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

Date Submitted: 10/13/19

Task 01:

Youtube Link: https://www.youtube.com/watch?v=e7QtbtYAJAg

Modified Schematic (if applicable): N/A

Modified Code:
```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"

uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

void UARTIntHandler(void)
{
    uint32_t ui32Status;
    ui32Status = UARTIntStatus(UART0_BASE, true);      //get interrupt status
    UARTIntClear(UART0_BASE, ui32Status);   //clear the assert3ed interrupts

    while(UARTCharsAvail(UART0_BASE))    //loop while there are chars
    {
        UARTCharPutNonBlocking(UART0_BASE, UARTCharGetNonBlocking(UART0_BASE));
//echo char
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);    //blink LED
        SysCtlDelay(SysCtlClockGet() / (1000*3)); //delay 1ms
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);   //turn off LED
    }
}

void Timer1IntHandler(void)
{

    char buff[20];
    // Clear the timer interrupt
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    //Clear interrupt flag
    ADCIntClear(ADC0_BASE, 1);
    ADCSequenceEnable(ADC0_BASE, 1);
    //Trigger ADC conversion with software
    ADCProcessorTrigger(ADC0_BASE, 1);

    //wait for ADC conversion to finish
    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
    }
    //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;

    ltoa(ui32TempValueF, buff); //convert temperature
    uint32_t i = 0;
    //transmit temperature until buffer is null
    while(buff[i] != '\0')
    {
        UARTCharPut(UART0_BASE, buff[i]);
        i++;
    }
    UARTCharPut(UART0_BASE, ' ');
    //UARTCharPut(UART0_BASE, '\n');
    ADCSequenceDisable(ADC0_BASE, 1);

}
void UARTsetup(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);    //enable GPIO portf
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2); //enable pin for LED PF2
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    IntMasterEnable();  //enable processor interrupts
    IntEnable(INT_UART0);   //enable the UART interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //enable both RX and TX
interrupts

}
```

```
int main(void) {

    uint32_t ui32Period;
         UARTsetup();

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);

    //Enable Timer1A
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);

    //Set timer to 2Hz
    ui32Period =  SysCtlClockGet()/ 2;
    TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period -1);
    //Enable timer interrupt
    IntEnable(INT_TIMER1A);

    //Configure ADC sequencer 1
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    //Sample Internal temp sensor with sequencer 1
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    //Sample temp sensor, set interrupt flag to end conversion, enable ADC
    ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    TimerEnable(TIMER1_BASE, TIMER_A);

    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'm');
    UARTCharPut(UART0_BASE, 'p');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'a');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'u');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');

    while (1)       //let interrupt handler to the UART echo function
    {
    }

}
```

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

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

Task 02:

Youtube Link: https://www.youtube.com/watch?v=2op487KWD9Q

Modified Schematic (if applicable): N/A

Modified Code:
```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"

uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

//function to print strings
void printString(char text[])
{
    int i = 0;
    while(text[i] != '\0')
    {
        UARTCharPut(UART0_BASE, text[i]);
        i++;
    }
}

void UARTIntHandler(void)
{
    uint32_t ui32Status;
    ui32Status = UARTIntStatus(UART0_BASE, true);      //get interrupt status
    UARTIntClear(UART0_BASE, ui32Status);   //clear the assert3ed interrupts

    char buffF[10];
    char buffC[10];
    char cmd;
    cmd = UARTCharGet(UART0_BASE);      //get user command

        //Turn on red LED
        if(cmd == 'R')
        {
            UARTCharPut(UART0_BASE, cmd);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
        }
        //turn off red LED
        else if(cmd == 'r')
        {
            UARTCharPut(UART0_BASE, cmd);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
        }
        //Turn on Green LED
        else if(cmd == 'G')
        {
            UARTCharPut(UART0_BASE, cmd);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
        }
        //Turn off Green LED
        else if(cmd == 'g')
        {
            UARTCharPut(UART0_BASE, cmd);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
        //Turn on Blue LED
        else if(cmd == 'B')
        {
            UARTCharPut(UART0_BASE, cmd);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);
        }

        //Turn off Blue LED
        else if(cmd == 'b')
        {
            UARTCharPut(UART0_BASE, cmd);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
        }

        else if(cmd == 'T')
        {
            UARTCharPut(UART0_BASE, cmd);
            //Clear interrupt flag
            ADCIntClear(ADC0_BASE, 1);
            ADCSequenceEnable(ADC0_BASE, 1);
            //Trigger ADC conversion with software
            ADCProcessorTrigger(ADC0_BASE, 1);
            //wait for ADC conversion to finish
            while(!ADCIntStatus(ADC0_BASE, 1, false))
            {
            }
            //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;
            ltoa(ui32TempValueF, buffF); //convert Fahrenheit temperature
            ltoa(ui32TempValueC, buffC); //convert Celsius temperature
            printString("\r\nTemp = ");
            printString(buffF);
            printString("F, ");
            printString(buffC);
            printString("C");
            ADCSequenceDisable(ADC0_BASE, 1);
        }

        else
        {
            UARTCharPut(UART0_BASE, cmd);
            UARTCharPut(UART0_BASE, '\r');
            UARTCharPut(UART0_BASE, '\n');
            printString("Usage: R - red, G - green, B - blue, T - temperature\r\n");
        }

    UARTCharPut(UART0_BASE, '\r');
    UARTCharPut(UART0_BASE, '\n');
    printString("Please enter command: ");


}

void UARTsetup(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);    //enable GPIO portf
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
//enable pin for LED PF2
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    IntMasterEnable();  //enable processor interrupts
    IntEnable(INT_UART0);   //enable the UART interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //enable both RX and TX
interrupts

}

int main(void) {

         UARTsetup();
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);
    //Configure ADC sequencer 1
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    //Sample Internal temp sensor with sequencer 1
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    //Sample temp sensor, set interrupt flag to end conversion, enable ADC
    ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);

    UARTCharPut(UART0_BASE, 'L');
    UARTCharPut(UART0_BASE, 'a');
    UARTCharPut(UART0_BASE, 'b');
    UARTCharPut(UART0_BASE, '0');
    UARTCharPut(UART0_BASE, '7');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'a');
    UARTCharPut(UART0_BASE, 's');
    UARTCharPut(UART0_BASE, 'k');
    UARTCharPut(UART0_BASE, '2');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');

    printString("Please Enter Command: ");

    while (1)       //let interrupt handler to the UART echo function
    {
    }

}


---------------------------------------------------------------------------------
```