Ron Joshua Recrio
Github root directory: https://github.com/recrio/submissions.2/tree/master/Lab%205

Date Submitted: 12/13/2019

Task 00: Execute provided code

```
#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"


#define PWM_FREQUENCY 55

int main(void)
{
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;


ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_64);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);

    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);

    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);

    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0);

    while(1)
    {
```

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

```
        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);
        }

        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);
        }

        ROM_SysCtlDelay(100000);
    }

}
```

Youtube Link:
https://youtu.be/L1f6kmuhqnc

Task 01:

Youtube Link:
https://youtu.be/fL_zFaSo7zU

Modified Schematic (if applicable):

Modified Code:

```
#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"

#define PWM_FREQUENCY 55

int main(void)
{
    volatile uint32_t ui32Load;
```

```
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 0;
    bool dirLeft;


ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_64);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);

    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);

    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);

    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0);



    while(1)
    {
        if (ui8Adjust < 1)
        {
            dirLeft = true;
        }
        if (ui8Adjust > 120)
        {
            dirLeft = false;
        }

        if (dirLeft)
        {
            ui8Adjust++;
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }
        else
        {
            ui8Adjust--;
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }
```

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

```
        ROM_SysCtlDelay(100000);
    }

}
```

Task 02:

Youtube Link:
https://youtu.be/szF0VryefQo

Modified Schematic (if applicable):

Modified Code:

```
#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"


#define PWM_FREQUENCY 55

int main(void)
{
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 90;


ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_64);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    ROM_GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
    ROM_GPIOPinConfigure(GPIO_PF1_M1PWM5);

    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);
```

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

```
    ui32PWMClock = SysCtlClockGet() / 64;
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
    PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, 100);

    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, ui8Adjust);
    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_2);

    while(1)
    {

        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00)
        {
            ui8Adjust--;
            if (ui8Adjust < 10)
            {
                ui8Adjust = 10;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, ui8Adjust);
        }

        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_0)==0x00)
        {
            ui8Adjust++;
            if (ui8Adjust > 90)
            {
                ui8Adjust = 90;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, ui8Adjust);
        }

        ROM_SysCtlDelay(100000);
    }

}
```

Task 03:

Modified Code:

```
#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/adc.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/rom.h"
```

Ron Joshua Recrio
Github root directory: https://github.com/recrio/submissions.2/tree/master/Lab%205

```
#define PWM_FREQUENCY 55 // PWM frequency at 55Hz

int main(void)
{
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint32_t ui32Adjust;


ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_64); // PWM clock is set to 625kHz after div
by 64

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // enable PWM 1 peripheral
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // enable GPIO PORTD to use as
output for DC motor
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); // enable GPIO PORTE using
analog input 8 (PE5)
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); //enable ADC0 peripheral

    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0); // set PORTD as a PWM output pin
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0); // use PWM motion control module 1
    ROM_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_5); // use PE5 (AIN8 - channel 8)
for potentiometer


    //setup of PWM period
    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);

    //setup of PWM
    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true); // set generator 0 as output
    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0); // enable pwm and generator 0

    //initialize variables ADC for potentiometer
    uint32_t ui32ADC0Value[4];  // array to store samples of ADC with 4 steps
    volatile uint32_t ui32ADCAvg; // store avg value

    //configure ADC
    ROM_ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0); // using ADC
sample sequencer 1 (SS1), set as the highest priority, and processor will trigger ADC
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH8); // ADC sample step 0
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH8); // ADC sample step 1
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH8); // ADC sample step 2
    ROM_ADCSequenceStepConfigure(ADC0_BASE,1, 3, ADC_CTL_CH8|ADC_CTL_IE|ADC_CTL_END);
//ADC sample step 3, set ADC interrupt flag, end sampling
    ROM_ADCSequenceEnable(ADC0_BASE, 1);    // enable ADC0


    while(1)
    {
        ROM_ADCIntClear(ADC0_BASE, 1);   // clear ADC interrupt
        ROM_ADCProcessorTrigger(ADC0_BASE, 1); // processor begins to trigger ADC
```

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

```
        while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) // wait for ADC conversion..
        {
        }

        ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); // get ADC value from
samples

        ui32ADCAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
        ui32Adjust = ui32ADCAvg; // store ADC avg value into the ui32Adjust variable
        ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui32Adjust * ui32Load / 1000); //
set the width of the PWM using the ui32Adjust value - DC motor speed set
    }

}
```