CPE403 – Advanced Embedded Systems

Design Assignment 3

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Meral Abu-Jaser

Email: abujaser@unlv.nevada.edu

Github Repository link (root): https://github.com/MeralAbuJaser/Advanced-Embedded-Systems

Youtube Playlist link (root): https://www.youtube.com/playlist?list=PLmRQUGgBgm2dlt37RCWlrSrGj7KxvDKmj

Follow the submission guideline to be awarded points for this Assignment.

Submit the following for all Assignments:

1. In the document, 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.
2. Create a private Github repository with a random name (no CPE/403, Lastname, Firstname). Place all labs under the root folder TIVAC, sub-folder named Assignment1, with one document and one video link file for each lab, place modified c files named as asng_taskxx.c.
3. If multiple c files or other libraries are used, create a folder asng1_t01 and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup_ccs.c and other include files, c) text file with youtube video links (see template).
5. Submit the doc file in canvas before the due date. The root folder of the github assignment directory should have the documentation and the text file with youtube video links.
6. Organize your youtube videos as playlist under the name “cpe403”. The playlist should have the video sequence arranged as submission or due dates.
7. Only submit pdf documents. Do not forget to upload this document in the github repository and in the canvas submission portal.

1. Task 01

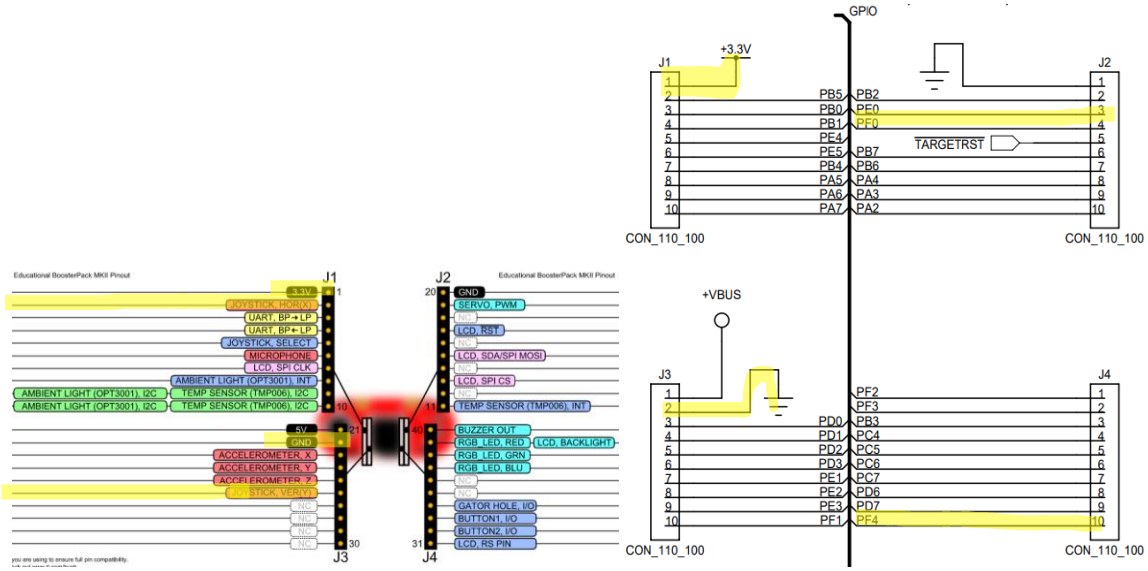
```
/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include <xdc/runtime/Log.h>                //needed for any Log_info() call
#include <xdc/cfg/global.h>                 //header file for statically defined objects/handles
#include <xdc/runtime/Diags.h>

/* BIOS module Headers */
#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/drivers/GPIO.h>

/* Include header files for adc and GPIO functions */
#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/tm4c123gh6pm.h"
#include "driverlib/debug.h"
#include "driverlib/pin_map.h"
#include "driverlib/adc.h"
#include "driverlib/rom.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include <time.h>
#include <inc/hw_gpio.h>
#include "driverlib/uart.h"
#include <ti/drivers/ADC.h>
#include <ti/display/Display.h>
volatile int16_t i16ToggleCount1 = 0;
volatile int16_t i16ToggleCount2 = 0;
#define TASKSTACKSIZE   512

Task_Struct task0Struct;
Char task0Stack[TASKSTACKSIZE];

/* ADC sample count */
#define ADC_SAMPLE_COUNT  (10)

#define THREADSTACKSIZE   (768)

/* ADC conversion result variables */
uint16_t adcValue0;
uint32_t adcValue0MicroVolt;
uint16_t adcValue1[ADC_SAMPLE_COUNT];
uint32_t adcValue1MicroVolt[ADC_SAMPLE_COUNT];

//*****************************************************************************
void delay_simple(void)
{
     SysCtlDelay(6700000);      // creates ~500ms delay - TivaWare fxn

}

/*
 *  ======== heartBeatFxn ========
 *  Toggle the Board_LED0. 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_LED0);
    }
}
```

```
104 void init_ADC()
105 {
106     GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);
107     SysCtlDelay(80u);
108
109     // Use ADC0 sequence 0 to sample channel 0 once for each timer period
110     ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PIOSC | ADC_CLOCK_RATE_HALF, 1);
111
112     SysCtlDelay(10); // Time for the clock configuration to set
113
114     IntDisable(INT_ADC0SS0);
115     ADCIntDisable(ADC0_BASE, 0u);
116     ADCSequenceDisable(ADC0_BASE,0u);
117     // With sequence disabled, it is now safe to load the new configuration parameters
118
119     ADCSequenceConfigure(ADC0_BASE, 0u, ADC_TRIGGER_TIMER, 0u);
120     ADCSequenceStepConfigure(ADC0_BASE,0u,0u,ADC_CTL_CH0| ADC_CTL_END | ADC_CTL_IE);
121     ADCSequenceEnable(ADC0_BASE,0u); //Once configuration is set, re-enable the sequencer
122     ADCIntClear(ADC0_BASE,0u);
123     ADCSequenceDMAEnable(ADC0_BASE,0);
124     IntEnable(INT_ADC0SS0);
125
126 }
127 void taskFxn1(void)
128 {
129     while(1)
130     {
131     Semaphore_pend(task1sem, BIOS_WAIT_FOREVER);
132     // LED values - 2=RED, 4=BLUE, 8=GREEN
133     if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
134     {
135         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
136     }
137     else
138     {
139         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
140     }
141
142   //  delay_simple();                                 // create a delay of ~1/2sec
143
144     i16ToggleCount1 += 1;                     // keep track of #toggles
145
146     Log_info1("LED2 TOGGLED [%u] times", i16ToggleCount1);    // send #toggles to Log Display
147     //System_printf("Count: %d\n", i16ToggleCount);
148     //System_flush();
149     }
150 }
151
152
153 void taskFxn2(void)
154 {
155     while(1)
156     {
157     Semaphore_pend(task2sem, BIOS_WAIT_FOREVER);
158     // LED values - 2=RED, 4=BLUE, 8=GREEN
159     if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_3))
160     {
161         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
162     }
163     else
164     {
165         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8);
166     }
167
168   //  delay_simple();                                 // create a delay of ~1/2sec
169
170     i16ToggleCount2 += 1;                     // keep track of #toggles
171
172     Log_info1("LED3 TOGGLED [%u] times", i16ToggleCount2);    // send #toggles to Log Display
173     //System_printf("Count: %d\n", i16ToggleCount);
174     //System_flush();
175     }
```

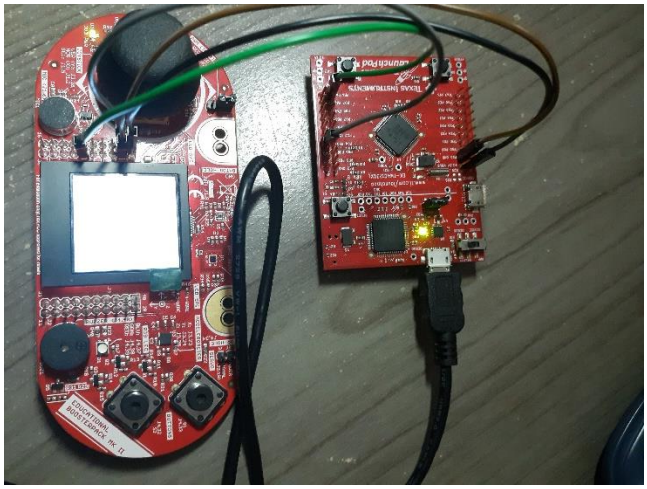
```
//Timer 2 setup
void timer2Init()
{
    uint32_t ui32Period;
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);           // enable Timer 2 periph clks
    TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);        // cfg Timer 2 mode - periodic

    ui32Period = (SysCtlClockGet()/500) / 2;
    TimerLoadSet(TIMER2_BASE, TIMER_A, ui32Period-1);           // set Timer 2 period

    TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);        // enables Timer 2 to interrupt CPU

    TimerEnable(TIMER2_BASE, TIMER_A);                      // enable Timer 2

}


volatile uint32_t tickCount=0;

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

    /* Set up the System Clock */
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    /* Enable all the peripherals */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    /* Construct heartBeat Task  thread */
       Task_Params_init(&taskParams);
       taskParams.arg0 = 1000;
       taskParams.stackSize = TASKSTACKSIZE;
       taskParams.stack = &task0Stack;
        Task_construct(&task0Struct, (Task_FuncPtr)heartBeatFxn, &taskParams, NULL);

    /* Unlock pin PF0 */
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK)= GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK)= 0;

    /* Configure Enable pin as output */
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3);

    timer2Init();
    BIOS_start();    /* Does not return */

    if (adc == NULL) {
            Display_printf(hSerial, 1, 0, "Error initializing CONFIG_ADC_0\n");
            while (1);
        }
        /* Configure the LED pin */
        GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);

        /* Turn on user LED */
        GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
        /* Blocking mode conversion */

    if (adc == NULL) {
         Display_printf(hSerial, 1, 0, "Error initializing CONFIG_ADC_0\n");
         while (1);
     }
     /* Configure the LED pin */
     GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);

     /* Turn on user LED */
     GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
     /* Blocking mode conversion */
```

```
    if (adc == NULL) {
        Display_printf(hSerial, 1, 0, "Error initializing CONFIG_ADC_0\n");
        while (1);
    }
    /* Configure the LED pin */
    GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);

    /* Turn on user LED */
    GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
    /* Blocking mode conversion */
    while(1)
    {
    res = ADC_convert(adc, &adcValue0);
    if (res == ADC_STATUS_SUCCESS) {
        Display_printf(hSerial, 1, 0, "CONFIG_ADC_0 raw result: %d\n", adcValue0);
        if(adcValue0 > 1000)
        {
            GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
        }
        else{
            GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_OFF);
        }
    }
    else {
        Display_printf(hSerial, 1, 0, "CONFIG_ADC_0 convert failed\n");
      }
    usleep(500000);
    }
    return (NULL);
}
//---------------------------------------------------------------------------
// Timer ISR  to be called by BIOS Hwi
//
// Posts Semaphore for releasing tasks
//---------------------------------------------------------------------------
void Timer_ISR(void)
{
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);         // must clear timer flag FROM timer
    tickCount++;  //tickCount is incremented every 2 ms.

    if(tickCount == 300)
    {
        Semaphore_post(task1sem);
    }
    else if(tickCount == 600)
    {
        Semaphore_post(task2sem);
        tickCount = 0;
    }
}
```

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



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



4. Declaration

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

"This assignment submission is my own, original work".
Meral Abu-Jaser

Dr. Venki,

I had a hard time working with this assignment, I know that I should use log to read the values and convert them, but it did not work with me, nor did the UART. I am going to try my best to work on it and keep updating it on GitHub. Hopefully, I will meet the end results that are required for this assignment.