**Date Submitted: 10/19/2019**

**Task 00: Execute provided code**

**Youtube Link:** <https://www.youtube.com/watch?v=8owXudf8IcQ>

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

**Task 01:**

Youtube Link: <https://www.youtube.com/watch?v=wHr1VI5x98k>

**Modified Schematic (if applicable):**

**Modified 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** "driverlib/adc.h"

**#include** "utils/uartstdio.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 the temperature values on the master Tx then polls the receive FIFO until

//! the temperature values 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.

//

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

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

//

// 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); // UART 0, baud rate 115200, clock 16MHz

}

// Function to initialize ADC module for

// Internal Temperature sensor

**void** **InitADC** (**void**) {

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //enable ADC0 peripheral

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0); // using ADC sample sequencer 1 (SS1), set as the highest priority, and processor will trigger ADC

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

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

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

**ADCSequenceStepConfigure**(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END); //ADC sample step 3, set ADC interrupt flag, end sampling

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

}

//reverse() function in association with itoa function

//reverses string

**void** **reverse**(**char** str[], **int** len)

{

**int** start, end;

**char** temp;

**for**(start=0, end=len-1; start < end; start++, end--) {

temp = \*(str+start);

\*(str+start) = \*(str+end);

\*(str+end) = temp;

}

}

// Function to convert integer to string

**char**\* **itoa**(**int** num, **char**\* str, **int** base)

{

**int** i = 0;

bool isNegative = false;

/\* A zero is same "0" string in all base \*/

**if** (num == 0) {

str[i] = '0';

str[i + 1] = '\0';

**return** str;

}

/\* negative numbers are only handled if base is 10

otherwise considered unsigned number \*/

**if** (num < 0 && base == 10) {

isNegative = true;

num = -num;

}

**while** (num != 0) {

**int** rem = num % base;

str[i++] = (rem > 9)? (rem-10) + 'A' : rem + '0';

num = num/base;

}

/\* Append negative sign for negative numbers \*/

**if** (isNegative){

str[i++] = '-';

}

str[i] = '\0';

reverse(str, i);

**return** str;

}

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

//

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

// the temp value, then wait for the temp value to come in. This will all be

// done using the polling method.

//

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

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

{

uint32\_t pui32DataTx;

uint32\_t pui32DataRx;

// global variables for ADC

uint32\_t ui32ADC0Value[4]; // array to store samples of ADC with 4 steps

**volatile** uint32\_t ui32TempAvg; // stores avg temp

**volatile** uint32\_t ui32TempValueC; // stores temp in Celsius

**volatile** uint32\_t ui32TempValueF; // stores temp in Fahrenheit

**char** tempstring[10]; // variable used to store string to display temp using UART

//

// 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.

//

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

SYSCTL\_XTAL\_16MHZ); // clock is 50MHz

//

// 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(); // initialize UART for displaying onto console

InitADC(); // initialize ADC (call function)

//

// Display the setup on the console. (prompt)

//

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);

**GPIOPinWrite**(GPIO\_PORTA\_BASE,GPIO\_PIN\_4,GPIO\_PIN\_4); // Write Rx value

//

// 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.

//

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

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

SSI\_MODE\_MASTER, 1000000, 8);

//

// Enable the SSI0 module.

//

**SSIEnable**(SSI0\_BASE);

**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.

//

**ADCIntClear**(ADC0\_BASE, 1); // clear ADC interrupt

**ADCProcessorTrigger**(ADC0\_BASE, 1); // processor begins to trigger ADC

**while**(!**ADCIntStatus**(ADC0\_BASE, 1, false)) // wait for ADC conversion..

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 1, ui32ADC0Value); // get ADC value from samples

//calculations of temperatures

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

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

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

pui32DataTx = ui32TempValueF;

**while**(**SSIDataGetNonBlocking**(SSI0\_BASE, &pui32DataRx))

{

}

//

// Display indication that the SSI is transmitting data.

//

UARTprintf("Sent:\n ");

itoa(pui32DataTx, tempstring, 10);

UARTprintf("%s ", tempstring);

**SSIDataPut**(SSI0\_BASE, pui32DataTx);

//

// 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 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);

//

// Display the data that SSI0 received.

//

itoa(pui32DataRx, tempstring, 10);

UARTprintf("%s \n", tempstring);

}

}

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

**Task 02:**

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

**Modified Schematic (if applicable):**

**Modified Code:**

**#include** <stdint.h>

**#include** <stdbool.h>

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

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

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

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

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

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

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

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

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

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

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

**#define** NUM\_LEDS 8

uint8\_t frame\_buffer[NUM\_LEDS\*3];

**void** **send\_data**(uint8\_t\* data, uint8\_t num\_leds);

**void** **fill\_frame\_buffer**(uint8\_t r, uint8\_t g, uint8\_t b, uint32\_t num\_leds);

**static** **volatile** uint32\_t ssi\_lut[] = {

0b100100100,

0b110100100,

0b100110100,

0b110110100,

0b100100110,

0b110100110,

0b100110110,

0b110110110

};

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

**FPULazyStackingEnable**();

// 80MHz

**SysCtlClockSet**(SYSCTL\_SYSDIV\_2\_5 | SYSCTL\_USE\_PLL | SYSCTL\_XTAL\_16MHZ |

SYSCTL\_OSC\_MAIN);

//initialize SPI

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

**SysCtlDelay**(50000);

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

**SysCtlDelay**(50000);

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

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

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

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

**GPIOPinTypeSSI**(GPIO\_PORTA\_BASE, GPIO\_PIN\_5);

**GPIOPinTypeSSI**(GPIO\_PORTA\_BASE, GPIO\_PIN\_2);

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

**GPIOPinTypeSSI**(GPIO\_PORTA\_BASE, GPIO\_PIN\_3);

//20 MHz data rate

**SSIConfigSetExpClk**(SSI0\_BASE, 80000000, SSI\_FRF\_MOTO\_MODE\_0, SSI\_MODE\_MASTER, 2400000, 9);

**SSIEnable**(SSI0\_BASE);

**while**(1)

{

fill\_frame\_buffer(255, 0, 0, NUM\_LEDS); // RED

send\_data(frame\_buffer, NUM\_LEDS);

**SysCtlDelay**(**SysCtlClockGet**()/5); // Delay for some time

fill\_frame\_buffer(0, 255, 0, NUM\_LEDS); // GREEN

send\_data(frame\_buffer, NUM\_LEDS);

**SysCtlDelay**(**SysCtlClockGet**()/5);

fill\_frame\_buffer(0, 0, 255, NUM\_LEDS); // BLUE

send\_data(frame\_buffer, NUM\_LEDS);

**SysCtlDelay**(**SysCtlClockGet**()/5);

fill\_frame\_buffer(255, 255, 0, NUM\_LEDS); // YELLOW

send\_data(frame\_buffer, NUM\_LEDS);

**SysCtlDelay**(**SysCtlClockGet**()/5);

fill\_frame\_buffer(255, 0, 255, NUM\_LEDS); // PURPLE

send\_data(frame\_buffer, NUM\_LEDS);

**SysCtlDelay**(**SysCtlClockGet**()/5);

fill\_frame\_buffer(0, 255, 255, NUM\_LEDS); // LIGHT BLUE

send\_data(frame\_buffer, NUM\_LEDS);

**SysCtlDelay**(**SysCtlClockGet**()/5);

fill\_frame\_buffer(255, 255, 255, NUM\_LEDS); // WHITE

send\_data(frame\_buffer, NUM\_LEDS);

**SysCtlDelay**(**SysCtlClockGet**()/5);

}

}

**void** **send\_data**(uint8\_t\* data, uint8\_t num\_leds)

{

uint32\_t i, j, curr\_lut\_index, curr\_rgb;

**for**(i = 0; i < (num\_leds\*3); i = i + 3) {

curr\_rgb = (((uint32\_t)data[i + 2]) << 16) | (((uint32\_t)data[i + 1]) << 8) | data[i];

**for**(j = 0; j < 24; j = j + 3) {

curr\_lut\_index = ((curr\_rgb>>j) & 0b111);

**SSIDataPut**(SSI0\_BASE, ssi\_lut[curr\_lut\_index]);

}

}

**SysCtlDelay**(50000); // delay more then 50us

}

**void** **fill\_frame\_buffer**(uint8\_t r, uint8\_t g, uint8\_t b, uint32\_t num\_leds)

{

uint32\_t i;

uint8\_t\* frame\_buffer\_index = frame\_buffer;

**for**(i = 0; i < num\_leds; i++) {

\*(frame\_buffer\_index++) = g;

\*(frame\_buffer\_index++) = r;

\*(frame\_buffer\_index++) = b;

}

}

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