Date Submitted:

Task 00: Execute provided code

Youtube Link:

https://youtu.be/8MtxIKTdNF0

Task 01:

Youtube Link:

https://youtu.be/xmoBGKFOkfo

Modified Code:

```
// Insert code here

#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ssi.h"
#include"inc/hw_ints.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"
#define TARGET_IS_BLIZZARD_RB1
#include "driverlib/rom.h"
#include "driverlib/debug.h"

#ifdef DEBUG
void__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif


//! 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
```

```c
//! - 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 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);
}

//*****************************************************************************
//
// This function initializes the ADC
//
//
//*****************************************************************************
```

```
void InitADC(void)
{


ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 64);
    ROM_ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);

    ROM_ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    ROM_ADCSequenceEnable(ADC0_BASE, 1);



}

//*****************************************************************************
//
// This function converts int to string
//
//
//*****************************************************************************

char* itoa( uint32_t num, char* str, int base)
{
    int i = 0;

    /* Handle 0 explicitely, otherwise empty string is printed for 0 */
    if (num == 0)
    {
        str[i++] = '0';
        str[i] = '\0';
        return str;
    }

    // Process individual digits
    while (num != 0)
    {
        int rem = num % base;
        str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';
        num = num/base;
    }

    str[i] = '\0'; // Append string terminator

    return str;
}
```

```
//*****************************************************************************
//
// 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 ui32ADC0Value[4];
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;
    uint32_t pui32DataTx[NUM_SSI_DATA];
    uint32_t pui32DataRx[NUM_SSI_DATA];
    uint32_t ui32Index;

    char str[10]; //buffer

    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);

    InitADC(); //intialize adc

    while(1){

        ROM_ADCIntClear(ADC0_BASE, 1);
        ROM_ADCProcessorTrigger(ADC0_BASE, 1);

        while(!ROM_ADCIntStatus(ADC0_BASE, 1, false))
        {
        }

        ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
        ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;


        itoa(ui32TempValueF, str, 10); //call itoa

        // 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] = str[2];

        pui32DataTx[1] = str[1];
        pui32DataTx[2] = str[0];
        // 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]);
    }
    UARTprintf("\n");
  }
}
```

Task 02:

Youtube Link:

https://youtu.be/a0zuTudzbEE

Modified Code:

```
// Insert code here
/*
 * 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 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);

    //fill_frame_buffer(48, 255, 255, NUM_LEDS);
    while(1)
    {
        fill_frame_buffer(255, 0, 0, NUM_LEDS); //red
        send_data(frame_buffer, NUM_LEDS);

        SysCtlDelay(1000000);

        fill_frame_buffer(0, 255, 0, NUM_LEDS); //green
        send_data(frame_buffer, NUM_LEDS);

        SysCtlDelay(1000000);

        fill_frame_buffer(0, 0, 255, NUM_LEDS); //blue
        send_data(frame_buffer, NUM_LEDS);

        SysCtlDelay(1000000);

        fill_frame_buffer(255, 255, 0, NUM_LEDS); //yellow
        send_data(frame_buffer, NUM_LEDS);

        SysCtlDelay(1000000);

        fill_frame_buffer(255, 0, 255, NUM_LEDS); //pink (or purple)
        send_data(frame_buffer, NUM_LEDS);

        SysCtlDelay(1000000);

        fill_frame_buffer(255, 0, 255, NUM_LEDS); //cyan
        send_data(frame_buffer, NUM_LEDS);

        SysCtlDelay(1000000);

        fill_frame_buffer(255, 255, 255, NUM_LEDS); //white
        send_data(frame_buffer, NUM_LEDS);

        SysCtlDelay(1000000);
    }

    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;
    }
}
---------------------------------------------------------------------------------
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.