**Date Submitted: October 13, 2019**

**Task 00: Execute provided code**

**Youtube Link: https://youtu.be/SXmPBXaJ5VU**

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

**Task 01:**

Youtube Link:

**Modified Schematic (if applicable):**

**Modified Code: https://youtu.be/Sppd1RpqFq0**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// temperature\_sensor.c - Example demonstrating the internal ADC temperature

// sensor.

//

// Copyright (c) 2010-2014 Texas Instruments Incorporated. All rights reserved.

// Software License Agreement

//

// Redistribution and use in source and binary forms, with or without

// modification, are permitted provided that the following conditions

// are met:

//

// Redistributions of source code must retain the above copyright

// notice, this list of conditions and the following disclaimer.

//

// Redistributions in binary form must reproduce the above copyright

// notice, this list of conditions and the following disclaimer in the

// documentation and/or other materials provided with the

// distribution.

//

// Neither the name of Texas Instruments Incorporated nor the names of

// its contributors may be used to endorse or promote products derived

// from this software without specific prior written permission.

//

// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

//

// This is part of revision 2.1.0.12573 of the Tiva Firmware Development Package.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* This code was made to show a simple ADC read.

\*

\* It was made from the example provided by TivaWare but it was a some modifications

\* like the math

\*

\*

\* Luís Afonso

\*

\*

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* This code has been modified to satisfy the requirements for CPE 403 Assignment 7, Task 1

\*

\*

\* Geovanni Portillo

\*

\*

\*/

//#define PART\_TM4C123GH6PM

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "stdlib.h"

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

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

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

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

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

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

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

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

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

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

**#include** "driverlib/pin\_map.h"

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

**#include** "driverlib/rom\_map.h"

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

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

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

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

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

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

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

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

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

**#include** <string.h>

**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 ~1 msec

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

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// This function sets up UART0 to be used for a console to display information

// as the example is running.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void**

**InitConsole**(**void**)

{

//

// Enable GPIO port A which is used for UART0 pins.

// **TODO**: change this to whichever GPIO port you are using.

//

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

//

// Configure the pin muxing for UART0 functions on port A0 and A1.

// This step is not necessary if your part does not support pin muxing.

// **TODO**: change this to select the port/pin you are using.

//

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

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

//

// Enable UART0 so that we can configure the clock.

//

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

//

// Use the internal 16MHz oscillator as the UART clock source.

//

**UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);

//

// Select the alternate (UART) function for these pins.

// **TODO**: change this to select the port/pin you are using.

//

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

//

// Initialize the UART for console I/O.

//

**UARTStdioConfig**(0, 115200, 16000000);

}

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

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ); //Changed from \_2\_5 to \_5 to match reused code from other assignments

InitConsole();

//LED Config

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

**SysCtlDelay**(3);

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

//

// Display the setup on the console.

//

**UARTprintf**("ADC ->\n");

**UARTprintf**(" Type: Internal Temperature Sensor\n");

**UARTprintf**(" Samples: One\n");

**UARTprintf**(" Update Rate: 250ms\n");

**UARTprintf**(" Input Pin: Internal temperature sensor\n\n");

//

// The ADC0 peripheral must be enabled for use.

//

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**SysCtlDelay**(3);

//

// Enable sample sequence 3 with a processor signal trigger. Sequence 3

// will do a single sample when the processor sends a singal to start the

// conversion. Each ADC module has 4 programmable sequences, sequence 0

// to sequence 3. This example is arbitrarily using sequence 3.

//

**ADCSequenceConfigure**(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

//

// Configure step 0 on sequence 3. Sample the temperature sensor

// (ADC\_CTL\_TS) and configure the interrupt flag (ADC\_CTL\_IE) to be set

// when the sample is done. Tell the ADC logic that this is the last

// conversion on sequence 3 (ADC\_CTL\_END). Sequence 3 has only one

// programmable step. Sequence 1 and 2 have 4 steps, and sequence 0 has

// 8 programmable steps. Since we are only doing a single conversion using

// sequence 3 we will only configure step 0. For more information on the

// ADC sequences and steps, reference the datasheet.

//

**ADCSequenceStepConfigure**(ADC0\_BASE, 3, 0, ADC\_CTL\_TS | ADC\_CTL\_IE |

ADC\_CTL\_END);

//

// Since sample sequence 3 is now configured, it must be enabled.

//

**ADCSequenceEnable**(ADC0\_BASE, 3);

//

// Clear the interrupt status flag. This is done to make sure the

// interrupt flag is cleared before we sample.

//

**ADCIntClear**(ADC0\_BASE, 3);

//

// Sample the temperature sensor forever. Display the value on the

// console.

//

//Timer1A Config

uint32\_t ui32Period;

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1);

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

ui32Period = (**SysCtlClockGet**() / 1) / 2; //To get period of 0.5s, divide system clock for 1s period and divide by 2 for 0.5s

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1);

**IntEnable**(INT\_TIMER1A);

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

**IntMasterEnable**();

**TimerEnable**(TIMER1\_BASE, TIMER\_A);

**while**(1)

{

}

}

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

{

//Used to check time between interrupts

**if**(**GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2))

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 0);

}

**else**

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4);

}

//

// This array is used for storing the data read from the ADC FIFO. It

// must be as large as the FIFO for the sequencer in use. This example

// uses sequence 3 which has a FIFO depth of 1. If another sequence

// was used with a deeper FIFO, then the array size must be changed.

//

uint32\_t ADCValues[1];

//

// These variables are used to store the temperature conversions for

// Celsius and Fahrenheit.

//

uint32\_t TempValueC ;

uint32\_t TempValueF ;

//

// Trigger the ADC conversion.

//

**ADCProcessorTrigger**(ADC0\_BASE, 3);

//

// Wait for conversion to be completed.

//

**while**(!**ADCIntStatus**(ADC0\_BASE, 3, false))

{

}

//

// Clear the ADC interrupt flag.

//

**ADCIntClear**(ADC0\_BASE, 3);

//

// Read ADC Value.

//

**ADCSequenceDataGet**(ADC0\_BASE, 3, ADCValues);

//

// Use non-calibrated conversion provided in the data sheet. I use floats in intermediate

// math but you could use intergers with multiplied by powers of 10 and divide on the end

// Make sure you divide last to avoid dropout.

//

TempValueC = (uint32\_t)(147.5 - ((75.0\*3.3 \*(**float**)ADCValues[0])) / 4096.0);

//

// Get Fahrenheit value. Make sure you divide last to avoid dropout.

//

TempValueF = ((TempValueC \* 9) + 160) / 5;

//

// Display the temperature value on the console.

//

**UARTprintf**("Temperature = %3d\*C or %3d\*F\r", TempValueC,

TempValueF);

//

// This function provides a means of generating a constant length

// delay. The function delay (in cycles) = 3 \* parameter. Delay

// 250ms arbitrarily.

//

**SysCtlDelay**(80000000 / 12);

}

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

**Task 02:**

Youtube Link: https://youtu.be/-T9GyBX5Z0o

**Modified Schematic (if applicable):**

**Modified Code:**

/\*

\* LAB07\_T02.c

\*

\* Created on: Oct 12, 2019

\* Author: gausp

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// temperature\_sensor.c - Example demonstrating the internal ADC temperature

// sensor.

//

// Copyright (c) 2010-2014 Texas Instruments Incorporated. All rights reserved.

// Software License Agreement

//

// Redistribution and use in source and binary forms, with or without

// modification, are permitted provided that the following conditions

// are met:

//

// Redistributions of source code must retain the above copyright

// notice, this list of conditions and the following disclaimer.

//

// Redistributions in binary form must reproduce the above copyright

// notice, this list of conditions and the following disclaimer in the

// documentation and/or other materials provided with the

// distribution.

//

// Neither the name of Texas Instruments Incorporated nor the names of

// its contributors may be used to endorse or promote products derived

// from this software without specific prior written permission.

//

// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

//

// This is part of revision 2.1.0.12573 of the Tiva Firmware Development Package.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* This code was made to show a simple ADC read.

\*

\* It was made from the example provided by TivaWare but it was a some modifications

\* like the math

\*

\*

\* Luís Afonso

\*

\*

\*/

//#define PART\_TM4C123GH6PM

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "stdlib.h"

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

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

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

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

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

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

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

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

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

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

**#include** "driverlib/pin\_map.h"

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

**#include** "driverlib/rom\_map.h"

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

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

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

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

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

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

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

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

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

**#include** <string.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// This function sets up UART0 to be used for a console to display information

// as the example is running.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void**

**InitConsole**(**void**)

{

//

// Enable GPIO port A which is used for UART0 pins.

// **TODO**: change this to whichever GPIO port you are using.

//

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

//

// Configure the pin muxing for UART0 functions on port A0 and A1.

// This step is not necessary if your part does not support pin muxing.

// **TODO**: change this to select the port/pin you are using.

//

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

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

//

// Enable UART0 so that we can configure the clock.

//

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

//

// Use the internal 16MHz oscillator as the UART clock source.

//

**UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);

//

// Select the alternate (UART) function for these pins.

// **TODO**: change this to select the port/pin you are using.

//

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

//

// Initialize the UART for console I/O.

//

**UARTStdioConfig**(0, 115200, 16000000);

}

**void** **returnTempUART**(**void**);

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

**SysCtlClockSet**(SYSCTL\_SYSDIV\_2\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

InitConsole();

//LED Config

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

**SysCtlDelay**(3);

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

//

// Display the setup on the console.

//

**UARTprintf**("UART and LED Demo\n");

**UARTprintf**("H: help, R: red, G: green, B: blue, T: temperature\n");

//

// The ADC0 peripheral must be enabled for use.

//

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**SysCtlDelay**(3);

//

// Enable sample sequence 3 with a processor signal trigger. Sequence 3

// will do a single sample when the processor sends a singal to start the

// conversion. Each ADC module has 4 programmable sequences, sequence 0

// to sequence 3. This example is arbitrarily using sequence 3.

//

**ADCSequenceConfigure**(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

//

// Configure step 0 on sequence 3. Sample the temperature sensor

// (ADC\_CTL\_TS) and configure the interrupt flag (ADC\_CTL\_IE) to be set

// when the sample is done. Tell the ADC logic that this is the last

// conversion on sequence 3 (ADC\_CTL\_END). Sequence 3 has only one

// programmable step. Sequence 1 and 2 have 4 steps, and sequence 0 has

// 8 programmable steps. Since we are only doing a single conversion using

// sequence 3 we will only configure step 0. For more information on the

// ADC sequences and steps, reference the datasheet.

//

**ADCSequenceStepConfigure**(ADC0\_BASE, 3, 0, ADC\_CTL\_TS | ADC\_CTL\_IE |

ADC\_CTL\_END);

//

// Since sample sequence 3 is now configured, it must be enabled.

//

**ADCSequenceEnable**(ADC0\_BASE, 3);

//

// Clear the interrupt status flag. This is done to make sure the

// interrupt flag is cleared before we sample.

//

**ADCIntClear**(ADC0\_BASE, 3);

//

// Sample the temperature sensor forever. Display the value on the

// console.

//

**char** command;

**while**(1)

{

**while**(**UARTCharsAvail**(UART0\_BASE))

{

command = **UARTCharGet**(UART0\_BASE);

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

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

**switch** (command)

{

**case** 'R' :

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 , GPIO\_PIN\_1);

**break**;

**case** 'r' :

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 , 0);

**break**;

**case** 'G' :

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3 , GPIO\_PIN\_3);

**break**;

**case** 'g' :

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3 , 0);

**break**;

**case** 'B' :

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

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

**break**;

**case** 'b' :

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2 , 0);

**break**;

**case** 'T' :

returnTempUART();

**default**:

**UARTprintf**("Invalid command entered. Please enter a following command\n");

**UARTprintf**("R: red, G: green, B: blue, T: temperature\n");

}

}

}

}

**void** **returnTempUART**(**void**) {

//

// This array is used for storing the data read from the ADC FIFO. It

// must be as large as the FIFO for the sequencer in use. This example

// uses sequence 3 which has a FIFO depth of 1. If another sequence

// was used with a deeper FIFO, then the array size must be changed.

//

uint32\_t ADCValues[1];

//

// These variables are used to store the temperature conversions for

// Celsius and Fahrenheit.

//

uint32\_t TempValueC ;

uint32\_t TempValueF ;

//

// Trigger the ADC conversion.

//

**ADCProcessorTrigger**(ADC0\_BASE, 3);

//

// Wait for conversion to be completed.

//

**while**(!**ADCIntStatus**(ADC0\_BASE, 3, false))

{

}

//

// Clear the ADC interrupt flag.

//

**ADCIntClear**(ADC0\_BASE, 3);

//

// Read ADC Value.

//

**ADCSequenceDataGet**(ADC0\_BASE, 3, ADCValues);

//

// Use non-calibrated conversion provided in the data sheet. I use floats in intermediate

// math but you could use intergers with multiplied by powers of 10 and divide on the end

// Make sure you divide last to avoid dropout.

//

TempValueC = (uint32\_t)(147.5 - ((75.0\*3.3 \*(**float**)ADCValues[0])) / 4096.0);

//

// Get Fahrenheit value. Make sure you divide last to avoid dropout.

//

TempValueF = ((TempValueC \* 9) + 160) / 5;

//

// Display the temperature value on the console.

//

**UARTprintf**("Temperature = %3d\*C or %3d\*F\r\n", TempValueC,

TempValueF);

//

// This function provides a means of generating a constant length

// delay. The function delay (in cycles) = 3 \* parameter. Delay

// 250ms arbitrarily.

//

**SysCtlDelay**(80000000 / 12);

}

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