CPE403 – Advanced Embedded Systems

Design Assignment 01

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.

Task 01 Code:

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

#define BAUD_RATE 115200
#define GPIO_PA0_U0RX 0x00000001
#define GPIO_PA1_U0TX 0x00000401
#define RED_LED   GPIO_PIN_1
#define BLUE_LED  GPIO_PIN_2
#define GREEN_LED GPIO_PIN_3
#define BUTTON GPIO_PIN_4

// Reserves memory to move the temperature FIFO into.
uint32_t temperatureFIFO[4];
// Variables to calculate the temperature with.
volatile uint32_t tAverage, tCelsius, tFahrenheit;
// Keeps track of the LED status.
```

```c
volatile bool ledOn = false;

int main(void) {
    uint32_t loadVal;

    // Calculates the cycle values for a 0.5s delay.
    loadVal = (SysCtlClockGet() / 2);

    // Sets up the system clock.
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);

    // Enables peripherals.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_WTIMER0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    // Sets up peripherals.
    TimerConfigure(WTIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(WTIMER0_BASE, TIMER_A, loadVal);
    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);
    ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

    // Sets up interrupts.
    IntEnable(INT_WTIMER0A);
    TimerIntEnable(WTIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntEnable(INT_GPIOF);
    GPIOIntTypeSet(GPIO_PORTF_BASE, BUTTON, GPIO_FALLING_EDGE);
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
    IntMasterEnable();

    // Enables stuff.
    TimerEnable(WTIMER0_BASE, TIMER_A);
    ADCSequenceEnable(ADC0_BASE, 2);

    UARTprintf("Starting...");
    while (1)
    {
    }
}

void timerhandler(void) {
    // Clears interrupt flag.
    TimerIntClear(WTIMER0_BASE, TIMER_TIMA_TIMEOUT);

    // Starts ADC conversion.
    ADCIntClear(ADC0_BASE, 2);
    ADCProcessorTrigger(ADC0_BASE, 2);
    ADCSequenceDataGet(ADC0_BASE, 2, temperatureFIFO);
    tAverage = (temperatureFIFO[0] + temperatureFIFO[1] + temperatureFIFO[2] +
temperatureFIFO[3] + 2)/4;
    tCelsius = (1475 - ((2250*tAverage))/4096)/10;
    tFahrenheit = ((tCelsius*9) + 160)/5;

    UARTprintf("ADC: %3d\t", tAverage);
    UARTprintf("Celsius: %3dC\t", tCelsius);
    UARTprintf("Fahrenheit: %3dF", tFahrenheit);
    UARTprintf("\n\r");
}

void buttonpresshandler() {
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
    ledOn = !ledOn;
    if (ledOn) {
        GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
RED_LED|BLUE_LED|GREEN_LED);
    } else {
        GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, 0);
    }
}
```

Task 02 Code:

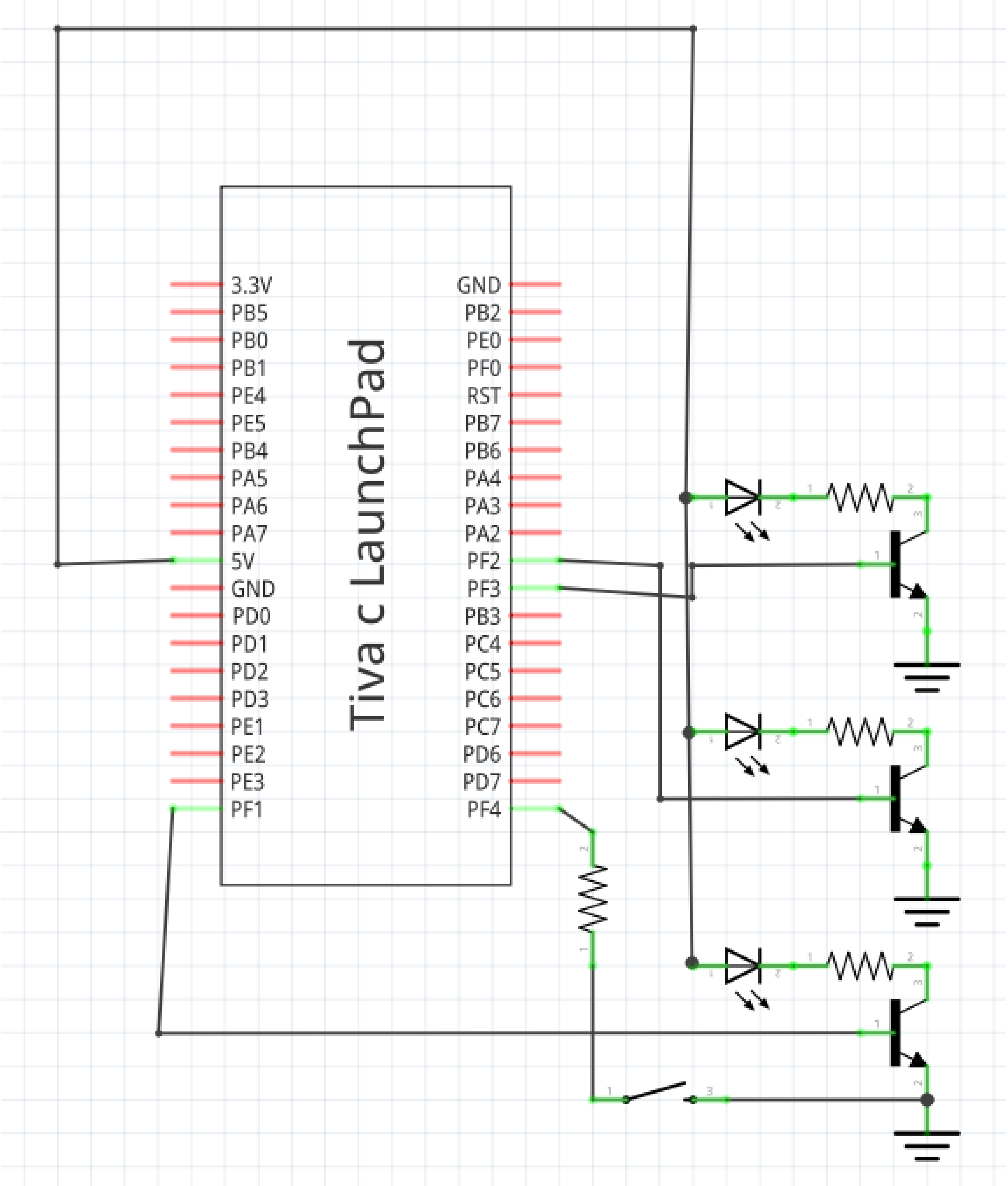
```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"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

#define BAUD_RATE 115200
#define GPIO_PA0_U0RX 0x00000001
#define GPIO_PA1_U0TX 0x00000401
#define RED_LED   GPIO_PIN_1
#define BLUE_LED  GPIO_PIN_2
#define GREEN_LED GPIO_PIN_3
#define BUTTON GPIO_PIN_4

// Reserves memory to move the temperature FIFO into.
uint32_t temperatureFIFO[4];
uint32_t ledSequence;
// Used to read in the cmd.
char cmd;
// Used to calculate temperature.
volatile uint32_t tAverage, tCelsius, tFahrenheit;
// Keeps track of the R G B LEDs' status.
volatile bool rLED = false;
volatile bool gLED = false;
volatile bool bLED = false;

int main(void) {
    uint32_t loadVal;

    // Calculates the cycle values for a 0.5s delay.
    loadVal = (SysCtlClockGet() / 2);

    // Sets up the system clock.
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);

    // Enables peripherals.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_WTIMER0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    // Sets up peripherals.
    TimerConfigure(WTIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(WTIMER0_BASE, TIMER_A, loadVal);
    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);
    ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

    // Sets up interrupts.
    IntEnable(INT_WTIMER0A);
    TimerIntEnable(WTIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntEnable(INT_GPIOF);
    GPIOIntTypeSet(GPIO_PORTF_BASE, BUTTON, GPIO_FALLING_EDGE);
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
    IntMasterEnable();

    // Enables stuff.
    TimerEnable(WTIMER0_BASE, TIMER_A);
    ADCSequenceEnable(ADC0_BASE, 2);

    UARTprintf("\n\rEnter command (R/r: Red LED On/Off\tG/g: Green LED On/Off\tB/b:
Blue LED On/Off\tT/t:"
            "Temperature in Celsius/Fahrenheit\tS: LED Status)\n\r");
    while (1)
    {
    }
}

void timerhandler(void) {
    // Clears interrupt flag.
    TimerIntClear(WTIMER0_BASE, TIMER_TIMA_TIMEOUT);

    // Starts ADC conversion.
    ADCIntClear(ADC0_BASE, 2);
    ADCProcessorTrigger(ADC0_BASE, 2);
    ADCSequenceDataGet(ADC0_BASE, 2, temperatureFIFO);

    tAverage = (temperatureFIFO[0] + temperatureFIFO[1] + temperatureFIFO[2] +
temperatureFIFO[3] + 2)/4;
    tCelsius = (1475 - ((2250*tAverage))/4096)/10;
    tFahrenheit = ((tCelsius*9) + 160)/5;
}


void buttonpresshandler() {
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_INT_PIN_4);

    rLED = !rLED;
    gLED = !gLED;
    bLED = !bLED;

    ledSequence = (rLED ? RED_LED : 0)|(gLED ? GREEN_LED : 0)|(bLED ? BLUE_LED : 0);

    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, ledSequence);
}

void uartinthandler() {
    uint32_t ui32Status;

    // Clears the interrupt.
    ui32Status = UARTIntStatus(UART0_BASE, true);
    UARTIntClear(UART0_BASE, ui32Status);
    // Grabs data from the UART.
    cmd = UARTCharGetNonBlocking(UART0_BASE);

    switch (cmd) {
    // The first 6 commands set the status of the corresponding LEDs.
    case 'R':
        rLED = true;
        break;
    case 'r':
        rLED = false;
        break;
    case 'G':
        gLED = true;
        break;
    case 'g':
        gLED = false;
        break;
    case 'B':
        bLED = true;
        break;
    case 'b':
        bLED = false;
        break;
    case 'T':
        // Sends the temperature in celsius.
        UARTprintf("Temperature: %3dC\n\r", tCelsius);
        break;
    case 't':
        // Sends the temperature in fahrenheit.
        UARTprintf("Temperature: %3dF\n\r", tFahrenheit);
        break;
    case 'S':
        // Displays the status of the RGB. Format is <R, G, B>
        UARTprintf("RGB Status: <%i, %i, %i>\n\r", rLED, gLED, bLED);
        break;
    default:
        // Occurs when the TivaC receives data not accounted for.
        UARTprintf("Command %c not found.\n\r", cmd);
    }

    ledSequence = (rLED ? RED_LED : 0)|(gLED ? GREEN_LED : 0)|(bLED ? BLUE_LED : 0);

    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, ledSequence);
}
```

Task 03 Code:

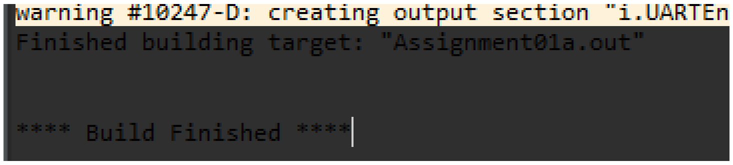
```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_adc.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include "driverlib/udma.h"
#include "utils/uartstdio.c"

#define BAUD_RATE 115200
#define GPIO_PA0_U0RX 0x00000001
#define GPIO_PA1_U0TX 0x00000401
#define RED_LED   GPIO_PIN_1
#define BLUE_LED  GPIO_PIN_2
#define GREEN_LED GPIO_PIN_3
#define BUTTON GPIO_PIN_4
#define MEM_BUFFER_SIZE 64
#pragma DATA_ALIGN(pui8ControlTable, 1024)

uint8_t pui8ControlTable[1024];
static uint16_t destination[MEM_BUFFER_SIZE];
uint32_t ledSequence;
uint16_t i;
char cmd;
volatile uint32_t tAverage, tCelsius, tFahrenheit;
volatile bool rLED = false;
volatile bool gLED = false;
volatile bool bLED = false;

void inituDMA(void)
{
    // Initizlies the uDMA and sets it to transfer from the ADC to the memory array
pointed to by 'destination'.
    uDMAEnable();
    uDMAControlBaseSet(pui8ControlTable);
    uDMAChannelAttributeDisable(UDMA_CHANNEL_ADC0, UDMA_ATTR_ALTSELECT |
UDMA_ATTR_HIGH_PRIORITY | UDMA_ATTR_REQMASK);
    uDMAChannelAttributeEnable(UDMA_CHANNEL_ADC0, UDMA_ATTR_USEBURST);
    uDMAChannelControlSet(UDMA_CHANNEL_ADC0 | UDMA_PRI_SELECT, UDMA_SIZE_16 |
UDMA_SRC_INC_NONE | UDMA_DST_INC_16 | UDMA_ARB_128);
    uDMAChannelTransferSet(UDMA_CHANNEL_ADC0 | UDMA_PRI_SELECT, UDMA_MODE_AUTO, (void
*)(ADC0_BASE + ADC_O_SSFIFO0), destination, MEM_BUFFER_SIZE);
    uDMAChannelEnable(UDMA_CHANNEL_ADC0);
    //uDMAChannelRequest(UDMA_CHANNEL_ADC0);
}

int main(void) {
    uint32_t loadVal;

    // Calculates the cycle value for a 0.5s delay.
    loadVal = (SysCtlClockGet() / 2);

    // Sets up the system clock.
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);
```

```
    // Enables peripherals.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_WTIMER0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);

    // Sets up peripherals.
    inituDMA();
    TimerConfigure(WTIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(WTIMER0_BASE, TIMER_A, loadVal);
    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);
    ADCSequenceDMAEnable(ADC0_BASE, 0);
    ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

    // Sets up interrupts.
    IntEnable(INT_WTIMER0A);
    TimerIntEnable(WTIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntEnable(INT_GPIOF);
    GPIOIntTypeSet(GPIO_PORTF_BASE, BUTTON, GPIO_FALLING_EDGE);
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
    IntEnable(INT_UDMAERR);
    IntMasterEnable();

    // Enables stuff.
    TimerEnable(WTIMER0_BASE, TIMER_A);
    ADCSequenceEnable(ADC0_BASE, 0);

    UARTprintf("\n\rEnter command (R/r: Red LED On/Off\tG/g: Green LED On/Off\tB/b:
Blue LED On/Off\tT/t:"
            "Temperature in Celsius/Fahrenheit\tS: LED Status)\n\r");
    while (1)
    {
    }
}

void timerhandler(void) {
    // Clears interrupt flag.
    TimerIntClear(WTIMER0_BASE, TIMER_TIMA_TIMEOUT);

    // Starts ADC conversion.
    ADCIntClear(ADC0_BASE, 0);
    ADCProcessorTrigger(ADC0_BASE, 0);
    uDMAChannelEnable(UDMA_CHANNEL_ADC0);
}


void buttonpresshandler() {
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_INT_PIN_4);

    rLED = !rLED;
    gLED = !gLED;
    bLED = !bLED;

    ledSequence = (rLED ? RED_LED : 0)|(gLED ? GREEN_LED : 0)|(bLED ? BLUE_LED : 0);

    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, ledSequence);
}

void uartinthandler() {
    uint32_t ui32Status;

    // Clears the interrupt.
    ui32Status = UARTIntStatus(UART0_BASE, true);
    UARTIntClear(UART0_BASE, ui32Status);

    // Grabs data from the UART.
    cmd = UARTCharGetNonBlocking(UART0_BASE);

    switch (cmd) {
    // The first 6 commands set the status of the corresponding LEDs.
    case 'R':
        rLED = true;
        break;
    case 'r':
        rLED = false;
        break;
    case 'G':
        gLED = true;
        break;
    case 'g':
        gLED = false;
        break;
    case 'B':
        bLED = true;
        break;
    case 'b':
        bLED = false;
        break;
    case 'T':
        // Collects data from the buffer and averages the temperature.
        for (i = 0; i < MEM_BUFFER_SIZE; i++) {
            tAverage += destination[i];
            destination[i] = 0;
        }
        tAverage = (tAverage + (MEM_BUFFER_SIZE/2))/MEM_BUFFER_SIZE;
        // Converts average to celsius and outputs to the terminal.
        tCelsius = (1475 - ((2250*tAverage))/4096)/10;
        uDMAChannelTransferSet(UDMA_CHANNEL_ADC0 | UDMA_PRI_SELECT, UDMA_MODE_AUTO,
(void *)(ADC0_BASE + ADC_O_SSFIFO0), destination, MEM_BUFFER_SIZE);
        uDMAChannelEnable(UDMA_CHANNEL_ADC0);
        UARTprintf("Temperature: %3dC\n\r", tCelsius);
        break;
    case 't':
        // Collects data from the buffer and averages the temperature.
        for (i = 0; i < MEM_BUFFER_SIZE; i++) {
            tAverage += destination[i];
            destination[i] = 0;
        }
        tAverage = (tAverage + (MEM_BUFFER_SIZE/2))/MEM_BUFFER_SIZE;
        tCelsius = (1475 - ((2250*tAverage))/4096)/10;
        // Converts average to fahrenheit and outputs to the terminal.
        tFahrenheit = ((tCelsius*9) + 160)/5;
        uDMAChannelTransferSet(UDMA_CHANNEL_ADC0 | UDMA_PRI_SELECT, UDMA_MODE_AUTO,
(void *)(ADC0_BASE + ADC_O_SSFIFO0), destination, MEM_BUFFER_SIZE);
        uDMAChannelEnable(UDMA_CHANNEL_ADC0);
        UARTprintf("Temperature: %3dF\n\r", tFahrenheit);
        break;
    case 'S':
        // Displays the status of the RGB. Format is <R, G, B>
        UARTprintf("RGB Status: <%i, %i, %i>\n\r", rLED, gLED, bLED);
        break;
    default:
        // Occurs when the TivaC receives data not accounted for.
        UARTprintf("Command %c not found.\n\r", cmd);
    }

    ledSequence = (rLED ? RED_LED : 0)|(gLED ? GREEN_LED : 0)|(bLED ? BLUE_LED : 0);

    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, ledSequence);
}

void uDMAErrorHandler(void)
{
    uint32_t ui32Status;
    ui32Status = uDMAErrorStatusGet();
    if(ui32Status)
    {
        uDMAErrorStatusClear();
    }
}
```

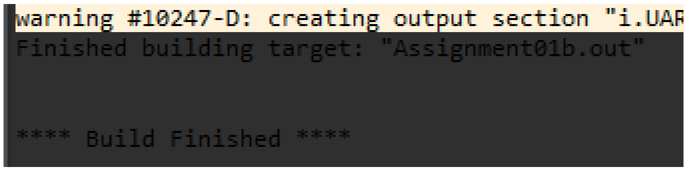
2. Block diagram and/or Schematics showing the components, pins used, and interface.



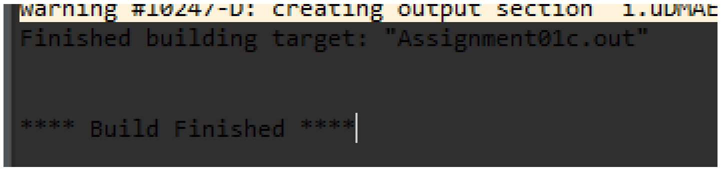
3. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.



```
warning #10247-D: creating output section "i.UARTEn
Finished building target: "Assignment01a.out"


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



```
warning #10247-D: creating output section "i.UAR
Finished building target: "Assignment01b.out"


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



```
warning #10247-D: creating output section "i.uDMAE
Finished building target: "Assignment01c.out"


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

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