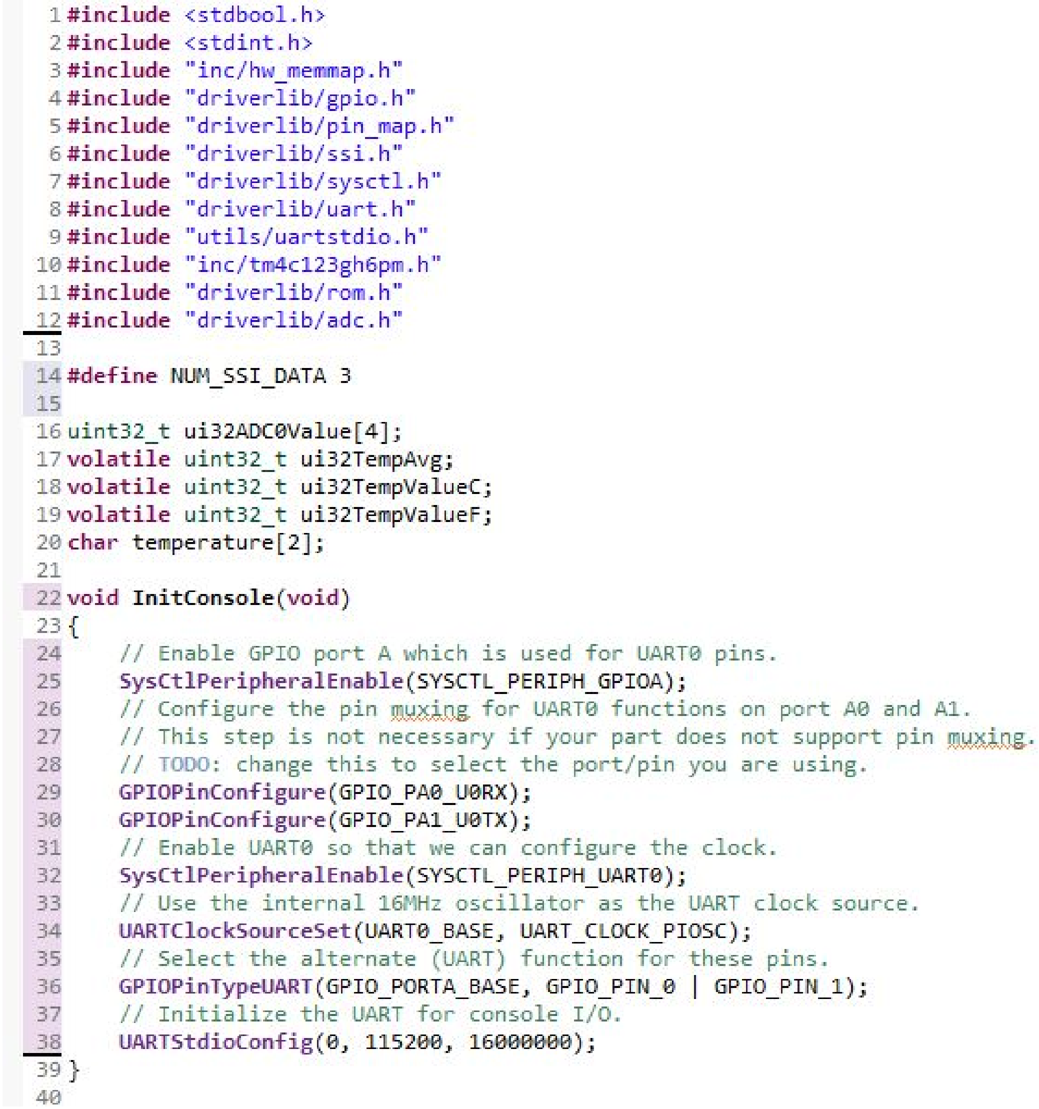
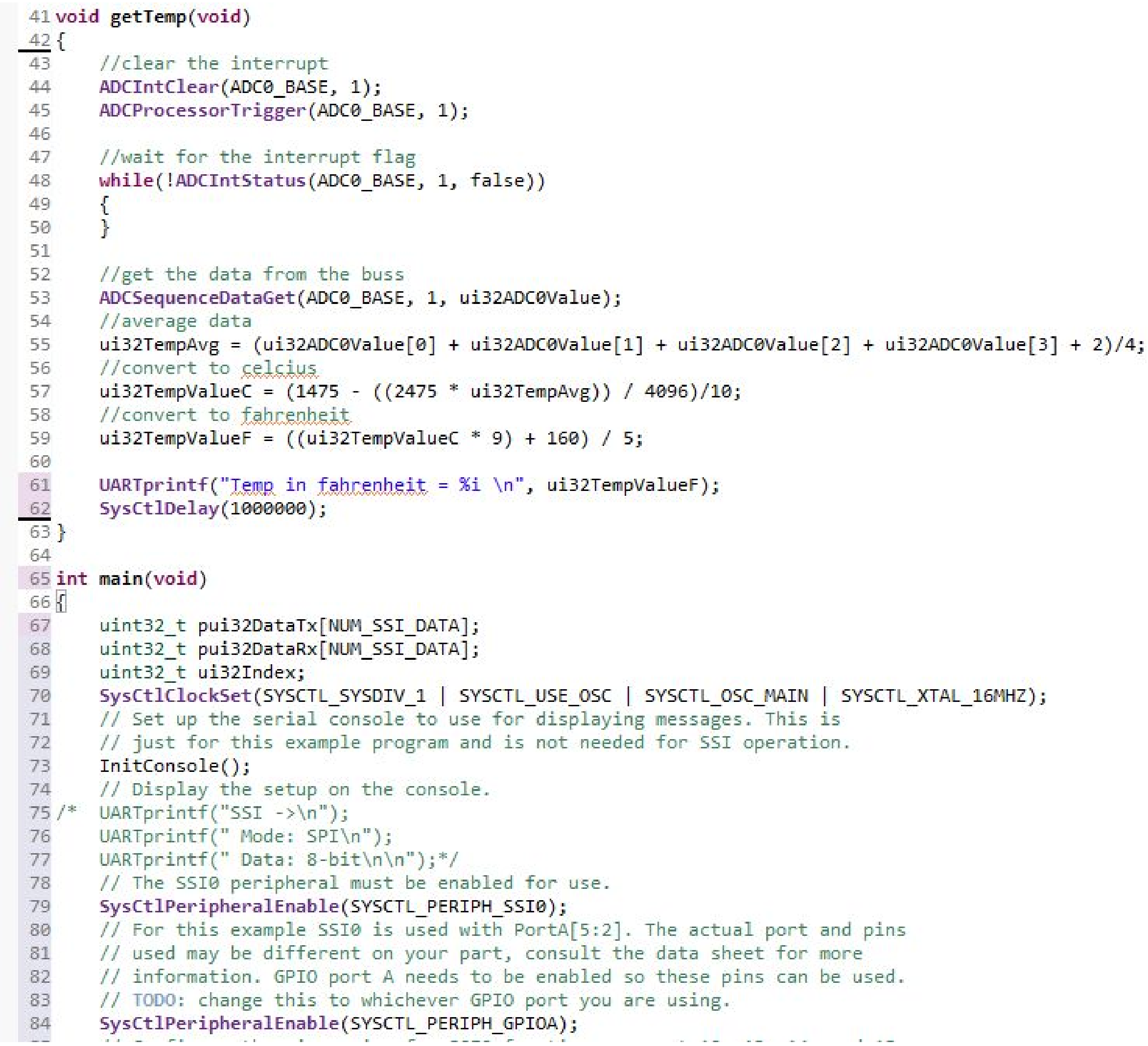
Joseph Sharp Halpin
CpE 403 Section 1001
10/16/2018

Task 01:
Youtube Link: https://youtu.be/ZxC-YBjpdb4

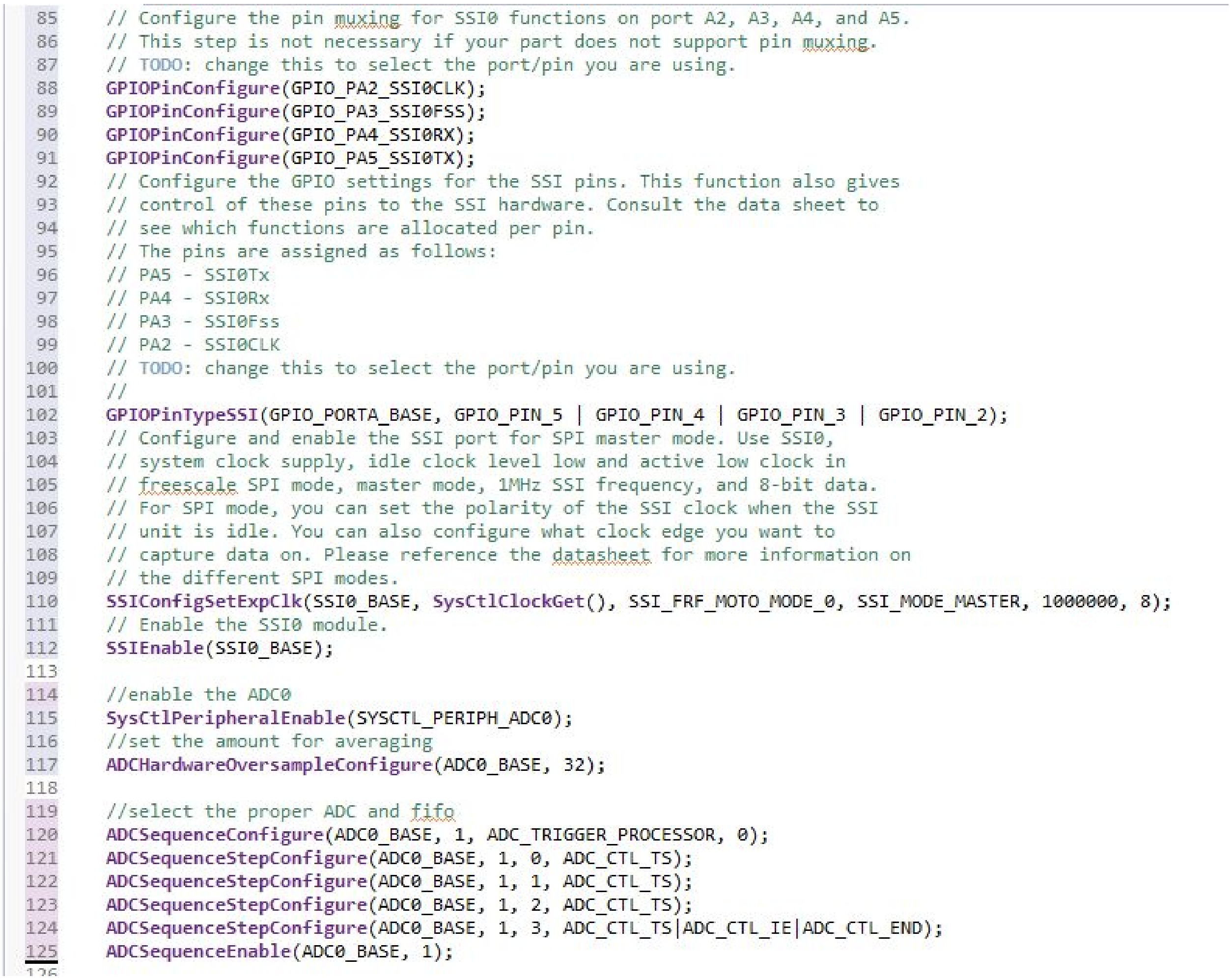
Code:



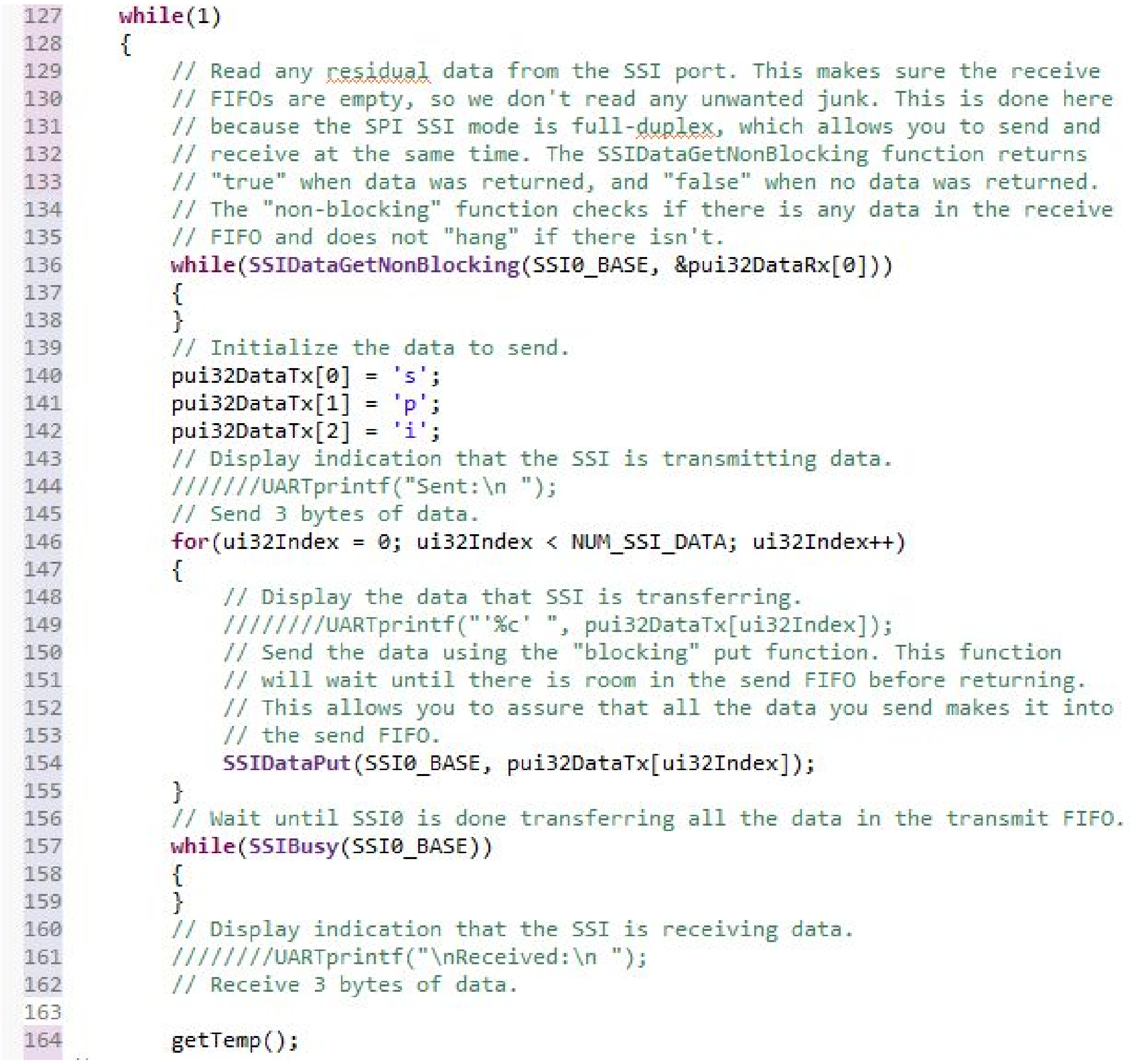
```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/ssi.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/rom.h"
#include "driverlib/adc.h"

#define NUM_SSI_DATA 3

uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
char temperature[2];

void InitConsole(void)
{
    // Enable GPIO port A which is used for UART0 pins.
    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.
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    // Initialize the UART for console I/O.
    UARTStdioConfig(0, 115200, 16000000);
}

```



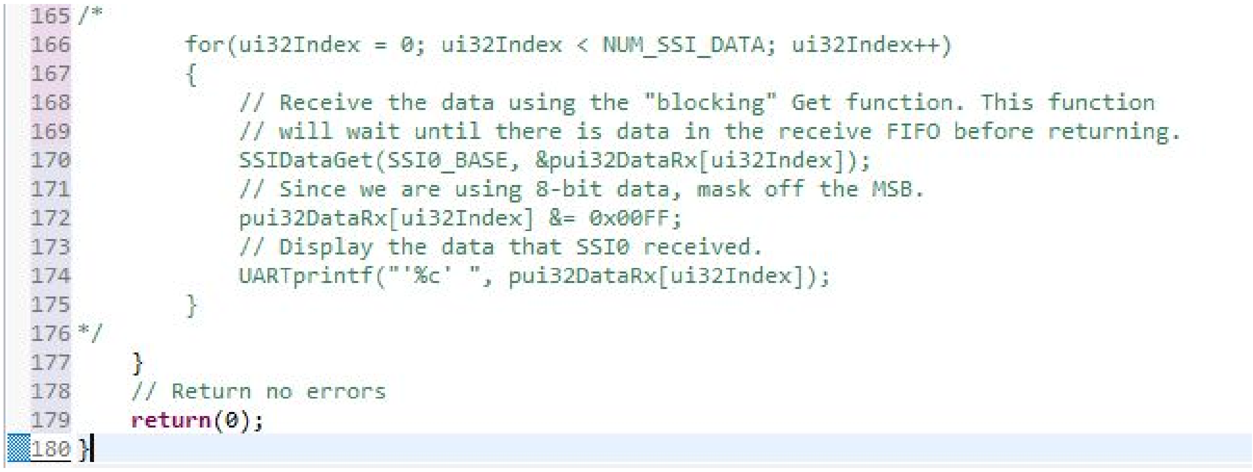
```c
void getTemp(void)
{
    //clear the interrupt
    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

    //wait for the interrupt flag
    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
    }

    //get the data from the buss
    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
    //average data
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
    //convert to celcius
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
    //convert to fahrenheit
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

    UARTprintf("Temp in fahrenheit = %i \n", ui32TempValueF);
    SysCtlDelay(1000000);
}

int main(void)
{
    uint32_t pui32DataTx[NUM_SSI_DATA];
    uint32_t pui32DataRx[NUM_SSI_DATA];
    uint32_t ui32Index;
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
    // Set up the serial console to use for displaying messages. This is
    // just for this example program and is not needed for SSI operation.
    InitConsole();
    // Display the setup on the console.
/*  UARTprintf("SSI ->\n");
    UARTprintf(" Mode: SPI\n");
    UARTprintf(" Data: 8-bit\n\n");*/
    // The SSI0 peripheral must be enabled for use.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
    // For this example SSI0 is used with PortA[5:2]. The actual port and pins
    // used may be different on your part, consult the data sheet for more
    // information. GPIO port A needs to be enabled so these pins can be used.
    // TODO: change this to whichever GPIO port you are using.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```



```
    // Configure the pin muxing for SSI0 functions on port A2, A3, A4, and A5.
    // 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_PA2_SSI0CLK);
    GPIOPinConfigure(GPIO_PA3_SSI0FSS);
    GPIOPinConfigure(GPIO_PA4_SSI0RX);
    GPIOPinConfigure(GPIO_PA5_SSI0TX);
    // Configure the GPIO settings for the SSI pins. This function also gives
    // control of these pins to the SSI hardware. Consult the data sheet to
    // see which functions are allocated per pin.
    // The pins are assigned as follows:
    // PA5 - SSI0Tx
    // PA4 - SSI0Rx
    // PA3 - SSI0Fss
    // PA2 - SSI0CLK
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2);
    // Configure and enable the SSI port for SPI master mode. Use SSI0,
    // system clock supply, idle clock level low and active low clock in
    // freescale SPI mode, master mode, 1MHz SSI frequency, and 8-bit data.
    // For SPI mode, you can set the polarity of the SSI clock when the SSI
    // unit is idle. You can also configure what clock edge you want to
    // capture data on. Please reference the datasheet for more information on
    // the different SPI modes.
    SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 1000000, 8);
    // Enable the SSI0 module.
    SSIEnable(SSI0_BASE);

    //enable the ADC0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    //set the amount for averaging
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);

    //select the proper ADC and fifo
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);
```



```
    while(1)
    {
        // Read any residual data from the SSI port. This makes sure the receive
        // FIFOs are empty, so we don't read any unwanted junk. This is done here
        // because the SPI SSI mode is full-duplex, which allows you to send and
        // receive at the same time. The SSIDataGetNonBlocking function returns
        // "true" when data was returned, and "false" when no data was returned.
        // The "non-blocking" function checks if there is any data in the receive
        // FIFO and does not "hang" if there isn't.
        while(SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[0]))
        {
        }
        // Initialize the data to send.
        pui32DataTx[0] = 's';
        pui32DataTx[1] = 'p';
        pui32DataTx[2] = 'i';
        // Display indication that the SSI is transmitting data.
        ///////UARTprintf("Sent:\n ");
        // Send 3 bytes of data.
        for(ui32Index = 0; ui32Index < NUM_SSI_DATA; ui32Index++)
        {
            // Display the data that SSI is transferring.
            ////////UARTprintf("'%c' ", pui32DataTx[ui32Index]);
            // Send the data using the "blocking" put function. This function
            // will wait until there is room in the send FIFO before returning.
            // This allows you to assure that all the data you send makes it into
            // the send FIFO.
            SSIDataPut(SSI0_BASE, pui32DataTx[ui32Index]);
        }
        // Wait until SSI0 is done transferring all the data in the transmit FIFO.
        while(SSIBusy(SSI0_BASE))
        {
        }
        // Display indication that the SSI is receiving data.
        ////////UARTprintf("\nReceived:\n ");
        // Receive 3 bytes of data.

        getTemp();
```

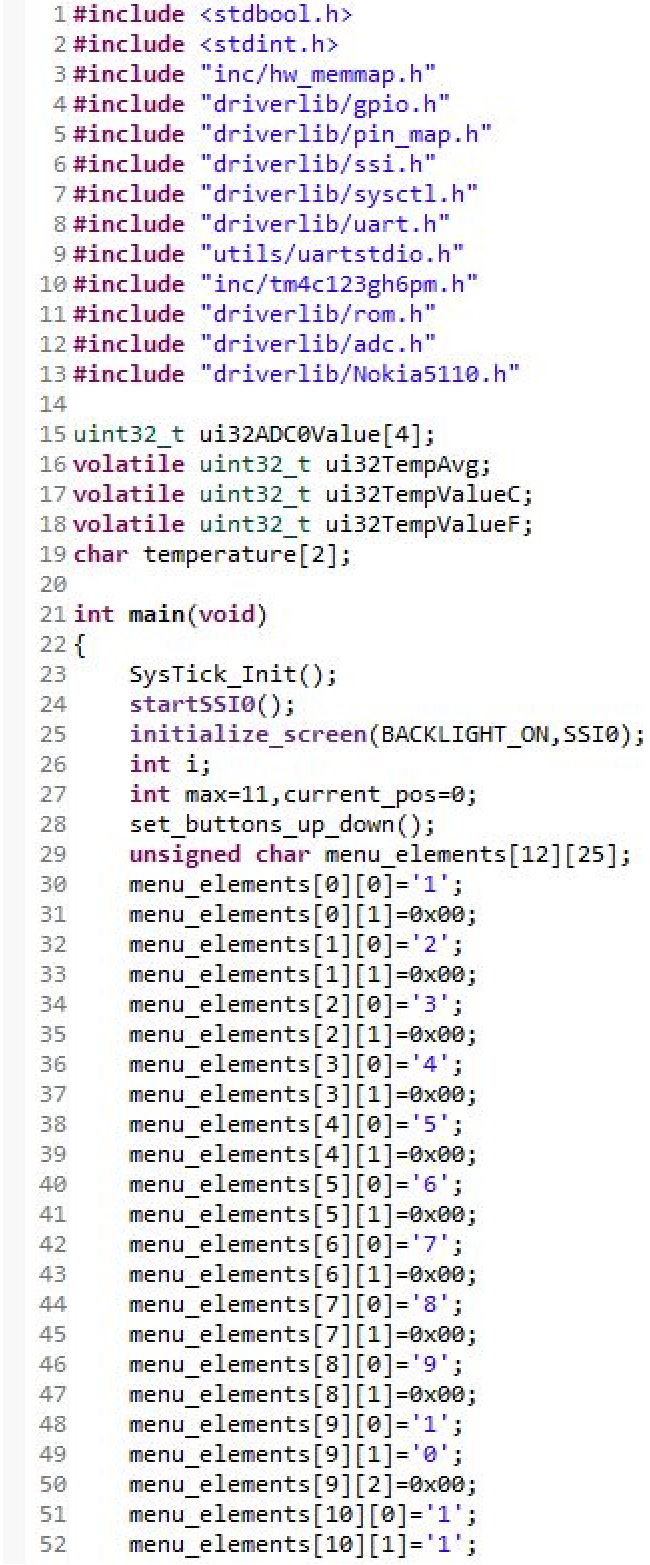


```
/*
        for(ui32Index = 0; ui32Index < NUM_SSI_DATA; ui32Index++)
        {
            // Receive the data using the "blocking" Get function. This function
            // will wait until there is data in the receive FIFO before returning.
            SSIDataGet(SSI0_BASE, &pui32DataRx[ui32Index]);
            // Since we are using 8-bit data, mask off the MSB.
            pui32DataRx[ui32Index] &= 0x00FF;
            // Display the data that SSI0 received.
            UARTprintf("'%c' ", pui32DataRx[ui32Index]);
        }
*/
    }
    // Return no errors
    return(0);
}
```

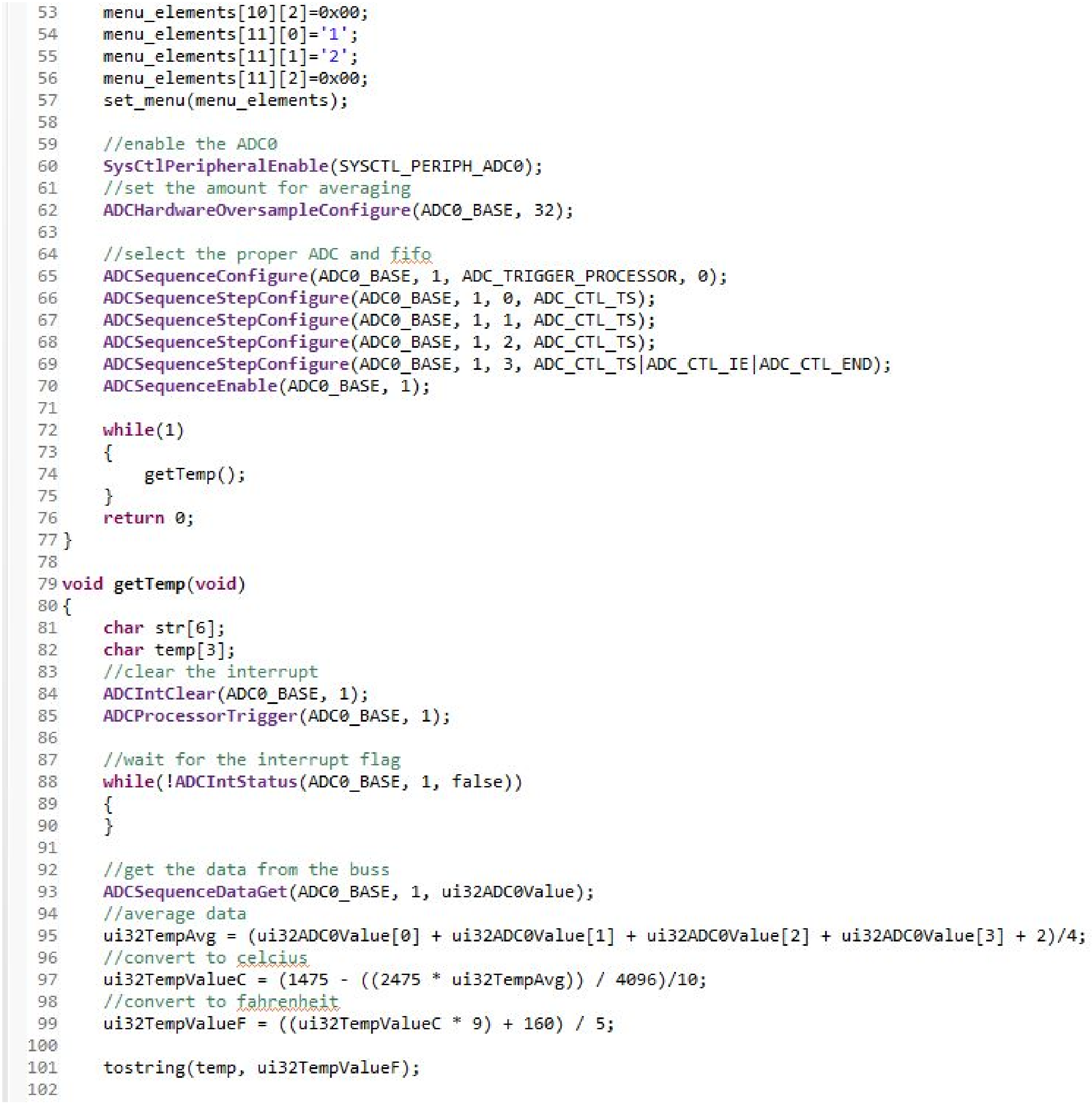
Task 02:

Youtube Link: https://youtu.be/c-wiUyGatB4

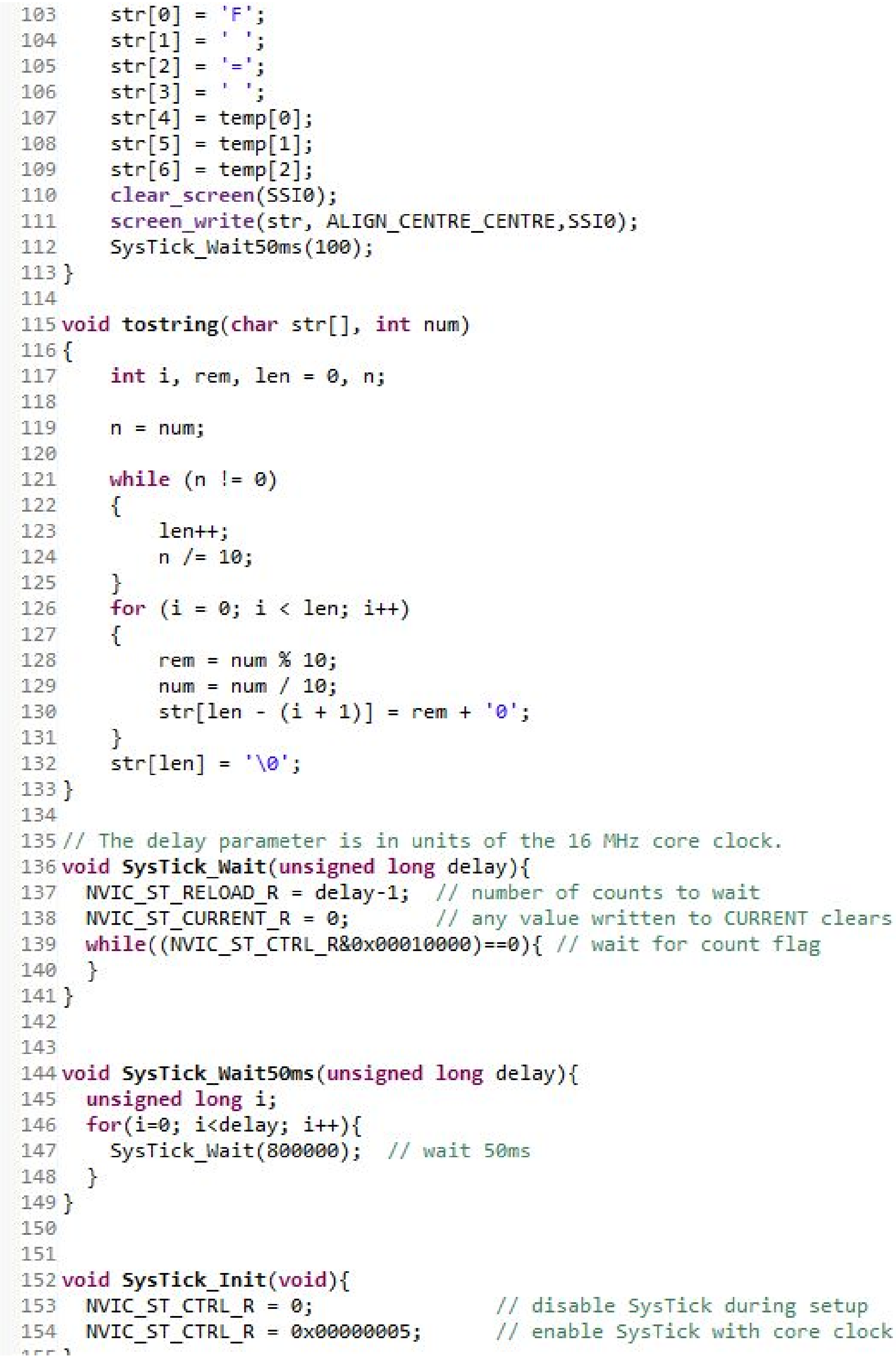
Code:



```c
#include <stdbool.h>
#include <stdint.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 "inc/tm4c123gh6pm.h"
#include "driverlib/rom.h"
#include "driverlib/adc.h"
#include "driverlib/Nokia5110.h"

uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
char temperature[2];

int main(void)
{
    SysTick_Init();
    startSSI0();
    initialize_screen(BACKLIGHT_ON,SSI0);
    int i;
    int max=11,current_pos=0;
    set_buttons_up_down();
    unsigned char menu_elements[12][25];
    menu_elements[0][0]='1';
    menu_elements[0][1]=0x00;
    menu_elements[1][0]='2';
    menu_elements[1][1]=0x00;
    menu_elements[2][0]='3';
    menu_elements[2][1]=0x00;
    menu_elements[3][0]='4';
    menu_elements[3][1]=0x00;
    menu_elements[4][0]='5';
    menu_elements[4][1]=0x00;
    menu_elements[5][0]='6';
    menu_elements[5][1]=0x00;
    menu_elements[6][0]='7';
    menu_elements[6][1]=0x00;
    menu_elements[7][0]='8';
    menu_elements[7][1]=0x00;
    menu_elements[8][0]='9';
    menu_elements[8][1]=0x00;
    menu_elements[9][0]='1';
    menu_elements[9][1]='0';
    menu_elements[9][2]=0x00;
    menu_elements[10][0]='1';
    menu_elements[10][1]='1';
```



```
    menu_elements[10][2]=0x00;
    menu_elements[11][0]='1';
    menu_elements[11][1]='2';
    menu_elements[11][2]=0x00;
    set_menu(menu_elements);

    //enable the ADC0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    //set the amount for averaging
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);

    //select the proper ADC and fifo
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);

    while(1)
    {
        getTemp();
    }
    return 0;
}

void getTemp(void)
{
    char str[6];
    char temp[3];
    //clear the interrupt
    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

    //wait for the interrupt flag
    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
    }

    //get the data from the buss
    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
    //average data
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
    //convert to celcius
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
    //convert to fahrenheit
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

    tostring(temp, ui32TempValueF);

```



```c
    str[0] = 'F';
    str[1] = ' ';
    str[2] = '=';
    str[3] = ' ';
    str[4] = temp[0];
    str[5] = temp[1];
    str[6] = temp[2];
    clear_screen(SSI0);
    screen_write(str, ALIGN_CENTRE_CENTRE,SSI0);
    SysTick_Wait50ms(100);
}

void tostring(char str[], int num)
{
    int i, rem, len = 0, n;

    n = num;

    while (n != 0)
    {
        len++;
        n /= 10;
    }
    for (i = 0; i < len; i++)
    {
        rem = num % 10;
        num = num / 10;
        str[len - (i + 1)] = rem + '0';
    }
    str[len] = '\0';
}

// The delay parameter is in units of the 16 MHz core clock.
void SysTick_Wait(unsigned long delay){
  NVIC_ST_RELOAD_R = delay-1;  // number of counts to wait
  NVIC_ST_CURRENT_R = 0;       // any value written to CURRENT clears
  while((NVIC_ST_CTRL_R&0x00010000)==0){ // wait for count flag
  }
}


void SysTick_Wait50ms(unsigned long delay){
  unsigned long i;
  for(i=0; i<delay; i++){
    SysTick_Wait(800000);  // wait 50ms
  }
}


void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;               // disable SysTick during setup
  NVIC_ST_CTRL_R = 0x00000005;      // enable SysTick with core clock
}
```