**Date Submitted: 10/28/2018**

**Assignment Youtube Playlist:** **https://www.youtube.com/playlist?list=PL4oTyvRrubXdlUTVzMETSIC4nmOH8Xt4E**

# Task 00: Execute the supplied code, display the temperatures in the built-in Graph Tool.

For Task 00, I have taken the included code for Lab 07 and modified it to include our internal temperature sensor ADC. We then run our code, which is now simultaneously reading/echoing UART characters and recording temperature information.

**Youtube Link:** **https://youtu.be/MSgvZxrcGIU**

**Original Code (added comments):**

**#include** <stdint.h>

**#include** <stdbool.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/adc.h"

**#include** "driverlib/timer.h"

//Include all referenced functions.

//This is our interrupt handler for UART, triggered on character receipt (or FIFO level reached if enabled)

//or on timeout if another character isn't received in a 32-bit period.

**void** **UARTIntHandler**(**void**)

{

uint32\_t ui32Status;

ui32Status = **UARTIntStatus**(UART0\_BASE, true); //get interrupt status

**UARTIntClear**(UART0\_BASE, ui32Status); //Clear the asserted interrupts

**while**(**UARTCharsAvail**(UART0\_BASE)) //loop while there are chars

{

**UARTCharPutNonBlocking**(UART0\_BASE, **UARTCharGetNonBlocking**(UART0\_BASE)); //echo character

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2); //blink LED

**SysCtlDelay**(**SysCtlClockGet**() / (1000 \* 3)); //delay ~1msec

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0); //turn off LED

}

}

//Declaring our temperature variables as global for reference by other tools.

uint32\_t ui32ADC0Value[8];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**int** **main**(**void**) {

//Set up our system clock, 40MHz.

**SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

//Enable UART0 and GPIOA peripherals. GPIO A is where the UART TX/RX are located.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

//Configure pins A0 and A1 for UART.

**GPIOPinConfigure**(GPIO\_PA0\_U0RX);

**GPIOPinConfigure**(GPIO\_PA1\_U0TX);

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//Enable GPIOF, for on-board LED usage.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

//Configure our UART clock based on system clock and desired baud rate of 115,200.

**UARTConfigSetExpClk**(UART0\_BASE, **SysCtlClockGet**(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

//Enable all interrupts.

**IntMasterEnable**();

**IntEnable**(INT\_UART0);

**UARTIntEnable**(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

//Register our UART interrupt handler so it can be used.

**UARTIntRegister**(UART0\_BASE, UARTIntHandler);

//Enable ADC peripheral.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //Enable clock to ADC0.

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

//Configure hardware averaging of ADC0, 32 samples.

**ADCSequenceConfigure**(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

//ADC0, Sample Sequencer 0, triggered by processor, highest priority (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);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 4, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 5, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 6, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 7, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 0);

//Configure the eight steps of our sequencer. All using ADC0 and sequencer 0, measuring the internal temperature

//sensor, then triggering the interrupt flag when finished and signal the last conversion on sequencer 0.

//Enable ADC after all eight steps are set.

**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, 'T');

**UARTCharPut**(UART0\_BASE, 'e');

**UARTCharPut**(UART0\_BASE, 'x');

**UARTCharPut**(UART0\_BASE, 't');

**UARTCharPut**(UART0\_BASE, ':');

**UARTCharPut**(UART0\_BASE, ' ');

//Transmit the phrase "Enter Text: " one character at a time.

**while** (1)

{

**if** (**UARTCharsAvail**(UART0\_BASE)) **UARTCharPut**(UART0\_BASE, **UARTCharGet**(UART0\_BASE)); //If there is a character available, retrieve and echo it.

**ADCIntClear**(ADC0\_BASE, 0); //Clear ADC interrupt flag.

**ADCProcessorTrigger**(ADC0\_BASE, 0); //Trigger ADC conversion with the processor.

**while**(!**ADCIntStatus**(ADC0\_BASE, 0, false)) //Wait for ADC conversion to complete.

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 0, ui32ADC0Value); //After conversion is complete, pull the data into the array.

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + ui32ADC0Value[4] + ui32ADC0Value[5] + ui32ADC0Value[6] + ui32ADC0Value[7] + 4) /8;

//Average the eight array temperatures, adding +4 to compensate for rounding due to integer math.

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10; //Calculate degrees C.

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5; //Calculate degrees F.

}

}

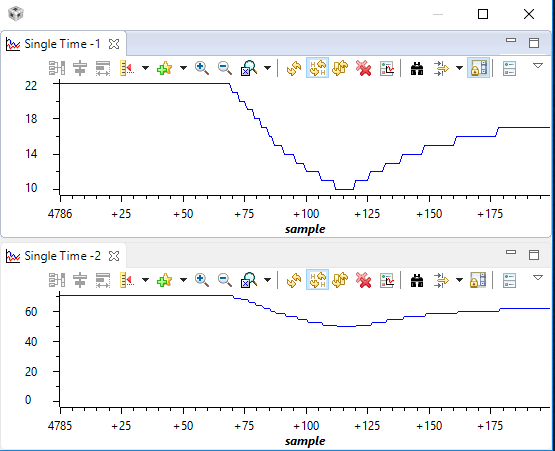


Figure - Temperature change

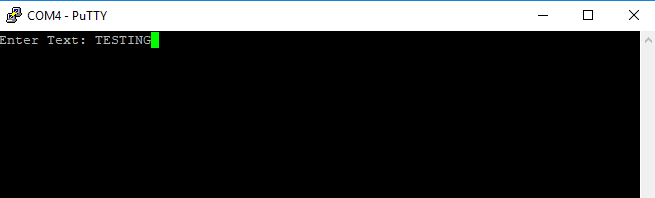


Figure - PuTTY terminal greeting and echo

**-----------------------------------------------------------------------------------**

# Task 01: Task 01: 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.

**Youtube Link:** **https://youtu.be/51ZpNoMQ2x4**

For Task 01, I adjusted the code to include a timer as well as added additional formatting to display our current temperature in Fahrenheit and Celsius. Unfortunately, GUI composer drivers would not install, per Dr. Venki I have opted to skip this part of the task.

**#include** <stdint.h>

**#include** <stdbool.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/debug.h"

**#include** "driverlib/uart.h"

**#include** "inc/hw\_ints.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/adc.h"

**#include** "driverlib/timer.h"

**#include** "utils/uartstdio.h"

**#include** "driverlib/timer.h"

//Include all referenced functions.

**#define** UART\_BUFFERED

**#define** baudrate 115200

//#define baudrate 9600

**#ifdef** DEBUG //Record file name and line number of any errors from a library API with incorrect parameters.

**void** **\_\_error\_\_**(**char** \*pcFilename, uint32\_t ui32Line)

{

}

**#endif**

//This is our interrupt handler for UART, triggered on character receipt (or FIFO level reached if enabled)

//or on timeout if another character isn't received in a 32-bit period.

**void** **UARTIntHandler**(**void**)

{

uint32\_t ui32Status;

ui32Status = **UARTIntStatus**(UART0\_BASE, true); //get interrupt status

**UARTIntClear**(UART0\_BASE, ui32Status); //Clear the asserted interrupts

**while**(**UARTCharsAvail**(UART0\_BASE)) //loop while there are chars

{

**UARTCharPutNonBlocking**(UART0\_BASE, **UARTCharGetNonBlocking**(UART0\_BASE)); //echo character

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2); //blink LED

**SysCtlDelay**(**SysCtlClockGet**() / (1000 \* 3)); //delay ~1msec

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0); //turn off LED

}

}

//Declaring our temperature variables as global for reference by other tools.

uint32\_t ui32ADC0Value[8];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**int** **main**(**void**)

{

//Set up our system clock, 40MHz.

**SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

//Enable UART0 and GPIOA peripherals. GPIO A is where the UART TX/RX are located.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

//Configure pins A0 and A1 for UART.

**GPIOPinConfigure**(GPIO\_PA0\_U0RX);

**GPIOPinConfigure**(GPIO\_PA1\_U0TX);

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//Enable GPIOF, for on-board LED usage.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

//Configure our UART clock based on system clock and desired baud rate, defined above.

**UARTStdioConfig**(0, baudrate, **SysCtlClockGet**());

//Enable all interrupts.

**IntMasterEnable**();

**IntEnable**(INT\_UART0);

**UARTIntEnable**(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

//Register our UART interrupt handler so it can be used.

**UARTIntRegister**(UART0\_BASE, UARTIntHandler);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //Enable clock to ADC0.

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

//Configure hardware averaging of ADC0, 32 samples.

**ADCSequenceConfigure**(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

//ADC0, Sample Sequencer 0, triggered by processor, highest priority (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);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 4, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 5, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 6, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 7, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 0);

//Configure the eight steps of our sequencer. All using ADC0 and sequencer 0, measuring the internal temperature

//sensor, then triggering the interrupt flag when finished and signal the last conversion on sequencer 0.

//Enable ADC after all eight steps are set.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); //Enable Timer 1 peripheral.

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC); //Configure timer 1 to be periodic.

uint32\_t ui32Clock = **SysCtlClockGet**(); //Retrieve system clock and store in variable.

uint32\_t ui32TimerDelay = (ui32Clock / 2); //Trigger at 2Hz, 0.5 second period.

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32TimerDelay -1); //Load button on-period into timer 1.

**IntEnable**(INT\_TIMER1A); //Enable vector associated with Timer 1A.

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT); //Enable interrupt from Timer 1A.

**IntMasterEnable**();

**TimerEnable**(TIMER1\_BASE, TIMER\_A); //Enable and start timer 1A.

**while**(1)

{

}

}

**void** **Timer1IntHandler**(**void**)

{

// Clear the timer interrupt

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

**ADCIntClear**(ADC0\_BASE, 0); //Clear ADC interrupt flag.

**ADCProcessorTrigger**(ADC0\_BASE, 0); //Trigger ADC conversion with the processor.

**while**(!**ADCIntStatus**(ADC0\_BASE, 0, false)) //Wait for ADC conversion to complete.

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 0, ui32ADC0Value); //After conversion is complete, pull the data into the array.

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + ui32ADC0Value[4] + ui32ADC0Value[5] + ui32ADC0Value[6] + ui32ADC0Value[7] + 4) /8;

//Average the eight array temperatures, adding +4 to compensate for rounding due to integer math.

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10; //Calculate degrees C.

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5; //Calculate degrees F.

**UARTprintf**("\f\n");

**UARTprintf**("Current Device Temperature in Celsius: %i\n", ui32TempValueC);

**UARTprintf**("Current Device Temperature in Fahrenheit: %i\n", ui32TempValueF);

//UARTprintf("Baud Rate: %i\n", baudrate);

}

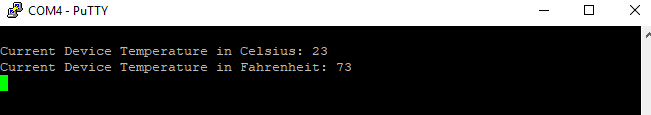


Figure - PuTTY displaying UART Celsius and Fahrenheit temperature readout

**-----------------------------------------------------------------------------------**

# Task 02: 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

**Youtube Link:** **https://youtu.be/9mHFDdCaSlU**

In this task I have modified the code to compare to a list of instructions. Pressing RGB in capitals will turn the respective LED on, and rgb in lowercase will turn them off. Pressing T or t will display the current temperature, which I have created a separate function for and is now calculated on-demand. As we are dealing with slow input signals, this has no noticeable delay and only has the ADC calculate the temperature when we request it.

**#include** <stdint.h>

**#include** <stdbool.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/debug.h"

**#include** "driverlib/uart.h"

**#include** "inc/hw\_ints.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/adc.h"

**#include** "utils/uartstdio.h"

//Include all referenced functions.

**#define** UART\_BUFFERED

**#define** baudrate 115200

//#define baudrate 9600

**#ifdef** DEBUG //Record file name and line number of any errors from a library API with incorrect parameters.

**void** **\_\_error\_\_**(**char** \*pcFilename, uint32\_t ui32Line)

{

}

**#endif**

//Declaring our temperature variables as global for reference by other tools.

uint32\_t ui32ADC0Value[8];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

//Creating a temperature calculation function to be called on-demand.

//As we are dealing with slow signals, no noticeable delay is perceived.

//We only take the ADC value of the temperature when we need it.

**void** **TemperatureCalculation**(**void**)

{

**ADCProcessorTrigger**(ADC0\_BASE, 0); //Trigger ADC conversion with the processor.

**while**(!**ADCIntStatus**(ADC0\_BASE, 0, false)) //Wait for ADC conversion to complete.

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 0, ui32ADC0Value); //After conversion is complete, pull the data into the array.

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + ui32ADC0Value[4] + ui32ADC0Value[5] + ui32ADC0Value[6] + ui32ADC0Value[7] + 4) /8;

//Average the eight array temperatures, adding +4 to compensate for rounding due to integer math.

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10; //Calculate degrees C.

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5; //Calculate degrees F.

}

//This is our interrupt handler for UART, triggered on character receipt (or FIFO level reached if enabled)

//or on timeout if another character isn't received in a 32-bit period.

**void** **UARTIntHandler**(**void**)

{

uint32\_t ui32Status;

uint8\_t ReadCharacter;

ui32Status = **UARTIntStatus**(UART0\_BASE, true); //get interrupt status

**UARTIntClear**(UART0\_BASE, ui32Status); //Clear the asserted interrupts

**while**(**UARTCharsAvail**(UART0\_BASE)) //loop while there are chars

{

ReadCharacter = **UARTCharGetNonBlocking**(UART0\_BASE); //Read our input character from UART.

**UARTprintf**("\f");

**UARTprintf**("Enter the command: R: Red LED, G: Green LED, B: Blue LED, T: Temperature \n");

**UARTprintf**("Upper-case turns LED on, lower-case turns LED off. Enter selection: \n");

//Check if the input character matches any of the instructions and perform the desired action.

**if**(ReadCharacter == 'R')

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, GPIO\_PIN\_1); //turn on LED

**UARTprintf**("Red LED on. \n");

}

**else** **if**(ReadCharacter == 'r')

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0); //turn off LED

**UARTprintf**("Red LED off. \n");

}

**else** **if**(ReadCharacter == 'G')

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, GPIO\_PIN\_3); //turn on LED

**UARTprintf**("Green LED on. \n");

}

**else** **if**(ReadCharacter == 'g')

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0); //turn off LED

**UARTprintf**("Green LED off. \n");

}

**else** **if**(ReadCharacter == 'B')

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2); //turn on LED

**UARTprintf**("Blue LED on. \n");

}

**else** **if**(ReadCharacter == 'b')

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0); //turn off LED

**UARTprintf**("Blue LED off. \n");

}

**else** **if**(ReadCharacter == 't'|| ReadCharacter == 'T')

{

TemperatureCalculation(); //Call our ADC function to check temperature then display.

**UARTprintf**("\n");

**UARTprintf**("Current Device Temperature in Celsius: %i\n", ui32TempValueC);

**UARTprintf**("Current Device Temperature in Fahrenheit: %i\n", ui32TempValueF);

}

**else**

{

**UARTprintf**("\n Invalid Selection.\n"); //No matching character sent, error detection.

}

}

}

**int** **main**(**void**)

{

//Set up our system clock, 40MHz.

**SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

//Enable UART0 and GPIOA peripherals. GPIO A is where the UART TX/RX are located.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

//Configure pins A0 and A1 for UART.

**GPIOPinConfigure**(GPIO\_PA0\_U0RX);

**GPIOPinConfigure**(GPIO\_PA1\_U0TX);

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//Enable GPIOF, for on-board LED usage.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

//Configure our UART clock based on system clock and desired baud rate, defined above.

**UARTStdioConfig**(0, baudrate, **SysCtlClockGet**());

//Enable all interrupts.

**IntMasterEnable**();

**IntEnable**(INT\_UART0);

**UARTIntEnable**(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

//Register our UART interrupt handler so it can be used.

**UARTIntRegister**(UART0\_BASE, UARTIntHandler);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //Enable clock to ADC0.

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

//Configure hardware averaging of ADC0, 32 samples.

**ADCSequenceConfigure**(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

//ADC0, Sample Sequencer 0, triggered by processor, highest priority (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);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 4, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 5, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 6, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 7, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 0);

//Configure the eight steps of our sequencer. All using ADC0 and sequencer 0, measuring the internal temperature

//sensor, then triggering the interrupt flag when finished and signal the last conversion on sequencer 0.

//Enable ADC after all eight steps are set.

**IntMasterEnable**();

//Print our initial instructions.

**UARTprintf**("\f");

**UARTprintf**("Enter the command: R: Red LED, G: Green LED, B: Blue LED, T: Temperature \n");

**UARTprintf**("Upper-case turns LED on, lower-case turns LED off. Enter selection:");

**while**(1) //Wait here until UART has input.

{

}

}

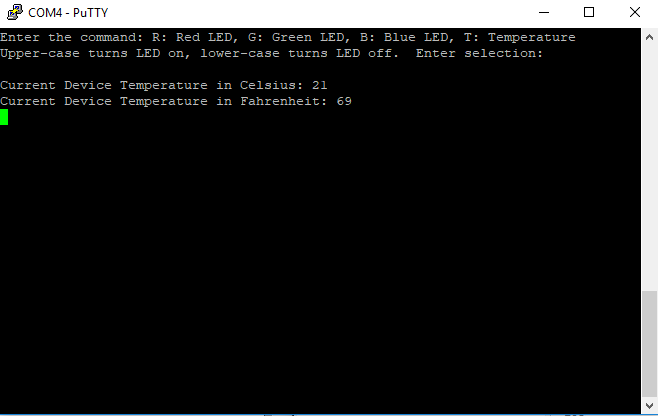


Figure - Temperature requested

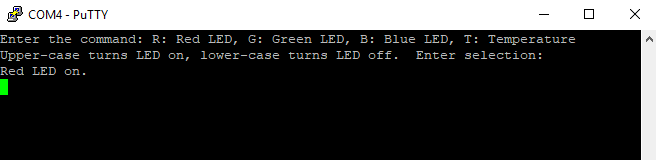


Figure - Red LED on requested

**------------------------------------------------------------------------------------**