**Task 1:**

Modify the supplied code to transmit and receive the Internal Temperature and verify the

results.

**#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

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

**#define** TARGET\_IS\_BLIZZARD\_RB1

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

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

//

//! \addtogroup ssi\_examples\_list

//! <h1>SPI Master (spi\_master)</h1>

//!

//! 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 3

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

//

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

// as the example is running.

//

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

**void**

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

{

// ENABLE ADC FUNCTIONALITY

// enable periphery for GPIO\_F

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

// GPIO output enable for LEDs

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

// enable ADC0, set oversampling to 64

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64);

// configure ADC sequencer to 2 and assign the ADC value types

ROM\_ADCSequenceConfigure(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

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

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

ROM\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

ROM\_ADCSequenceEnable(ADC0\_BASE, 2);

// -----

//

// 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);

}

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

//

// 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**)

{

**#if** defined(TARGET\_IS\_TM4C129\_RA0) || \

defined(TARGET\_IS\_TM4C129\_RA1) || \

defined(TARGET\_IS\_TM4C129\_RA2)

uint32\_t ui32SysClock;

**#endif**

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

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

uint32\_t ui32Index;

//

// Set the clocking to run directly from the external crystal/oscillator.

// **TODO**: The SYSCTL\_XTAL\_ value must be changed to match the value of the

// crystal on your board.

//

**#if** defined(TARGET\_IS\_TM4C129\_RA0) || \

defined(TARGET\_IS\_TM4C129\_RA1) || \

defined(TARGET\_IS\_TM4C129\_RA2)

ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |

SYSCTL\_OSC\_MAIN |

SYSCTL\_USE\_OSC), 25000000);

**#else**

**SysCtlClockSet**(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN |

SYSCTL\_XTAL\_16MHZ);

**#endif**

//

// 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.

//

**#if** defined(TARGET\_IS\_TM4C129\_RA0) || \

defined(TARGET\_IS\_TM4C129\_RA1) || \

defined(TARGET\_IS\_TM4C129\_RA2)

SSIConfigSetExpClk(SSI0\_BASE, ui32SysClock, SSI\_FRF\_MOTO\_MODE\_0,

SSI\_MODE\_MASTER, 1000000, 8);

**#else**

**SSIConfigSetExpClk**(SSI0\_BASE, **SysCtlClockGet**(), SSI\_FRF\_MOTO\_MODE\_0,

SSI\_MODE\_MASTER, 1000000, 8);

**#endif**

//

// Enable the SSI0 module.

//

**SSIEnable**(SSI0\_BASE);

**while**(1)

{

// clear ADC interrupt and set processor trigger to sequence 2

ROM\_ADCIntClear(ADC0\_BASE, 2);

ROM\_ADCProcessorTrigger(ADC0\_BASE, 2);

// while busy, keep looping

**while**(!ROM\_ADCIntStatus(ADC0\_BASE, 2, false))

{}

// grab ADC data and compute the avg value and temps for F and C

ROM\_ADCSequenceDataGet(ADC0\_BASE, 2, ui32ADC0Value);

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

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

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.

//

// ADC value, int to ascii goes here

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.

//

**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 2:**

Display the z-axis results in Nokia5110 GLCD. If task is not working, display the Lab 5 –

Temperature on the LCD as: “Temperature: 72.92 F, 20.34 F”. Update every sec. using the timer.