CPE403 – Advanced Embedded Systems

Design Assignment #01

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Cameron Kirk

Email: kirkc1@unlv.nevada.edu

Github Repository link (root): https://github.com/kirkster96/DqF514-not-embedded

Youtube Playlist link (root):

https://www.youtube.com/playlist?list=PLiqmqQ7XKuf7ArV7lO6b20D1ES5SUp0Yk

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. 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.

Task 1

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/debug.h"
#include "driverlib/adc.h"
#include "utils/uartstdio.h"
#include <string.h>


#ifdef DEBUG
void__error__(char *pcFilename, uint32_t ui32Line)
{

}
#endif

//Globals
uint32_t ui32Period;
uint8_t ui8PinData=1;
char buffer[4];

uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempC;
volatile uint32_t ui32TempF;

void GPIOF4IntHandler(void)
{
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_4);


    if(ui8PinData==1){
        ui8PinData=0;
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0xF);
        SysCtlDelay(200000);
    } else {
        ui8PinData=1;
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x0);
        SysCtlDelay(200000);
    }

}

//Timer 1 ISR
void Timer1IntHandler(void)
{
    // Clear the timer ineterrupt
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

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

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

    }

    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

    ui32TempAvg = 
(ui32ADC0Value[0]+ui32ADC0Value[1]+ui32ADC0Value[2]+ui32ADC0Value[3]+2)/4;
    ui32TempC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
    ui32TempF = ((ui32TempC*9) + 160)/5;
    UARTprintf("C %3d\t",ui32TempC);
    UARTprintf("F %3d\t",ui32TempF);
    UARTprintf("\n");
}

int main(void)
{
    //Configure Clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | 
SYSCTL_XTAL_16MHZ);
    // Configure peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF))
           {
           }
    // Configure IO
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA, 
GPIO_PIN_TYPE_STD_WPU);

    //register the interrupt handler for PF4
    GPIOIntRegister(GPIO_PORTF_BASE, GPIOF4IntHandler);
    //sw2 goes low when pressed
    GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_FALLING_EDGE);
    //enable interrupts on PF4
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);

    // Configure ADC
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);

    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0); // Changed to
sequencer #2
    ADCSequenceStepConfigure(ADC0_BASE,1,0,ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE,1,1,ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE,1,2,ADC_CTL_TS);

    ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE,1);

    //Configure Timer 1 module
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    ui32Period = SysCtlClockGet()/2;    //Period of 0.5s or 2Hz
    TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period - 1);
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    //Configure pins for UART
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
    //Enable interrupts
    IntMasterEnable();
    TimerEnable(TIMER1_BASE, TIMER_A);
    ADCSequenceEnable(ADC0_BASE, 2);
    //Initial message to terminal display
    UARTprintf("Temperature:\n");
    while(1) //wait forever
    {

    }

}
```

Task 2

```
//*****************************************************************************
//
// qs-rgb.c - Quickstart for the EK-TM4C123GXL.
//
// Copyright (c) 2012-2017 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
```

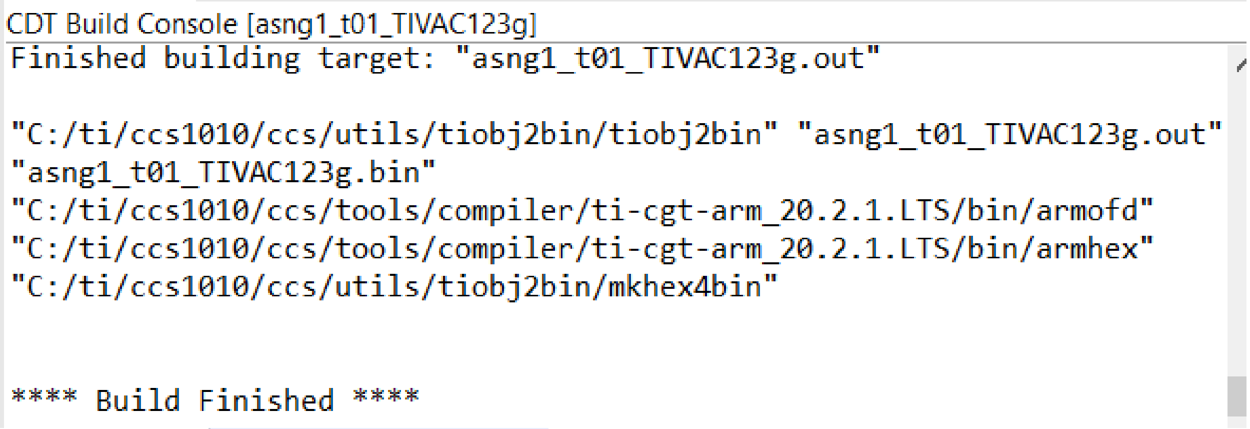
```c
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.4.178 of the EK-TM4C123GXL Firmware Package.
//
//*****************************************************************************

#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include <time.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_hibernate.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/hibernate.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "utils/cmdline.h"
#include "drivers/rgb.h"
#include "drivers/buttons.h"
#include "rgb_commands.h"
#include "asng1_t02_TIVAC123g.h"
#include "driverlib/adc.h"


//*****************************************************************************
//
// Input buffer for the command line interpreter.
//
//*****************************************************************************
static char g_cInput[APP_INPUT_BUF_SIZE];


//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif
uint8_t ui8RedData=0;
uint8_t ui8GreenData=0;
uint8_t ui8BlueData=0;

uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempC;
volatile uint32_t ui32TempF;


int EnableRed(void){
    ui8RedData=1;
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
    return 1;
}

int DisableRed(void){
    ui8RedData=0;
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
    return 1;
}
int EnableGreen(void){
    ui8GreenData=1;
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8);
    return 1;
}

int DisableGreen(void){
    ui8GreenData=0;
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0x00);
    return 1;
}

int EnableBlue(void){
    ui8BlueData=1;
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
    return 1;
}

int DisableBlue(void){
    ui8BlueData=0;
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
    return 1;
}

int printCel(void){
    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

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

        }
    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

    ui32TempAvg =
(ui32ADC0Value[0]+ui32ADC0Value[1]+ui32ADC0Value[2]+ui32ADC0Value[3]+2)/4;
    ui32TempC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
    UARTprintf("C %3d\t",ui32TempC);
    UARTprintf("\n");

    return 1;
}

int printFar(void){
    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

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

        }

    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

    ui32TempAvg =
(ui32ADC0Value[0]+ui32ADC0Value[1]+ui32ADC0Value[2]+ui32ADC0Value[3]+2)/4;
    ui32TempC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
    ui32TempF = ((ui32TempC*9) + 160)/5;
    UARTprintf("F %3d\t",ui32TempF);
    UARTprintf("\n");

    return 1;
}

int PrintStatus(void){

    UARTprintf("Red is ");
    if(ui8RedData != 0)
        UARTprintf(" on. ");
    else
        UARTprintf(" off. ");

    UARTprintf("Green is ");
    if(ui8GreenData != 0)
        UARTprintf(" on. ");
    else
        UARTprintf(" off. ");

    UARTprintf("Blue is ");
    if(ui8BlueData != 0)
        UARTprintf(" on. ");
    else
        UARTprintf(" off. ");
    UARTprintf("\n");

    return 1;
```

```c
}

//*****************************************************************************
//
// Configure the UART and its pins.  This must be called before UARTprintf().
//
//*****************************************************************************
void
ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

//*****************************************************************************
//
// Main function performs init and manages system.
//
// Called automatically after the system and compiler pre-init sequences.
// Performs system init calls, restores state from hibernate if needed and
// then manages the application context duties of the system.
//
//*****************************************************************************
int
main(void)
{
    int32_t i32CommandStatus;


    //
    // Set the system clock to run at 40Mhz off PLL with external crystal as
    // reference.
    //
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
                       SYSCTL_OSC_MAIN);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
       while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF))
              {
              }
       // Configure IO
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    // Configure ADC
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
        ADCHardwareOversampleConfigure(ADC0_BASE, 64);

        ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0); // Changed to
sequencer #2
        ADCSequenceStepConfigure(ADC0_BASE,1,0,ADC_CTL_TS);
        ADCSequenceStepConfigure(ADC0_BASE,1,1,ADC_CTL_TS);
        ADCSequenceStepConfigure(ADC0_BASE,1,2,ADC_CTL_TS);

        ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
        ADCSequenceEnable(ADC0_BASE,1);

    //
    // Enable and Initialize the UART.
    //
    ConfigureUART();
    ADCSequenceEnable(ADC0_BASE, 2);


    UARTprintf("Welcome to the Tiva C Series TM4C123G LaunchPad!\n");
    UARTprintf("Type 'help' for a list of commands\n");
    UARTprintf("> ");

    //
    // spin forever and wait for carriage returns or state changes.
    //
    while(1)
    {

        UARTprintf("\n>");


        //
        // Peek to see if a full command is ready for processing
        //
        while(UARTPeek('\r') == -1)
        {
            //
            // millisecond delay.  A SysCtlSleep() here would also be OK.
            //
            SysCtlDelay(SysCtlClockGet() / (1000 / 3));

        }

        //
        // a '\r' was detected get the line of text from the user.
        //
        UARTgets(g_cInput,sizeof(g_cInput));

        //
        // Pass the line from the user to the command processor.
        // It will be parsed and valid commands executed.
        //
        i32CommandStatus = CmdLineProcess(g_cInput);

        //
        // Handle the case of bad command.
        //
        if(i32CommandStatus == CMDLINE_BAD_CMD)
        {
            UARTprintf("Bad command!\n");
        }

        //
        // Handle the case of too many arguments.
        //
        else if(i32CommandStatus == CMDLINE_TOO_MANY_ARGS)
        {
            UARTprintf("Too many arguments for command processor!\n");
        }
    }
}
```

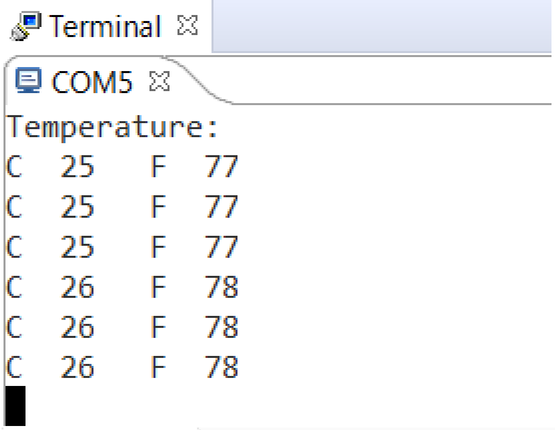
2. Block diagram and/or Schematics showing the components, pins used, and interface.

None on this assignment.

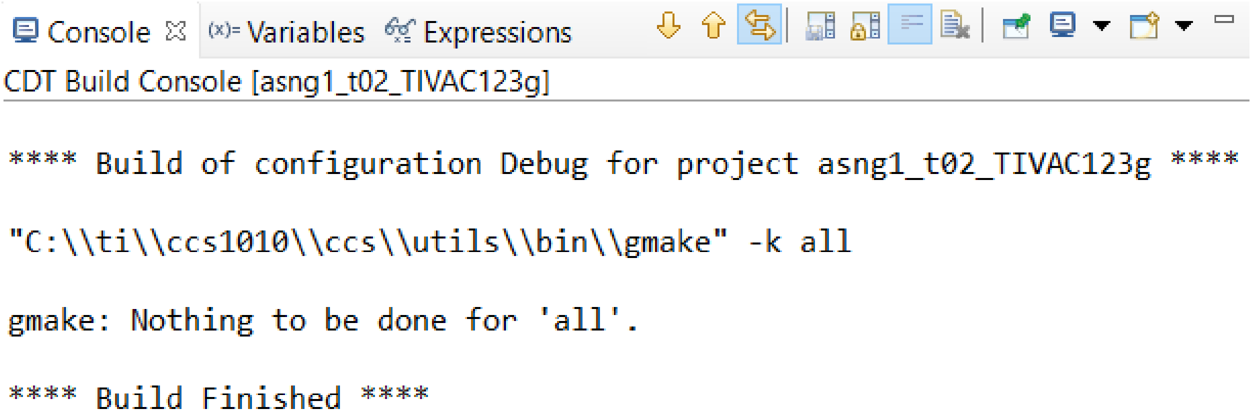
3. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.



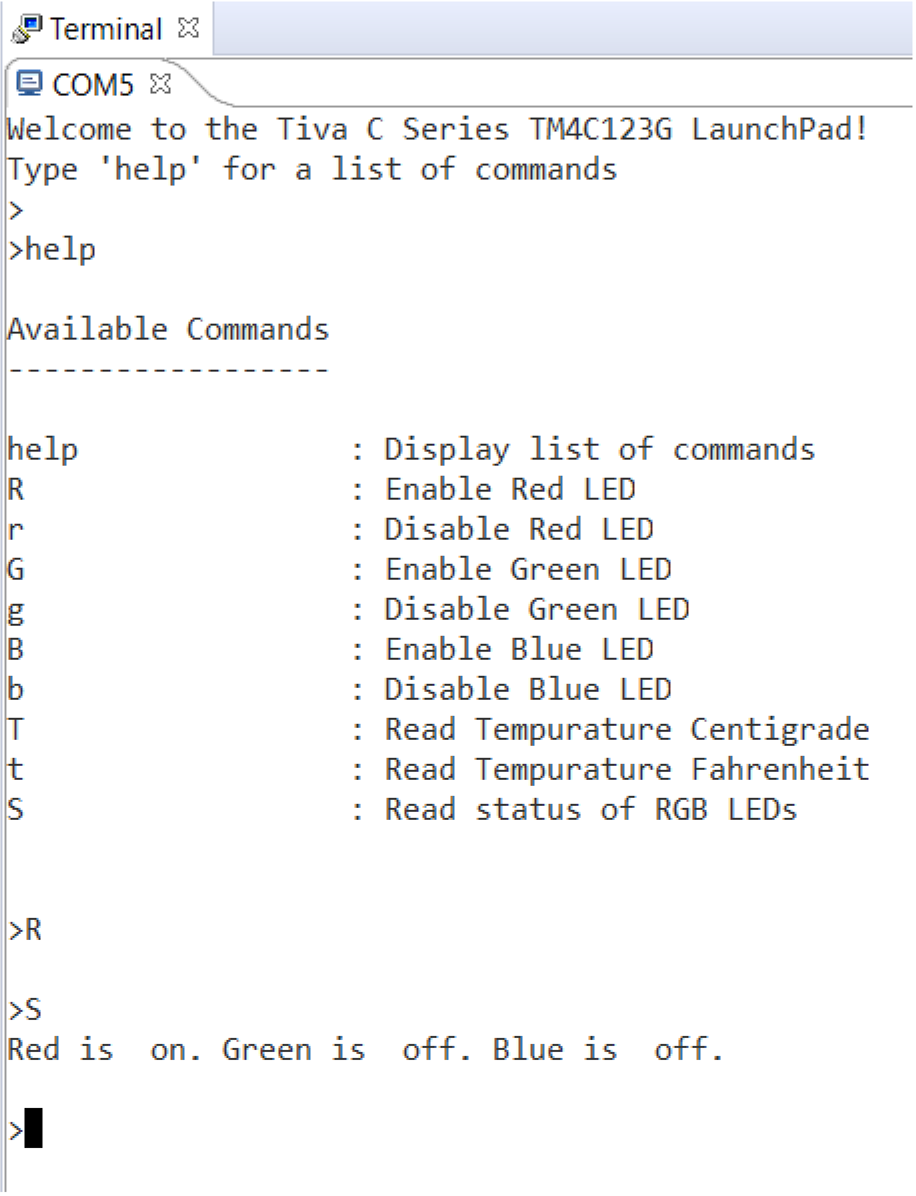
```
Finished building target: "asng1_t01_TIVAC123g.out"

"C:/ti/ccs1010/ccs/utils/tiobj2bin/tiobj2bin" "asng1_t01_TIVAC123g.out"
"asng1_t01_TIVAC123g.bin"
"C:/ti/ccs1010/ccs/tools/compiler/ti-cgt-arm_20.2.1.LTS/bin/armofd"
"C:/ti/ccs1010/ccs/tools/compiler/ti-cgt-arm_20.2.1.LTS/bin/armhex"
"C:/ti/ccs1010/ccs/utils/tiobj2bin/mkhex4bin"


**** Build Finished ****
```



```
Temperature:
C   25    F   77
C   25    F   77
C   25    F   77
C   26    F   78
C   26    F   78
C   26    F   78
```



```
**** Build of configuration Debug for project asng1_t02_TIVAC123g ****

"C:\\ti\\ccs1010\\ccs\\utils\\bin\\gmake" -k all

gmake: Nothing to be done for 'all'.

**** Build Finished ****
```



```
Welcome to the Tiva C Series TM4C123G LaunchPad!
Type 'help' for a list of commands
>
>help

Available Commands
------------------

help               : Display list of commands
R                  : Enable Red LED
r                  : Disable Red LED
G                  : Enable Green LED
g                  : Disable Green LED
B                  : Enable Blue LED
b                  : Disable Blue LED
T                  : Read Tempurature Centigrade
t                  : Read Tempurature Fahrenheit
S                  : Read status of RGB LEDs


>R

>S
Red is  on. Green is  off. Blue is  off.

>
```

4. Declaration

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

“This assignment submission is my own, original work”.
Cameron Kirk