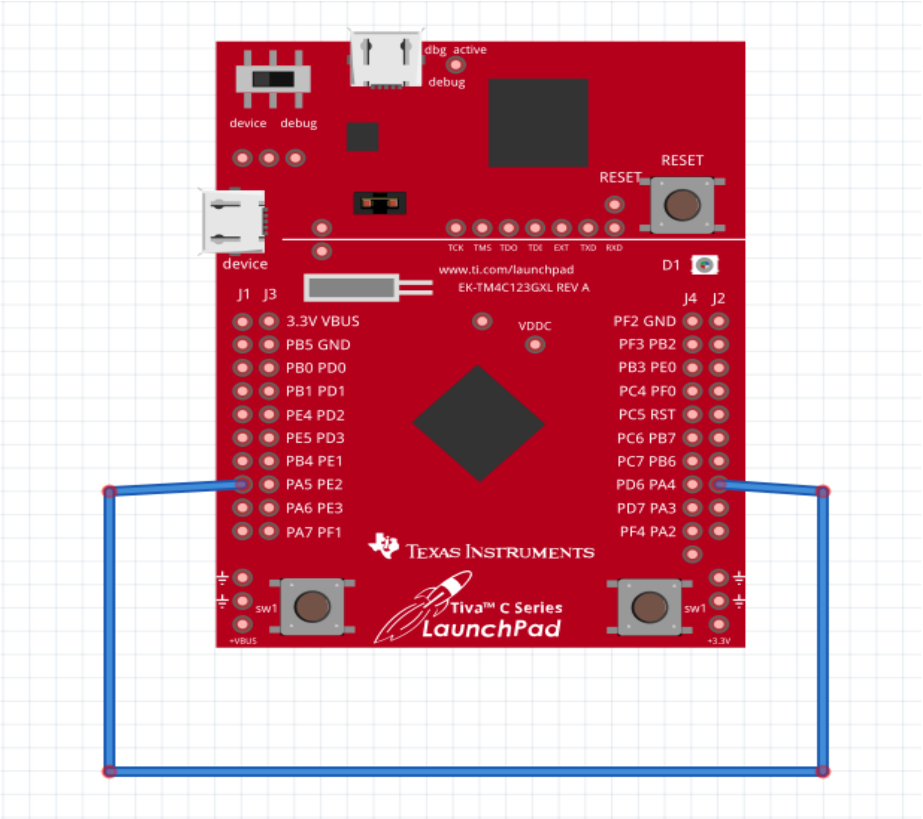
Nathan Hanuscin
Github root directory: https://github.com/nhanuscin/HappyFunStuff

Date Submitted: 10/19/19

Task 01:

Youtube Link: https://www.youtube.com/watch?v=tr27sTP2TSI

Modified Schematic (if applicable):



Modified Code:

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_ssi.h"
#include "inc/hw_types.h"
#include "driverlib/ssi.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include "driverlib/adc.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/debug.h"
```

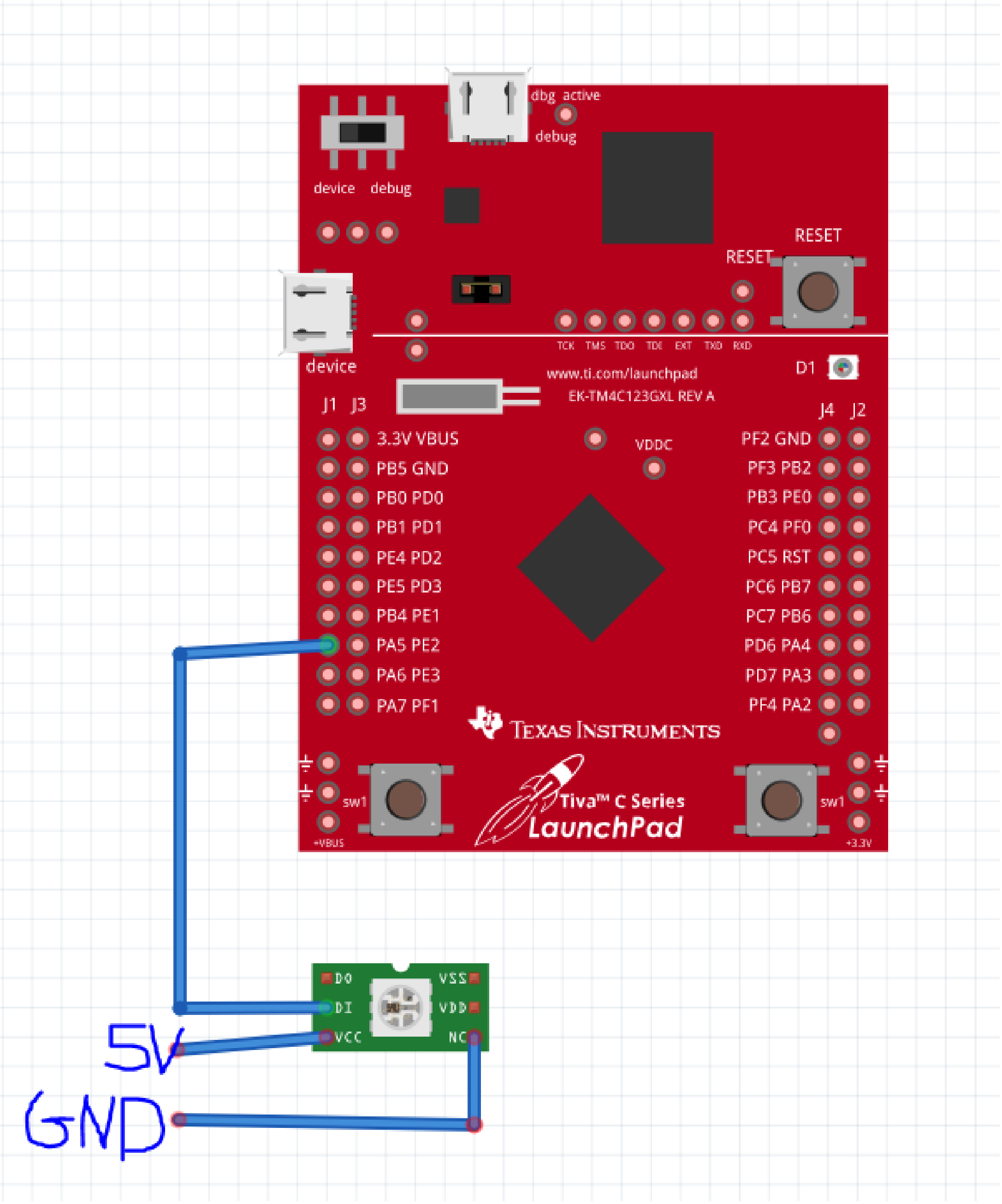
```
#define NUM_SSI_DATA     2

void initADC(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);   // enable ADC0
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);
    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);

}

int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
//40mhz clock
    initADC();
    //I had the following UART initialization lines in the init Console function
    //but my program kept going to the FaultISR so I moved them to main and then it
began working
    // 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.
    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);

    // The SSI0 peripheral must be enabled for use.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
    // 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.
    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);
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4, GPIO_PIN_4);
    // 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.
    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);
    // Display the setup on the console.
    UARTprintf("SSI ->\r\n");
    UARTprintf("  Mode: SPI\r\n");
    UARTprintf("  Data: 8-bit\r\n\r\n");

    //SSI variables
    //uint32_t pui32DataTx[NUM_SSI_DATA];
    uint32_t pui32DataRx[NUM_SSI_DATA];
    uint32_t ui32Index;
    //ADC variables
    uint32_t ui32ADC0Value[4];
    uint32_t ui32TempAvg;
    uint32_t ui32TempValueC;
    uint32_t ui32TempValueF;
    char temperatureTx[10];

    while(1)
    {
        while(SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[0]))
        {
        }
        // turn on ADC
        ADCIntClear(ADC0_BASE, 1);
        ADCSequenceEnable(ADC0_BASE, 1);
        ADCProcessorTrigger(ADC0_BASE, 1);
        while(!ADCIntStatus(ADC0_BASE, 1, false))
        {
          //wait until ADC finishes
        }
        //Get ADC values from SS1
        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
        //Average and Calculate Temperature
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
        ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

        //Convert Temperature to Character string
        ltoa(ui32TempValueF, temperatureTx);
        ADCSequenceDisable(ADC0_BASE, 0);

        // Display indication that the SSI is transmitting data.
        UARTprintf("\r\nSent:\r\n  '");
        // Send 2 bytes of data.
        for(ui32Index = 0; ui32Index < NUM_SSI_DATA; ui32Index++)
        {
            // Display the data that SSI is transferring.
            UARTprintf("%c", temperatureTx[ui32Index]);
            // Send the data using the "blocking" put function.  This function
            // will wait until there is room in the send FIFO before returning.
            SSIDataPut(SSI0_BASE, temperatureTx[ui32Index]);
        }
        UARTprintf("'F");
        // Wait until SSI0 is done transferring all the data in the transmit FIFO
        while(SSIBusy(SSI0_BASE))
        {
        }
        UARTprintf("\r\nReceived:\r\n  '");

        // Receive 2 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("'F");
        SysCtlDelay(5000000);
    }

}
------------------------------------------------------------------------------------
```

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

Task 02:

Youtube Link: https://www.youtube.com/watch?v=f2ms46w2_4c

Modified Schematic (if applicable):



Used single LED since fritzing doesn't have the WS2818 model.

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

Modified Code:

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_ssi.h"
#include "inc/hw_types.h"
#include "driverlib/ssi.h"
#include "driverlib/gpio.h"
#include "driverlib/fpu.h"
#include "driverlib/rom.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include "driverlib/debug.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);

    while(1)
    {
        fill_frame_buffer(255, 0, 0, NUM_LEDS);
        send_data(frame_buffer, NUM_LEDS);
        fill_frame_buffer(0, 255, 0, NUM_LEDS);
        send_data(frame_buffer, NUM_LEDS);
        fill_frame_buffer(0, 0, 255, NUM_LEDS);
        send_data(frame_buffer, NUM_LEDS);
        fill_frame_buffer(255, 255, 0, NUM_LEDS);
        send_data(frame_buffer, NUM_LEDS);
        fill_frame_buffer(255, 0, 255, NUM_LEDS);
        send_data(frame_buffer, NUM_LEDS);
        fill_frame_buffer(0, 255, 255, NUM_LEDS);
        send_data(frame_buffer, NUM_LEDS);
        fill_frame_buffer(255, 255, 255, NUM_LEDS);
        send_data(frame_buffer, NUM_LEDS);
    }

    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(5000000); //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;
    }
}
----------------------------------------------------------------------------------
```

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