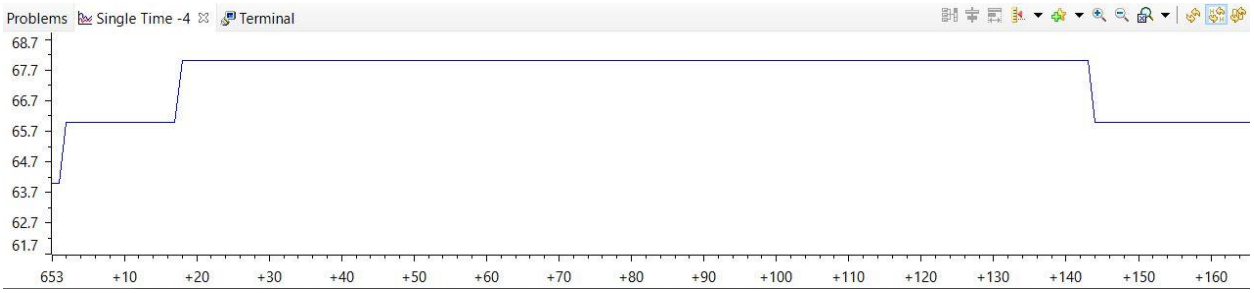
Jett Guerrero
Github root directory: https://github.com/guerrj1/Advanced-Embedded-Systems

Date Submitted: **10/29/2019**

Task 00: Execute provided code

Youtube Link: **https://youtu.be/7uMl-enhV9U**



Temperature Graph

Task 01:

Youtube Link: https://youtu.be/e0RSSJGJwac

Modified Code:

```
//Lab7 Task1
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/debug.h"
#include "driverlib/adc.h"
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "driverlib/rom.h"
#include "utils/cmdline.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/timer.h"
#include "driverlib/debug.h"

volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
volatile uint32_t TempF;

int main(void)
{
	uint32_t ui32Period;

	//clock initialization
```

```
	SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

	//enable settings
	SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

	//GPIO settings
	GPIOPinConfigure(GPIO_PA0_U0RX);
	GPIOPinConfigure(GPIO_PA1_U0TX);
	GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

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

	//characters to be printed
	UARTCharPut(UART0_BASE, 'D');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, 'v');
	UARTCharPut(UART0_BASE, 'i');
	UARTCharPut(UART0_BASE, 'c');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, ' ');
	UARTCharPut(UART0_BASE, 'T');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, 'm');
	UARTCharPut(UART0_BASE, 'p');
	UARTCharPut(UART0_BASE, ':');
	UARTCharPut(UART0_BASE, ' ');

	//ADC settings
	ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
	ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 32);
	ROM_ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
	ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_TS);
	ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_TS);
	ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_TS);
	ROM_ADCSequenceStepConfigure(ADC0_BASE,2,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
	ROM_ADCSequenceEnable(ADC0_BASE, 2);

	//Timer 1
	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
	TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);

	ui32Period = SysCtlClockGet()/2;
	TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period -1);
	IntEnable(INT_TIMER1A);
	TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
	IntMasterEnable();
	TimerEnable(TIMER1_BASE, TIMER_A);

	while (1)
	{
	}
}
```

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

```
//timer1 function
void Timer1IntHandler(void)
{
	uint32_t ui32ADC0Value[4];
	uint8_t arr[10];
	TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

	ADCIntClear(ADC0_BASE, 2);
	ADCProcessorTrigger(ADC0_BASE, 2);

	//reprints
	UARTCharPut(UART0_BASE, 'D');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, 'v');
	UARTCharPut(UART0_BASE, 'i');
	UARTCharPut(UART0_BASE, 'c');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, ' ');
	UARTCharPut(UART0_BASE, 'T');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, 'm');
	UARTCharPut(UART0_BASE, 'p');
	UARTCharPut(UART0_BASE, ':');
	UARTCharPut(UART0_BASE, ' ');

	while(!ADCIntStatus(ADC0_BASE, 2, false))
	{
	}
	//temperature conversation
	ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0Value);
	ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
	ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
	ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
	TempF = ui32TempValueF;
	int i = 0;

	//temp array
	while(ui32TempValueF != 0)
	{
		arr[i++] = (ui32TempValueF%10) + '0'; //ones place
		ui32TempValueF = ui32TempValueF / 10; //tens place
	}

	for(i=0; i<5; i++)
	{
		UARTCharPut(UART0_BASE, arr[i]);
	}

	UARTCharPut(UART0_BASE, 'F'); //displays F for farenheit

	UARTCharPut(UART0_BASE, '\n');  //prints new line
	UARTCharPut(UART0_BASE, '\r');  //return line
}
```

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

Jett Guerrero
Github root directory: https://github.com/guerrj1/Advanced-Embedded-Systems

---

Task 02:

Youtube Link: https://youtu.be/IR2B1WxRfbI

Modified Code:

```
//Lab7 Task2
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/debug.h"
#include "driverlib/adc.h"
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "driverlib/rom.h"
#include "utils/cmdline.h"

volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

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

	//enable settings
	SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

	//GPIO settings
	GPIOPinConfigure(GPIO_PA0_U0RX);
	GPIOPinConfigure(GPIO_PA1_U0TX);
	GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

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

	IntMasterEnable();
	IntEnable(INT_UART0);
	UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

	//prints initial text
	UARTCharPut(UART0_BASE, 'E');
	UARTCharPut(UART0_BASE, 'n');
```

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

```
	UARTCharPut(UART0_BASE, 't');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, 'r');
	UARTCharPut(UART0_BASE, ' ');
	UARTCharPut(UART0_BASE, 'T');
	UARTCharPut(UART0_BASE, 'h');
	UARTCharPut(UART0_BASE, 'e');
	UARTCharPut(UART0_BASE, ' ');
	UARTCharPut(UART0_BASE, 'C');
	UARTCharPut(UART0_BASE, 'm');
	UARTCharPut(UART0_BASE, 'd');
	UARTCharPut(UART0_BASE, ':');
	UARTCharPut(UART0_BASE, ' ');

	//ADC settings
	ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
	ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 32);
	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);

	while (1)
	{
	}
}

//UART function
void UARTIntHandler(void)
{

	uint32_t ui32Status;
	ui32Status = UARTIntStatus(UART0_BASE, true);
	uint32_t ui32ADC0Value[4];
	uint8_t arr[10];
	UARTIntClear(UART0_BASE, ui32Status);

	char command; //variable for letter

	while(UARTCharsAvail(UART0_BASE))
	{
		command = UARTCharGet(UART0_BASE);
		UARTCharPut(UART0_BASE, command);

		//red on
		if(command == 'R')
		{
			GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
2);
			UARTCharPut(UART0_BASE, ':');
			UARTCharPut(UART0_BASE, ' ');
			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');
        }

        //red off
        if(command == 'r')
        {
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
0);
                UARTCharPut(UART0_BASE, ':');
                UARTCharPut(UART0_BASE, ' ');
                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');
        }

        //green on
        if(command == 'G')
        {
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
8);
                UARTCharPut(UART0_BASE, ':');
                UARTCharPut(UART0_BASE, ' ');
                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');
        }

        //green off
        if(command == 'g')
        {
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
0);
                UARTCharPut(UART0_BASE, ':');
                UARTCharPut(UART0_BASE, ' ');
                UARTCharPut(UART0_BASE, 'G');
                UARTCharPut(UART0_BASE, 'R');
                UARTCharPut(UART0_BASE, 'E');
```

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

```
                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');
        }

        //blue on
        if(command == 'B')
        {
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
4);
                UARTCharPut(UART0_BASE, ':');
                UARTCharPut(UART0_BASE, ' ');
                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');
        }

        //blue off
        if(command == 'b')
        {
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
0);
                UARTCharPut(UART0_BASE, ':');
                UARTCharPut(UART0_BASE, ' ');
                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');
        }

        //temp reading
        if(command == 'T')
        {
                UARTCharPut(UART0_BASE, ':'); //space
                UARTCharPut(UART0_BASE, ' ');
```

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

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

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

                        //temperature conversation
                        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;

                        int i = 0;

                        //temp array
                        while(ui32TempValueF != 0)
                        {
                               arr[i++] = (ui32TempValueF%10) + '0'; //ones place
                               ui32TempValueF = ui32TempValueF / 10; //tens place
                        }

                        for(i=0; i<3; i++)
                        {
                               UARTCharPut(UART0_BASE, arr[i]);
                        }

                        UARTCharPut(UART0_BASE, 'F'); //displays F for farenheit
                }

                UARTCharPut(UART0_BASE, '\n');  //prints new line
                UARTCharPut(UART0_BASE, '\r');  //return line
        }

        //reprints text
        UARTCharPut(UART0_BASE, 'E');
        UARTCharPut(UART0_BASE, 'n');
        UARTCharPut(UART0_BASE, 't');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, 'r');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'T');
        UARTCharPut(UART0_BASE, 'h');
        UARTCharPut(UART0_BASE, 'e');
        UARTCharPut(UART0_BASE, ' ');
```

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

```
        UARTCharPut(UART0_BASE, 'C');
        UARTCharPut(UART0_BASE, 'm');
        UARTCharPut(UART0_BASE, 'd');
        UARTCharPut(UART0_BASE, ':');
        UARTCharPut(UART0_BASE, ' ');

}
```
