Student Name
Github root directory: (insert link here)

Date Submitted:

Task 00: Execute provided code

Youtube Link: https://youtu.be/xWmZW3kB_x8

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/pwm.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/rom.h"

// 55Hz to control the servo
#define PWM_FREQUENCY 55

int main(void)
{
    // program the PWM, 83 is the center to create a 1.5ms pulse to the PWM
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;


    // run the clk at 40MHz

ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    //pwm module clocked by the sys clk through a divider, (625 khz)
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_64);


    // enable the pwm1 and gpiod modules (for output on pd0)
    // and gpiof module (for the launchpad buttons on pf0 and pf4)
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);



    // PORT D PIN 0 CONFIGURED as a pwm outputpin for module 1, pwm generator 0
    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);


    // Port F pin 0 and pin 4 are connected to the S2 and S1 switches on the
LaunchPad.
    // In order for the state of the pins to be read in our code, the pins must be
pulled up.
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
    ROM_GPIODirModeSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0, GPIO_DIR_MODE_IN);
    ROM_GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);



    // divide the pwm clock by the desired frequency to determine the count loaded
into the load register
    // config module 1 pwm generator 0
    ui32PWMClock = SysCtlClockGet() / 64;
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
    PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);



    // FINAL PWN SETTINGS AND ENABLE IT
    //first line setsthe pulse width
    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 100);

    // pwm module 1, gen 0 needs to be enabled as an output and enabled
    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0);



    // Read pf4 pin to see if sw1 is pressed
    //
    while(1)
    {

        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00)
        {
            ui8Adjust--;
            if (ui8Adjust < 56)
            {
                ui8Adjust = 56;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }

        //read the pf0 pin to see if sw2 is pressed
        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_0)==0x00)
        {
            ui8Adjust++;
            if (ui8Adjust > 111)
            {
                ui8Adjust = 111;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }
```

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

```
        // determines the speed
        ROM_SysCtlDelay(100000);
    }

}


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

Task 01:

Youtube Link:
https://youtu.be/c6O_88lv1fo

Modified Schematic (if applicable):

Modified Code:

```
// Insert code here
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/pwm.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/rom.h"

// 55Hz to control the servo
#define PWM_FREQUENCY 55

int main(void)
{
    // program the PWM, 83 is the center to create a 1.5ms pulse to the PWM
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;


    // run the clk at 40MHz

ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    //pwm module clocked by the sys clk through a divider, (625 khz)
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_64);


    // enable the pwm1 and gpiod modules (for output on pd0)
    // and gpiof module (for the launchpad buttons on pf0 and pf4)
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);



    // PORT D PIN 0 CONFIGURED as a pwm outputpin for module 1, pwm generator 0
    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);


    // Port F pin 0 and pin 4 are connected to the S2 and S1 switches on the
LaunchPad.
    // In order for the state of the pins to be read in our code, the pins must be
pulled up.
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
    ROM_GPIODirModeSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0, GPIO_DIR_MODE_IN);
    ROM_GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);



    // divide the pwm clock by the desired frequency to determine the count loaded
into the load register
    // config module 1 pwm generator 0
    ui32PWMClock = SysCtlClockGet() / 64;
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
    PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);



    // FINAL PWN SETTINGS AND ENABLE IT
    //first line setsthe pulse width
    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 100);

    // pwm module 1, gen 0 needs to be enabled as an output and enabled
    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0);



    // Read pf4 pin to see if sw1 is pressed
    //
    while(1)
    {

        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00)
        {
            ui8Adjust--;
            if (ui8Adjust < 20)
            {
                ui8Adjust = 20;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }

        //read the pf0 pin to see if sw2 is pressed
        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_0)==0x00)
        {
            ui8Adjust++;
            if (ui8Adjust > 150)
            {
                ui8Adjust = 150;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }



        // determines the speed
        ROM_SysCtlDelay(100000);
    }

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

Task 02:

Youtube Link: https://youtu.be/YAntXP3kxaI

Modified Schematic (if applicable):

Modified Code:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/interrupt.h"
// Rather than adding the peripheral driver library, we call them from rom. less code size
#define TARGET_IS_BLIZZARD_RB1
#include "driverlib/rom.h"


    //aray storing data read from ADC fifo
    uint32_t ui32ADC0Value[4];
    // variables for calculating temp from sensor data
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;
    volatile uint32_t ui32Period;


void Timer1IntHandler(void)
{
    TimerIntClear(TIMER1_BASE,TIMER_A);// Always clear the interrupt for the values
that may depend on it in the future

    // Clear the ADC interrup status flag
             ROM_ADCIntClear(ADC0_BASE, 2);
            // Trigger ADC conversion with software
            ROM_ADCProcessorTrigger(ADC0_BASE, 2);


           // waith for the conversion to complete
           while(!ROM_ADCIntStatus(ADC0_BASE, 2, false))
           {
           }



           // we can read the ADC value from the ADC sample sequencer 1 FIFO
           ROM_ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0Value);

           // calculate the average of the temperature sensor data
           ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
           // calculate celsius value
           ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
           // calculate farenheit value
           ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;



    if(ui32TempValueF > 72)
    {

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);   // turn on the led when the
temperature is grreater than 72
    }
    else
        if(ui32TempValueF <= 72 )
        {


            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);   //turn off the led if the
temperature goes below 72
        }

}
```

```
int main(void)
{


    // set clock to 40 MHz

ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);                // enable GPIO
peripherals
       GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);   //
configure pins as outputs for LEDs


       SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);   // enable clock to timer1
           TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);     // configure timer 1
in periodic mode

          // SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);    // enable clock to
peripherals
               //TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);   // Configure
Timer 0 in periodic mode

               ui32Period = (SysCtlClockGet() / 1) / 2;     // sets the delay
               TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period -1);   // load into
Timer's Interval Load register

               IntEnable(INT_TIMER1A);  // enables specific vector associated with
Timer 0A
               TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT); // enables a specific
event within the timer to generate an interrupt (on timeouts)
               IntMasterEnable();  // master interrupt enable for all interrupts

               TimerEnable(TIMER1_BASE, TIMER_A);// finally enable the timer



    // ENABLE THE ADC0 Peripheral
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);


    // number of samples to be averaged  32 for task 2
    ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 32);




    //configure the ADC Sequencer ( ADC0, sample sequencer 1, processor triggers the
sequence, highest priority)
    ROM_ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);


    // configure the four steps in the sequencer, 0-2 on sequencer 1 to sample temp
(ADC_CTL_TS), ADC0, sequencer 1, step 0-2...
```

```
    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);





    // The last must sample the temp and configure the interrupt flag to be set when
sample is done. Tell ADC logic that this is the last conversion on seq 1
    ROM_ADCSequenceStepConfigure(ADC0_BASE,2,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    // Enable the Sequencer 1 adc
    ROM_ADCSequenceEnable(ADC0_BASE, 2);
    ADCIntEnable(ADC0_BASE,2);
    while(1)
    {


    }
}

// Insert code here

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

Student Name
Github root directory: (insert link here)

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