**Date Submitted: October 20, 2019**

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

**Task 01:**

Youtube Link: https://youtu.be/UgNqeK3ALyI

**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** "utils/uartstdio.h"

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

**#include** "driverlib/adc.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 1 //Only sending Temperature value one at a time

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

//

// 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);

}

uint32\_t **getADCtemp**(**void**);

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

//

// 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;

uint32\_t pui32ADCDataTx;

uint32\_t pui32ADCDataRx;

//

// 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**

//

// The ADC0 peripheral must be enabled for use.

//

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

**SysCtlDelay**(3);

//

// Enable sample sequence 3 with a processor signal trigger. Sequence 3

// will do a single sample when the processor sends a singal to start the

// conversion. Each ADC module has 4 programmable sequences, sequence 0

// to sequence 3. This example is arbitrarily using sequence 3.

//

**ADCSequenceConfigure**(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

//

// Configure step 0 on sequence 3. Sample the temperature sensor

// (ADC\_CTL\_TS) and configure the interrupt flag (ADC\_CTL\_IE) to be set

// when the sample is done. Tell the ADC logic that this is the last

// conversion on sequence 3 (ADC\_CTL\_END). Sequence 3 has only one

// programmable step. Sequence 1 and 2 have 4 steps, and sequence 0 has

// 8 programmable steps. Since we are only doing a single conversion using

// sequence 3 we will only configure step 0. For more information on the

// ADC sequences and steps, reference the datasheet.

//

**ADCSequenceStepConfigure**(ADC0\_BASE, 3, 0, ADC\_CTL\_TS | ADC\_CTL\_IE |

ADC\_CTL\_END);

//

// Since sample sequence 3 is now configured, it must be enabled.

//

**ADCSequenceEnable**(ADC0\_BASE, 3);

//

// Clear the interrupt status flag. This is done to make sure the

// interrupt flag is cleared before we sample.

//

**ADCIntClear**(ADC0\_BASE, 3);

//

// 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){

//

// 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';

pui32ADCDataTx = getADCtemp();

//

// Display indication that the SSI is transmitting data.

//

**UARTprintf**("\nTemperature Sent: ");

//

// Send 3 bytes of data.

//

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

{

//

// Display the data that SSI is transferring.

//

//UARTprintf("'%c' ", pui32DataTx[ui32Index]);

**UARTprintf**("%d ", pui32ADCDataTx);

//

// 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]);

**SSIDataPut**(SSI0\_BASE, pui32ADCDataTx);

}

**UARTprintf**("\n");

//

// 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**("\nTemperature Received: ");

//

// 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**("%d ", pui32DataRx[ui32Index]);

}

**UARTprintf**("\n");

}

//

// Return no errors

//

**return**(0);

}

uint32\_t **getADCtemp**(**void**) {

//

// This array is used for storing the data read from the ADC FIFO. It

// must be as large as the FIFO for the sequencer in use. This example

// uses sequence 3 which has a FIFO depth of 1. If another sequence

// was used with a deeper FIFO, then the array size must be changed.

//

uint32\_t ADCValues[1];

//

// These variables are used to store the temperature conversions for

// Celsius and Fahrenheit.

//

uint32\_t TempValueC ;

uint32\_t TempValueF ;

//

// Trigger the ADC conversion.

//

**ADCProcessorTrigger**(ADC0\_BASE, 3);

//

// Wait for conversion to be completed.

//

**while**(!**ADCIntStatus**(ADC0\_BASE, 3, false))

{

}

//

// Clear the ADC interrupt flag.

//

**ADCIntClear**(ADC0\_BASE, 3);

//

// Read ADC Value.

//

**ADCSequenceDataGet**(ADC0\_BASE, 3, ADCValues);

//

// Use non-calibrated conversion provided in the data sheet. I use floats in intermediate

// math but you could use intergers with multiplied by powers of 10 and divide on the end

// Make sure you divide last to avoid dropout.

//

TempValueC = (uint32\_t)(147.5 - ((75.0\*3.3 \*(**float**)ADCValues[0])) / 4096.0);

//

// Get Fahrenheit value. Make sure you divide last to avoid dropout.

//

TempValueF = ((TempValueC \* 9) + 160) / 5;

//

// This function provides a means of generating a constant length

// delay. The function delay (in cycles) = 3 \* parameter. Delay

// 250ms arbitrarily.

//

**SysCtlDelay**(80000000 / 12);

**return** TempValueF;

}

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

**Task 02:**

Youtube Link: https://youtu.be/6mzuEv5ZfLU

**Modified Schematic (if applicable):**

**Modified Code:**

/\*

\* main.c

\*/

**#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 1

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);

**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);

uint32\_t i = 0;

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

**while**(1)

{

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

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

**SysCtlDelay**(800000); // delay more then 50us

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

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

**SysCtlDelay**(800000); // delay more then 50us

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

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

**SysCtlDelay**(800000); // delay more then 50us

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

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

**SysCtlDelay**(800000); // delay more then 50us

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

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

**SysCtlDelay**(800000); // delay more then 50us

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

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

**SysCtlDelay**(800000); // delay more then 50us

fill\_frame\_buffer(40, 255, 255, NUM\_LEDS); //RGB

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

**SysCtlDelay**(800000); // delay more then 50us

}

**return** 0;

}

**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;

}

}

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