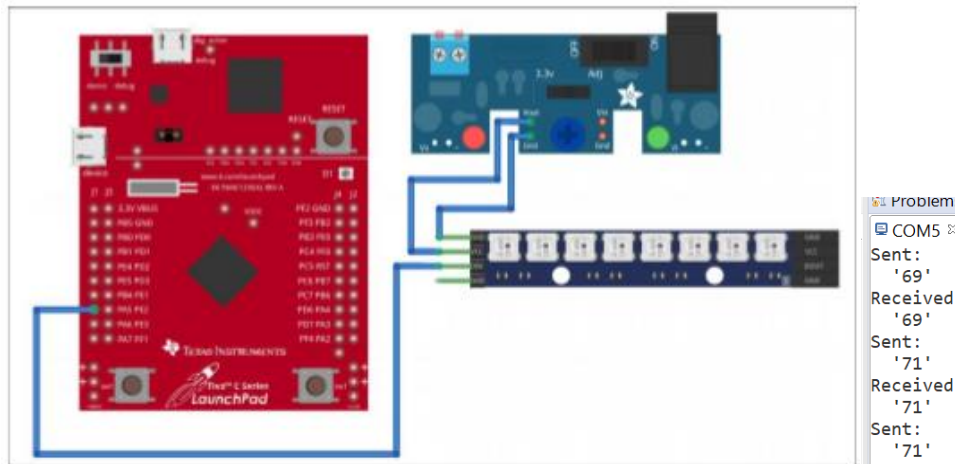


Date Submitted: 11/4/19**Task 01:**

Youtube Link:

<https://www.youtube.com/watch?v=eq2gsozjW7g>

Modified Schematic (if applicable):



Modified Code:

```
// Insert code here
#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 "driverlib/adc.h"
#include "driverlib/debug.h"

#define NUM_SSI_DATA          1

/*****
//
// 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);
```

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

```

// 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.
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);
}

//*****
//
// 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();

    // The SSI0 peripheral must be enabled for use.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
    // The SSI0 peripheral is on Port A and pins 2,3,4 and 5.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // 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(SSIO_BASE, SysCtlClockGet(),
SSI_FRF_MOTO_MODE_0,SSI_MODE_MASTER, 1000000, 8);
// Enable the SSI0 module.
SSIEnable(SSIO_BASE);

//Variables for Temperature
uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

//Set system clock
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

//Enable ADC
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

//Configure all four steps of 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);

//Configure interrupt flag = ADC_CTL_IE
//Tell ADC logic that this is the last conversion on sequencer
ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

//Enable ADC sequencer 1
ADCSequenceEnable(ADC0_BASE, 1);

while(1)
{
    //clear interrupt flag
    ADCIntClear(ADC0_BASE, 1);
    //trigger ADC conversion with software
    ADCProcessorTrigger(ADC0_BASE, 1);

    //wait for conversion
    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
    }
}

```

```

//get data from a buffer in memory
ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

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

// The "non-blocking" function checks if there is any data in the receive
// FIFO and does not "hang" if there isn't.
while(SSIDataGetNonBlocking(SSIO_BASE, &pui32DataRx[0]))
{
}

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

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]);

    // 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(SSIO_BASE, pui32DataTx[ui32Index]);
}

// Wait until SSI0 is done transferring all the data in the transmit FIFO.
while(SSIBusy(SSIO_BASE))
{
}
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(SSIO_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);
}

```

Task 02:

Youtube Link:

<https://www.youtube.com/watch?v=0DDUr4plDsM>

Modified Schematic (if applicable):

Modified Code:

```

// Ricky Perez
// CpE 403
// Lab 8
// Task 2
// Interface the WS2818B 1x8 RGB LED strip with TivaC using SPI interface.
// Implement a running R, G, B, RG, RB, GB, and RGB light sequence.
#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 "driverlib/adc.h"
#include "driverlib/debug.h"

#define MAX_RED          255
#define MAX_GREEN        255
#define MAX_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);
void config_SET_LEDS(uint8_t red, uint8_t green, uint8_t blue, uint8_t led_numbs,
uint8_t *buf, int div_num);
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)
    {
        // R, G, B, RG, RB, GB, and RGB light sequence.
        // config_SET_LEDS(MAX_RED, 0, 0, NUM_LEDS, frame_buffer, 5);
        // Red
        config_SET_LEDS(MAX_RED, 0, 0, NUM_LEDS, frame_buffer, 5);

        // Green
        config_SET_LEDS(0, MAX_GREEN, 0, NUM_LEDS, frame_buffer, 5);

        // Blue
        config_SET_LEDS(0, 0, MAX_BLUE, NUM_LEDS, frame_buffer, 5);

        // Red and Green
        config_SET_LEDS(MAX_RED, MAX_GREEN, 0, NUM_LEDS, frame_buffer, 5);

        // Red and Blue
        config_SET_LEDS(MAX_RED, 0, MAX_BLUE, NUM_LEDS, frame_buffer, 5);

        // Green Blue
        config_SET_LEDS(0, MAX_GREEN, MAX_BLUE, NUM_LEDS, frame_buffer, 4);

        // Red Green Blue
        config_SET_LEDS( MAX_RED, MAX_GREEN, MAX_BLUE, NUM_LEDS, frame_buffer, 4);

    }
    return 0;
}

```

```

}

void config_SET_LEDs(uint8_t red, uint8_t green, uint8_t blue, uint8_t led_numbs,
uint8_t *buf, int div_num){
    fill_frame_buffer(red, green, blue, led_numbs );
    send_data(buf, led_numbs);
    SysCtlDelay( SysCtlClockGet()/div_num); // small delay
}

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);
            SSIDDataPut(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;
    }
}

```
