taskParams.priority = 1;
    Task_construct(&task5Struct, (Task_FuncPtr)taskFunction5, &taskParams, NULL);

    // semaphore setup
    Semaphore_Params_init(&semParams);
    Semaphore_construct(&semStruct, 1, &semParams);
    semHandle = Semaphore_handle(&semStruct);

    // pushbutton setup
```

```
    GPIO_setConfig(CONFIG_GPIO_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
// falling edge read
    GPIO_setCallback(CONFIG_GPIO_0, pushButton);
// pushbutton callback function
    GPIO_enableInt(CONFIG_GPIO_0);
// enable interrupt

    // sleep 20ms
    sleepCount = 20000 / Clock_tickPeriod;

    BIOS_start();   // start bios
    return(0);
}
```



CC1352 Board Connections to 10k Potentiometer and ICM20948



Sample Terminal Output Displaying PWM Value, POT Value, and Temp

GitHub: https://github.com/brianwolak/advanced_submissions/tree/main/DA_4

YouTube: **https://youtu.be/ei7Jxw7cP_4**

"This assignment submission is my own, original work".
Brian Wolak