**Date Submitted: 12/3/19**

**Task 01:**

Youtube Link: https://youtu.be/cGsUAsH9ymk

**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"**

**//added**

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

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

**#define NUM\_SSI\_DATA 1**

**//added from lab 05**

**uint32\_t ui32ADC0Value[4];**

**volatile uint32\_t ui32TempAvg;**

**volatile uint32\_t ui32TempValueC;**

**volatile uint32\_t ui32TempValueF;**

**void configADC(void);**

**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 UARTfor 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)**

**{**

**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.**

**// The SSI0 peripheral is on Port A and pins 2,3,4 and 5.**

**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.**

**// This function/s configures the pin muxing on port A pins 2,3,4 and 5**

**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**

**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);**

**//Set system clock**

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

**configADC();**

**while(1)**

**{**

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

**ADCProcessorTrigger(ADC0\_BASE, 1); // trigger ADC conversion with software**

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

**{**

**// wait for conversion**

**}**

**ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value); //get data from a buffer in memory**

**// temperature calculations**

**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]))**

**{**

**}**

**//pui32DataTx[0] = 's';**

**pui32DataTx[0] = ui32TempValueF; // Initialize the data to send.**

**SysCtlDelay( (SysCtlClockGet()/3));**

**//SysCtlDelay( (SysCtlClockGet()/(3\*1000))\*1000 ) ; //**

**// 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("'%u' ", pui32DataTx[ui32Index]); // %u is unsigned**

**// 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))**

**{**

**}**

**SysCtlDelay( (SysCtlClockGet()/3));**

**// SysCtlDelay( (SysCtlClockGet()/(3\*1000))\*1000 ) ; //**

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

**}**

**}**

**// Return no errors**

**return(0);**

**}**

**void configADC(void)**

**{**

**//Enable ADC**

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

**//Configure ADC sequencer**

**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);**

**//Enable ADC sequencer 1**

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

**}**

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

**Task 02:**

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

**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"**

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

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

**#define RED 255**

**#define GREEN 255**

**#define BLUE 255**

**#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);**

**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)**

**{**

**// Red**

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

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

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

**// Green**

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

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

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

**// Blue**

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

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

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

**// Red + Green**

**fill\_frame\_buffer(RED, GREEN, 0, NUM\_LEDS);**

**SysCtlDelay((SysCtlClockGet()/4)); // delay**

**// Red + Blue**

**fill\_frame\_buffer(RED,0,BLUE, NUM\_LEDS);**

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

**SysCtlDelay((SysCtlClockGet()/4)); //delay**

**// Green Blue**

**fill\_frame\_buffer(0,GREEN,BLUE, NUM\_LEDS);**

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

**SysCtlDelay((SysCtlClockGet()/4)); //delay**

**// Red Green Blue**

**fill\_frame\_buffer(RED, GREEN, BLUE, NUM\_LEDS);**

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

**SysCtlDelay((SysCtlClockGet()/4));//delay**

**}**

**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); // 50us delay**

**}**

**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;**

**}**

**}**