**Date Submitted: 10/9/18**

**Task 01:**

Youtube Link: <https://www.youtube.com/watch?v=dj9ZLjT-_1M>

**#include** <stdbool.h>

**#include** <stdint.h>

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

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

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

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

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

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

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

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

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

//! This example shows how to configure the SSI0 as SPI Master. The code will

//! send three characters on the master Tx then polls the receive FIFO until

//! 3 characters are received on the master Rx.

//!

//! This example uses the following peripherals and I/O signals. You must

//! review these and change as needed for your own board:

//! - SSI0 peripheral

//! - GPIO Port A peripheral (for SSI0 pins)

//! - SSI0Clk - PA2

//! - SSI0Fss - PA3

//! - SSI0Rx - PA4

//! - SSI0Tx - PA5

//!

//! The following UART signals are configured only for displaying console

//! messages for this example. These are not required for operation of SSI0.

//! - UART0 peripheral

//! - GPIO Port A peripheral (for UART0 pins)

//! - UART0RX - PA0

//! - UART0TX - PA1

//!

//! This example uses the following interrupt handlers. To use this example

//! in your own application you must add these interrupt handlers to your

//! vector table.

//! - None.

//

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

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

//

// Number of bytes to send and receive.

//

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

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

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

//

// 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.

**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);

}

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

//

// Configure SSI0 in master Freescale (SPI) mode. This example will send out

// 3 bytes of data, then wait for 3 bytes of data to come in. This will all be

// done using the polling method.

//

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

**int**

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

{

// Holding the ADC values that come in FIFO

uint32\_t ui32ADC0Value[4];

uint32\_t tempTest;

// Variables for calculating temperature from the sensor

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Set up the system clock at 40 MHz

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

// Enable ADC0 Peripheral

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

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

// Configure ADC Sequencer

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

// Configuring all 4 steps of the ADC sequencer

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 0, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 1, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 2, ADC\_CTL\_TS);

// Sample Temperature sensor & configure interrupt flag

// Last conversion on sequencer 1

**ADCSequenceStepConfigure**(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

// Enable ADC sequencer 1

**ADCSequenceEnable**(ADC0\_BASE, 1);

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);

**while**(1){

// Clear the flag

**ADCIntClear**(ADC0\_BASE, 1);

// Trigger the ADC Conversion

**ADCProcessorTrigger**(ADC0\_BASE, 1);

// Wait for the conversion to Complete

**while**(!**ADCIntStatus**(ADC0\_BASE, 1, false))

{

}

// Read the value from the ADC Sample Sequencer 1 FIFO

**ADCSequenceDataGet**(ADC0\_BASE, 1, ui32ADC0Value);

// Calculate the Average temperature of sensor data

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

// TEMP in C = 147.5 - (( 75 \* (VREFP - VREFN) \* ADCVALUE) / 4096)

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

// Convert C to F => F = ( ( C \* 9 ) + 160 ) / 5

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

// 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.

tempTest = (ui32TempValueF/10) + 48;

pui32DataTx[0] = tempTest;

tempTest = (ui32TempValueF%10) + 48;

pui32DataTx[1] = tempTest;

// Display indication that the SSI is transmitting data.

**UARTprintf**("\nSent:\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.

**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]);

**SysCtlDelay**(1000000);

}

}

// Return no errors

**return**(0);

}

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

**Task 02:**

Youtube Link: <https://www.youtube.com/watch?v=Z_dliyFa3nM>

**#include** <stdbool.h>

**#include** <stdint.h>

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

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

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

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

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

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

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

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

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

**#include** "Nokia5110.h"

//! This example shows how to configure the SSI0 as SPI Master. The code will

//! send three characters on the master Tx then polls the receive FIFO until

//! 3 characters are received on the master Rx.

//!

//! This example uses the following peripherals and I/O signals. You must

//! review these and change as needed for your own board:

//! - SSI0 peripheral

//! - GPIO Port A peripheral (for SSI0 pins)

//! - SSI0Clk - PA2

//! - SSI0Fss - PA3

//! - SSI0Rx - PA4

//! - SSI0Tx - PA5

//!

//! The following UART signals are configured only for displaying console

//! messages for this example. These are not required for operation of SSI0.

//! - UART0 peripheral

//! - GPIO Port A peripheral (for UART0 pins)

//! - UART0RX - PA0

//! - UART0TX - PA1

//!

//! This example uses the following interrupt handlers. To use this example

//! in your own application you must add these interrupt handlers to your

//! vector table.

//! - None.

//

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

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

//

// Number of bytes to send and receive.

//

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

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

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

//

// 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.

**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);

}

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

//

// Configure SSI0 in master Freescale (SPI) mode. This example will send out

// 3 bytes of data, then wait for 3 bytes of data to come in. This will all be

// done using the polling method.

//

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

**int**

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

vvvbbggggggggggggggggggggggg{

// Holding the ADC values that come in FIFO

uint32\_t ui32ADC0Value[4];

uint32\_t tempTest;

// Variables for calculating temperature from the sensor

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Set up the system clock at 40 MHz

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

// Enable ADC0 Peripheral

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ADCHardwareOversampleConfigure(ADC0\_BASE, 64);

// Configure ADC Sequencer

ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

// Configuring all 4 steps of the ADC sequencer

ADCSequenceStepConfigure(ADC0\_BASE, 1, 0, ADC\_CTL\_TS);

ADCSequenceStepConfigure(ADC0\_BASE, 1, 1, ADC\_CTL\_TS);

ADCSequenceStepConfigure(ADC0\_BASE, 1, 2, ADC\_CTL\_TS);

// Sample Temperature sensor & configure interrupt flag

// Last conversion on sequencer 1

ADCSequenceStepConfigure(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

// Enable ADC sequencer 1

ADCSequenceEnable(ADC0\_BASE, 1);

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();

SysTick\_Init();

startSSI0();

initialize\_screen(BACKLIGHT\_OFF,SSI0);

**int** max=11,current\_pos=0;

set\_buttons\_up\_down();

screen\_write("Temperature: ",ALIGN\_CENTRE\_CENTRE,SSI0);

SysTick\_Wait50ms(100);

**unsigned** **char** menu\_elements[1][1];

**while**(1){

clear\_screen(SSI0);

// Clear the flag

ADCIntClear(ADC0\_BASE, 1);

// Trigger the ADC Conversion

ADCProcessorTrigger(ADC0\_BASE, 1);

// Wait for the conversion to Complete

**while**(!ADCIntStatus(ADC0\_BASE, 1, false))

{

}

// Read the value from the ADC Sample Sequencer 1 FIFO

ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value);

// Calculate the Average temperature of sensor data

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

// TEMP in C = 147.5 - (( 75 \* (VREFP - VREFN) \* ADCVALUE) / 4096)

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

// Convert C to F => F = ( ( C \* 9 ) + 160 ) / 5

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

// 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]))

{

}

// Display indication that the SSI is transmitting data.

tempTest = (ui32TempValueF/10) + 48;

menu\_elements[0][0]=tempTest;

tempTest = (ui32TempValueF%10) + 48;

menu\_elements[0][1]=tempTest;

set\_menu(menu\_elements);

clear\_screen(SSI0);

**char** data=GPIO\_PORTB\_DATA\_R&0x03;

**if**((data==0x01) && current\_pos<max)

{

current\_pos++;

}

**else**

{

**if**((data==0x02) && current\_pos>0)

{

current\_pos--;

}

}

show\_menu(current\_pos,SSI0);

SysTick\_Wait50ms(20);

}

// Return no errors

**return**(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

}

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