Ricky Perez
Github root directory: https://github.com/RickyPerez79/da_submissions_2019

Date Submitted: **11/01/19**

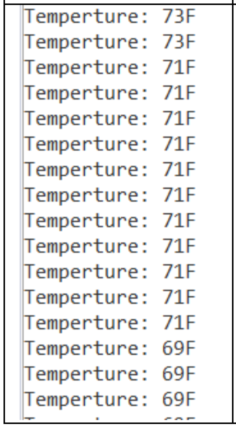
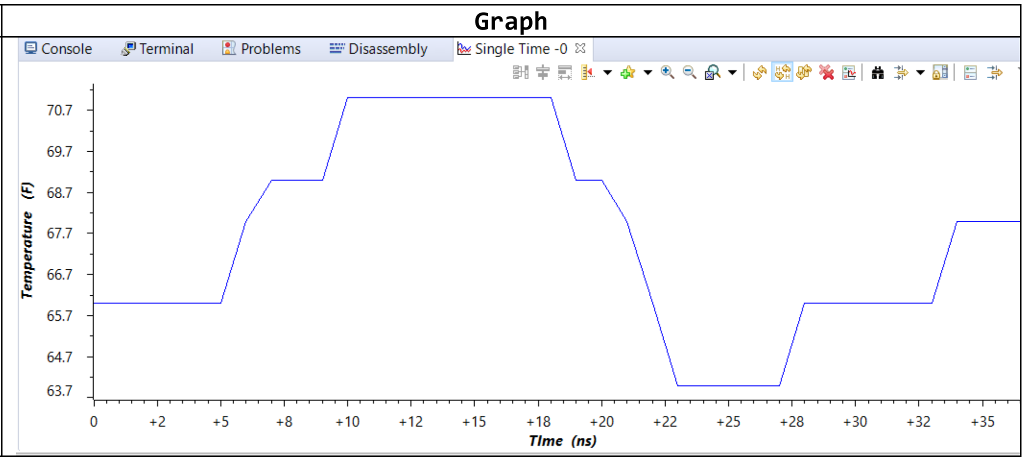
Task 00: Execute provided code

Youtube Link: https://youtu.be/cQWa0bvIrqs

Task 01:

Youtube Link: https://youtu.be/Whim_GUTku0

Screenshots:

Terminal	Graph
 Temperture: 73F Temperture: 73F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 71F Temperture: 69F Temperture: 69F Temperture: 69F	

Modified Code:

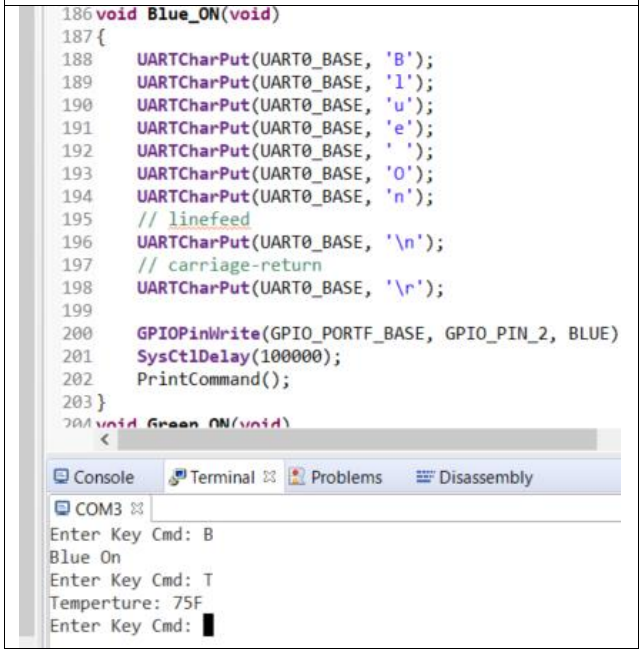
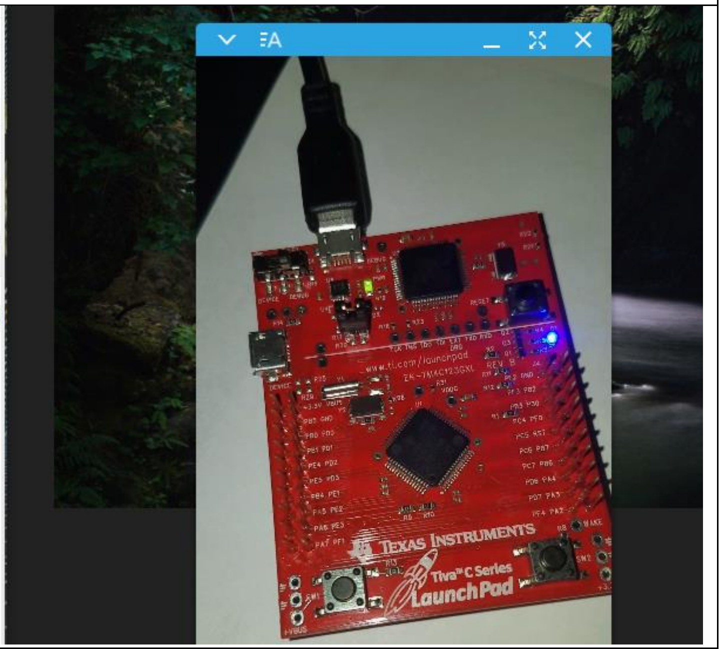
```
// Ricky Perez
// CpE 403
// Lab 7
// Task 1
// Continuously display the temperature of the device (internal temperature sensor)
// on the a) hyperterminal,
// and b) GUI Composer (Temp Sensor) using a timer interrupt every 0.5 secs

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

```
#include "driverlib/timer.h"
#include "driverlib/debug.h"

/////////////////////////////////////////
void PrintUART(void);
void configTimer1A(void);
void convertUARTtemp(uint32_t);
void UART_OutChar(char);
//////////////////////////////////////////

//////////////////////////////////////////
uint32_t halfPeriod;
uint32_t ui32ADC0Value[1];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
//////////////////////////////////////////


// 1 clock cycle = 1 / SysCtlClockGet() second
// 1 SysCtlDelay = 3 clock cycle = 3 / SysCtlClockGet() second
// 1 second = SysCtlClockGet() / 3
// 0.001 second = 1 ms = SysCtlClockGet() / 3 / 1000
// so 0.5 seconds = SysCtkClockGet() / 3 / 2
// => 0.5 seconds = SysCtkClockGet() / (3*2)


int main(void)
{
    // set up system clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);
    // Enable the URAT0 and GPIOA peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // configure the pins for the receiver and transmitter using GPIOPinConfigure
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
//enable pin for LED
    //  Initialize the parameters for the UART: 115200, 8-1-N
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);// enable the ADC0 peripheral

    ADCHardwareOversampleConfigure(ADC0_BASE, 32); // hardware averaging

    //configure the ADC sequencer.
    //We want to use ADC0
    //sample sequencer 1
    // we want the processor to trigger the sequence and we want to use the highest
priority.
    //ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    //ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE,3,0,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    halfPeriod = SysCtlClockGet() / 2 ;

    configTimer1A();

    ADCIntEnable(ADC0_BASE,3);
    ADCSequenceEnable(ADC0_BASE,3);

    while (1)
    {
    }
}
void Timer1IntHandler(void)// add to startup_ccs
{
   // halfPeriod = 0.5 * (SysCtlClockGet());
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    TimerLoadSet(TIMER1_BASE, TIMER_A, halfPeriod);
    // The indication that the ADC conversion process is complete will be the ADC
interrupt status flag.
    ADCIntClear(ADC0_BASE, 3);
    // Trigger the ADC conversion with software
    ADCProcessorTrigger(ADC0_BASE, 3);
    // Wait for the conversion to complete.
    while(!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }
    // When code execution exits the loop in the previous step,
    // we know that the conversion is complete and that we can read the ADC value
from the ADC Sample Sequencer 1 FIFO.
    // The function we’ll be using copies data from the specified sample sequencer
output FIFO to a buffer in memory.
    // The number of samples available in the hardware FIFO are copied into the
buffer, which must be large enough to hold that many samples.
    // This will only return the samples that are presently available, which might
not be the entire sample sequence if you attempt to access the FIFO before the
conversion is complete.

    ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value);
    //  Calculate the average of the temperature sensor data.

    ui32TempAvg = ui32ADC0Value[0];

    // Calculate the Celsius value of the temperature.
    // TEMP = 147.5 – ((75 * (VREFP – VREFN) * ADCVALUE) / 4096)
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;

    // The conversion from Celsius to Fahrenheit is F = ( C * 9)/5 +32.
    // Adjusting that a little gives: F = ((C * 9) + 160) / 5
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

    PrintUART();
}


void configTimer1A(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER1_BASE, TIMER_A, halfPeriod);
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    TimerEnable(TIMER1_BASE, TIMER_A);

    IntMasterEnable(); //enable processor interrupts
}

void PrintUART()
{
    //UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX
interrupts
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'm');
    UARTCharPut(UART0_BASE, 'p');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');
    convertUARTtemp(ui32TempValueF);
    UARTCharPut(UART0_BASE, 'F');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');

}
void convertUARTtemp(uint32_t tempF)
{
    if (tempF >= 10)
    {
        convertUARTtemp(tempF/10);
        tempF %= 10;
    }
    UART_OutChar(tempF + '0');
}

void UART_OutChar(char val)
{
    // Input: letter is an 8-bit ASCII character to be transferred
    // Output: 8-bit to serial port
    while((UART0_FR_R & UART_FR_TXFF) != 0);
    UART0_DR_R = val;
}
```

Task 02:

Youtube Link: https://youtu.be/VLFOhJHsUIk

Screenshot:

Terminal	Board
	

```
void Blue_ON(void)
{
    UARTCharPut(UART0_BASE, 'B');
    UARTCharPut(UART0_BASE, 'l');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'n');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, BLUE)
    SysCtlDelay(100000);
    PrintCommand();
}
void Green_ON(void)
```

```
Enter Key Cmd: B
Blue On
Enter Key Cmd: T
Temperture: 75F
Enter Key Cmd: █
```

Modified Code:

```
// Ricky Perez
// CpE 403
// Lab 7
// Task 02

// Interaction/User Interface: Develop a user interface using UART to perform the following:
// Enter the cmd: R: Red LED, G: Green LED, B: Blue LED,
// T: Temperature: Based on the command (cmd) the program should
// turn ON Red LED when R is entered in the terminal, etc.
// Command of ‘r’ will turn off the Red LED.

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
```

```
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/timer.h"
#include "driverlib/debug.h"
////////////////////////////////////////
#define RED     2 // Pin 1
#define BLUE    4 // Pin 2
#define GREEN   8 // Pin 3
////////////////////////////////////////

/////////////////////////////////////////
void PrintCommand(void);
void Red_ON(void);
void Blue_ON(void);
void Green_ON(void);

void Red_OFF(void);
void Blue_OFF(void);
void Green_OFF(void);

void PrintTemp(void);
void convertUARTtemp(uint32_t);
void UART_OutChar(char);
//////////////////////////////////////////

//////////////////////////////////////////
uint32_t halfPeriod;
uint32_t ui32ADC0Value[1];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
char key;
//////////////////////////////////////////


// 1 clock cycle = 1 / SysCtlClockGet() second
// 1 SysCtlDelay = 3 clock cycle = 3 / SysCtlClockGet() second
// 1 second = SysCtlClockGet() / 3
// 0.001 second = 1 ms = SysCtlClockGet() / 3 / 1000
// so 0.5 seconds = SysCtkClockGet() / 3 / 2
// => 0.5 seconds = SysCtkClockGet() / (3*2)


int main(void)
{
    // set up system clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);
    // Enable the URAT0 and GPIOA peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // configure the pins for the receiver and transmitter using GPIOPinConfigure
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
//enable pin for LED
    //  Initialize the parameters for the UART: 115200, 8-1-N
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);// enable the ADC0 peripheral

    ADCHardwareOversampleConfigure(ADC0_BASE, 32); // hardware averaging

    //configure the ADC sequencer.
    //We want to use ADC0
    //sample sequencer 1
    // we want the processor to trigger the sequence and we want to use the highest
priority.
    //ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    //ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE,3,0,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    //halfPeriod = SysCtlClockGet() / 2 ;

    IntMasterEnable(); //enable processor interrupts
    IntEnable(INT_UART0); //enable the UART interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX
interrupts
    ADCIntEnable(ADC0_BASE,3);
    ADCSequenceEnable(ADC0_BASE,3);
    PrintCommand();

    while (1)
    {
    }
}

void UARTIntHandler(void) // add to startup_ccs
{
    uint32_t ui32Status;
    ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status
    UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

    key = UARTCharGet(UART0_BASE);
    UARTCharPut(UART0_BASE, key);
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');
    switch(key)
    {
    case 'R' :
        Red_ON();
        break;
    case 'B' :
        Blue_ON();
        break;
    case 'G' :
        Green_ON();
        break;

    case 'r' :
        Red_OFF();
        break;
    case 'b' :
        Blue_OFF();
        break;
    case 'g' :
        Green_OFF();
        break;
    case 'T' :
        PrintTemp();
        break;
    default :
        PrintCommand();

    }
}

void PrintCommand(void)
{

    UARTCharPut(UART0_BASE, 'E');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'K');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'y');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'C');
    UARTCharPut(UART0_BASE, 'm');
    UARTCharPut(UART0_BASE, 'd');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');

}
void Red_ON(void)
{
    UARTCharPut(UART0_BASE, 'R');
    UARTCharPut(UART0_BASE, 'e');
```

```c
    UARTCharPut(UART0_BASE, 'd');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, RED); //blink LED
    SysCtlDelay(100000);
    PrintCommand();
}
void Blue_ON(void)
{
    UARTCharPut(UART0_BASE, 'B');
    UARTCharPut(UART0_BASE, 'l');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'n');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, BLUE); //blink LED
    SysCtlDelay(100000);
    PrintCommand();
}
void Green_ON(void)
{
    UARTCharPut(UART0_BASE, 'G');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'n');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GREEN); //blink LED
    SysCtlDelay(100000);
    PrintCommand();
}

void Red_OFF(void)
{
    UARTCharPut(UART0_BASE, 'R');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'd');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, 'f');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0); //turn off LED
    SysCtlDelay(100000);
    PrintCommand();
}
void Blue_OFF(void)
{
    UARTCharPut(UART0_BASE, 'B');
    UARTCharPut(UART0_BASE, 'l');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, 'f');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
    SysCtlDelay(100000);
    PrintCommand();
}

void Green_OFF(void)
{
    UARTCharPut(UART0_BASE, 'G');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, 'f');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0); //turn off LED
    SysCtlDelay(100000);
    PrintCommand();
}
void PrintTemp(void)
{
    ADCIntClear(ADC0_BASE, 3);
    // Trigger the ADC conversion with software
    ADCProcessorTrigger(ADC0_BASE, 3);
    // Wait for the conversion to complete.
    while(!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }
    // When code execution exits the loop in the previous step,
    // we know that the conversion is complete and that we can read the ADC value 
from the ADC Sample Sequencer 1 FIFO.
    // The function we’ll be using copies data from the specified sample sequencer 
output FIFO to a buffer in memory.
    // The number of samples available in the hardware FIFO are copied into the 
buffer, which must be large enough to hold that many samples.
    // This will only return the samples that are presently available, which might 
not be the entire sample sequence if you attempt to access the FIFO before the 
conversion is complete.

    ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value);
    //  Calculate the average of the temperature sensor data.

    ui32TempAvg = ui32ADC0Value[0];

    // Calculate the Celsius value of the temperature.
    // TEMP = 147.5 – ((75 * (VREFP – VREFN) * ADCVALUE) / 4096)
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;

    // The conversion from Celsius to Fahrenheit is F = ( C * 9)/5 +32.
    // Adjusting that a little gives: F = ((C * 9) + 160) / 5

    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

    //UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX 
interrupts
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'm');
    UARTCharPut(UART0_BASE, 'p');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');
    convertUARTtemp(ui32TempValueF);
    UARTCharPut(UART0_BASE, 'F');
    // linefeed
    UARTCharPut(UART0_BASE, '\n');
    // carriage-return
    UARTCharPut(UART0_BASE, '\r');
    PrintCommand();

}

void convertUARTtemp(uint32_t tempF)
{
    if (tempF >= 10)
    {
        convertUARTtemp(tempF/10);
        tempF %= 10;
    }
    UART_OutChar(tempF + '0');
}

void UART_OutChar(char val)
{
    // Input: letter is an 8-bit ASCII character to be transferred
    // Output: 8-bit to serial port
    while((UART0_FR_R & UART_FR_TXFF) != 0);
    UART0_DR_R = val;
}

-------------------------------------------------------------------------------------
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.