**Date Submitted:**

**Task 01:**

Youtube Link: <https://youtu.be/PsIV7T8XpAw>

**Modified Code:**

**Added the temperature ADC code used in lab 5 and modified the number of SSI data to be sent from 3 to 2 (one for F and other for C).**

**#include** <stdbool.h>

**#include** <stdint.h>

**#include** <stdlib.h>

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

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

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

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

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

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

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

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

**#define** NUM\_SSI\_DATA 2

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

{

uint32\_t pui32DataTx[NUM\_SSI\_DATA];

uint32\_t pui32DataRx[NUM\_SSI\_DATA];

uint32\_t ui32Index;

// ADC Variables

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// System clock at 50 MHz

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

// The SSI0 peripheral and port A must be enabled for use.

// Enable the SSI0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_SSI0);

// The SSI0 peripheral is on Port A and pins 2,3,4 and 5.

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

// This function/s configures the pin muxing on port A pins 2,3,4 and 5

**GPIOPinConfigure**(GPIO\_PA2\_SSI0CLK);

**GPIOPinConfigure**(GPIO\_PA3\_SSI0FSS);

**GPIOPinConfigure**(GPIO\_PA4\_SSI0RX);

**GPIOPinConfigure**(GPIO\_PA5\_SSI0TX);

**GPIOPinTypeSSI**(GPIO\_PORTA\_BASE, GPIO\_PIN\_5 | GPIO\_PIN\_4 | GPIO\_PIN\_3 |GPIO\_PIN\_2);

**GPIOPinWrite**(GPIO\_PORTA\_BASE,GPIO\_PIN\_4,GPIO\_PIN\_4);

// Configure and enable the SSI port for SPI master mode.

**SSIClockSourceSet**(SSI0\_BASE,SSI\_CLOCK\_SYSTEM);

**SSIConfigSetExpClk**(SSI0\_BASE, **SysCtlClockGet**(), SSI\_FRF\_MOTO\_MODE\_0,SSI\_MODE\_MASTER, 1000000, 8);

**SSIEnable**(SSI0\_BASE);

// 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.

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

// Initialize the UART for console I/O.

UARTStdioConfig(0, 115200, 16000000);

// Enable the ADC module

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

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 64);

// ADC0, sample sequencer 2, processor triggers, highest priority

// Typo on Assignment PDF instruction for Task 01

// Task 00 already used SS1 whereas "Task01 should being using SS2"

**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);

**ADCSequenceEnable**(ADC0\_BASE, 2);

**while**(1)

{

**ADCIntClear**(ADC0\_BASE, 2); // Clear ADC interrupt flag

**ADCProcessorTrigger**(ADC0\_BASE, 2); // Trigger the ADC conversion

**while**(!**ADCIntStatus**(ADC0\_BASE, 2, false)); // Poll until conversion completes

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value); // Store the temperature value

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] // Average the sampled temperatures

+ ui32ADC0Value[2] + ui32ADC0Value[3]

+ 2) / 4;

// Convert to Celsius

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

// Convert to Fahrenheit

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

// Print the temperature to the terminal

UARTprintf("\033[2J");

UARTprintf("Measured temp: %d F; %d C\n", ui32TempValueF, ui32TempValueC);

// Place the two temperatures into an array that's to be transmitted

pui32DataTx[0] = ui32TempValueF;

pui32DataTx[1] = ui32TempValueC;

UARTprintf("SSI ->\n");

UARTprintf(" Mode: SPI\n");

UARTprintf(" Data: 8-bit\n\n");

UARTprintf("Sent:\n ");

// Transmission process

**while**(**SSIDataGetNonBlocking**(SSI0\_BASE, &pui32DataRx[0]));

**for**(ui32Index = 0; ui32Index < NUM\_SSI\_DATA; ui32Index++)

{

// Display the data that SSI is transferring.

UARTprintf(" %d ", pui32DataTx[ui32Index]);

**SSIDataPut**(SSI0\_BASE, pui32DataTx[ui32Index]);

}

**while**(**SSIBusy**(SSI0\_BASE));

// Receiving the data

UARTprintf("\nReceived:\n ");

**for**(ui32Index = 0; ui32Index < NUM\_SSI\_DATA; ui32Index++)

{

**SSIDataGet**(SSI0\_BASE, &pui32DataRx[ui32Index]);

// Since we are using 8-bit data, mask off the MSB.

pui32DataRx[ui32Index] &= 0x00FF;

// Display the data that SSI0 received.

UARTprintf(" %d ", pui32DataRx[ui32Index]);

}

}

**return** 0;

}

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

**Task 02:**

Youtube Link:

**Modified Schematic (if applicable):**

**Modified Code:**

**// Insert code here**

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

**Task 03:**

Youtube Link:

**Modified Schematic (if applicable):**

**Modified Code:**

**// Insert code here**

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