Steven Lee
Github root directory: https://github.com/lees106/STAT411

Date Submitted: 10/14/2019

Task 00:

Youtube Link:
https://youtu.be/nRy4xZOhTc8

Task 01:

Youtube Link:
https://youtu.be/qrwD_69or9k

Modified Code:

```
#include <stdint.h>
#include <stdbool.h>
#include "\ti\tivaware_c_series_2_1_4_178\inc\hw_memmap.h"
#include "\ti\tivaware_c_series_2_1_4_178\inc\hw_types.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\sysctl.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\gpio.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\pin_map.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\uart.h"

#include "\ti\tivaware_c_series_2_1_4_178\inc\tm4c123gh6pm.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\timer.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\adc.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\debug.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\interrupt.h"

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

void configureTimer1A();
void UART_OutUDec(uint32_t);
void UART_OutChar(char data);

uint32_t ui32ADC0Value[1];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

int main(void) {
	SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
	SYSCTL_XTAL_16MHZ);

	SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //temp
```

```
	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

	SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); //temp
	ADCHardwareOversampleConfigure(ADC0_BASE, 32); //temp
	ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0); //temp
	ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);

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

	UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
		(UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

	configureTimer1A(); //temp

	ADCSequenceEnable(ADC0_BASE, 3); //temp
	ADCIntEnable(ADC0_BASE, 3); //temp
	while (1)
	{
		if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE,
		UARTCharGet(UART0_BASE));
	}
}

void configureTimer1A()
{
	int32_t ui32PeriodHigh = (SysCtlClockGet() / 1);

	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1); // Must call before calling
	peripheral specific driverlib function, or else Fault ISR
	IntMasterEnable();
	TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC); // Configures Timer 1 as a 32-bit
	timer in periodic mode (combines Timer 0A and 0B)

	TimerLoadSet(TIMER1_BASE, TIMER_A, 5 * (SysCtlClockGet() / 10)); // Since the
	interrupt fires at zero, you must subtract 1.

	IntEnable(INT_TIMER1A);
	TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
	TimerEnable(TIMER1_BASE, TIMER_A);
}

void Timer1IntHandler(void)
{
	int32_t ui32PeriodHigh = 0.5 * (SysCtlClockGet());

	TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

	TimerLoadSet(TIMER1_BASE, TIMER_A, ui32PeriodHigh);

	ADCIntClear(ADC0_BASE, 3); //clear adc conversion done flag before writing code
	that depends on it. change to sequence 2
	//Changed all sequence numbers below to sequence two
	ADCProcessorTrigger(ADC0_BASE, 3);
```

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

```
        while (!ADCIntStatus(ADC0_BASE, 3, false)) //wait for conversion to finish
        {
        } //if loop exited conversion is complete

        ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value); //gets samples from the array
        ui32TempValueC = (1475 - ((2475 * ui32ADC0Value[0])) / 4096) / 10;
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

        UART_OutUDec(ui32TempValueF);
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');
}

void UART_OutUDec(uint32_t n) { //
        if (n >= 10) {
                UART_OutUDec(n / 10);
                n = n % 10;
        }
        UART_OutChar(n + '0');
}

void UART_OutChar(char data) {
        while ((UART0_FR_R&UART_FR_TXFF) != 0);
        UART0_DR_R = data;
}
```

Task 02:

Youtube Link:
https://youtu.be/kUm68lKMCiE

Modified Code:

```
#include <stdint.h>
#include <stdbool.h>
#include "\ti\tivaware_c_series_2_1_4_178\inc\hw_memmap.h"
#include "\ti\tivaware_c_series_2_1_4_178\inc\hw_types.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\sysctl.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\gpio.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\pin_map.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\uart.h"

#include "\ti\tivaware_c_series_2_1_4_178\inc\tm4c123gh6pm.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\adc.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\debug.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\interrupt.h"

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

void UART_OutUDec(uint32_t);
```

```
void UART_OutChar(char data);
void UARTIntHandler(void);

uint32_t ui32ADC0Value[1];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

int main(void) {
	SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
	SYSCTL_XTAL_16MHZ);

	SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //temp
	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

	SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); //temp
	ADCHardwareOversampleConfigure(ADC0_BASE, 32); //temp
	ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0); //temp
	ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);

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

	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);


	ADCSequenceEnable(ADC0_BASE, 3); //temp
	ADCIntEnable(ADC0_BASE, 3); //temp
	while (1)
	{
	}
}

void UART_OutUDec(uint32_t n) {
	if (n >= 10) {
		UART_OutUDec(n / 10);
		n = n % 10;
	}
	UART_OutChar(n + '0');
}

void UART_OutChar(char data) {
	while ((UART0_FR_R&UART_FR_TXFF) != 0);
	UART0_DR_R = data;
}

void UARTIntHandler(void)
```

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

```
{
	uint32_t ui32Status;
	ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status
	UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

	switch (UARTCharGet(UART0_BASE)) {
	case 'B':
		UARTCharPut(UART0_BASE, 'B');
		UARTCharPut(UART0_BASE, 'l');
		UARTCharPut(UART0_BASE, 'u');
		UARTCharPut(UART0_BASE, 'e');
		UARTCharPut(UART0_BASE, ' ');
		UARTCharPut(UART0_BASE, 'l');
		UARTCharPut(UART0_BASE, 'e');
		UARTCharPut(UART0_BASE, 'd');
		UARTCharPut(UART0_BASE, ' ');
		UARTCharPut(UART0_BASE, 'O');
		UARTCharPut(UART0_BASE, 'N');
		UARTCharPut(UART0_BASE, '\n');
		UARTCharPut(UART0_BASE, '\r');

		GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
		SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
		break;
	case 'b':
		UARTCharPut(UART0_BASE, 'B');
		UARTCharPut(UART0_BASE, 'l');
		UARTCharPut(UART0_BASE, 'u');
		UARTCharPut(UART0_BASE, 'e');
		UARTCharPut(UART0_BASE, ' ');
		UARTCharPut(UART0_BASE, 'l');
		UARTCharPut(UART0_BASE, 'e');
		UARTCharPut(UART0_BASE, 'd');
		UARTCharPut(UART0_BASE, ' ');
		UARTCharPut(UART0_BASE, 'O');
		UARTCharPut(UART0_BASE, 'F');
		UARTCharPut(UART0_BASE, 'F');
		UARTCharPut(UART0_BASE, '\n');
		UARTCharPut(UART0_BASE, '\r');

		GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //blink LED
		SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
		break;
	case 'R':
		UARTCharPut(UART0_BASE, 'R');
		UARTCharPut(UART0_BASE, 'e');
		UARTCharPut(UART0_BASE, 'd');
		UARTCharPut(UART0_BASE, ' ');
		UARTCharPut(UART0_BASE, 'l');
		UARTCharPut(UART0_BASE, 'e');
		UARTCharPut(UART0_BASE, 'd');
		UARTCharPut(UART0_BASE, ' ');
		UARTCharPut(UART0_BASE, 'O');
		UARTCharPut(UART0_BASE, 'N');
		UARTCharPut(UART0_BASE, '\n');
		UARTCharPut(UART0_BASE, '\r');
```

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

```
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1); //blink LED
            SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
            break;
        case 'r':
            UARTCharPut(UART0_BASE, 'R');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'd');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, 'l');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'd');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, 'O');
            UARTCharPut(UART0_BASE, 'F');
            UARTCharPut(UART0_BASE, 'F');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
```

```
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0); //blink LED
            SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
            break;
        case 'G':
            UARTCharPut(UART0_BASE, 'G');
            UARTCharPut(UART0_BASE, 'r');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'n');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, 'l');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'd');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, 'O');
            UARTCharPut(UART0_BASE, 'N');
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
```

```
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3); //blink LED
            SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
            break;
        case 'g':
            UARTCharPut(UART0_BASE, 'G');
            UARTCharPut(UART0_BASE, 'r');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'n');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, 'l');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'd');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, 'O');
            UARTCharPut(UART0_BASE, 'F');
            UARTCharPut(UART0_BASE, 'F');
            UARTCharPut(UART0_BASE, '\n');
```

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

```
            UARTCharPut(UART0_BASE, '\r');

            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0); //blink LED
            SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec

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

            ADCIntClear(ADC0_BASE, 3); //clear adc conversion done flag before writing
            code that depends on it. change to sequence 2
            //Changed all sequence numbers below to sequence two
            ADCProcessorTrigger(ADC0_BASE, 3);

            while (!ADCIntStatus(ADC0_BASE, 3, false)) //wait for conversion to finish
            {
            } //if loop exited conversion is complete

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

            UART_OutUDec(ui32TempValueF);
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            break;
        }

}
```

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