Ricky Perez
Github root directory: https://github.com/RickyPerez79/da_submissions_2019

Date Submitted: **12/13/19**

Task 00: Execute provided code

Youtube Link: **https://youtu.be/b07wqF9gMsw**

Task 01:

Youtube Link: https://youtu.be/0xqXa5RrSw4

Modified Code:

```
// Insert code here
// task 01
#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"

// We'll use a 55Hz base frequency to control the servo
#define PMW_FREQUENCY 55
/*
The following variables will be used to program the PWM. They are
defined as "volatile"
to guarantee that the compiler will not eliminate them, regardless of
the optimization setting.
The ui8Adjust variable will allow us to adjust the position of the
servo.
83 is thecenter position to create a 1.5mS pulse from the PWM.

Here's how we came up with 83 … In the servo control code (covered
shortly) we're going to divide the PWM period by 1000.
Since the programmed frequency is 55HZ and the period is 18.2mS,
dividing that by 1000 gives us a pulse resolution of 1.82µS.
 Multiplying that by 83 gives us a pulse-width of 1.51mS. Other
selections for the resolution, etc.
```

```
would be just as valid as long as they produced a 1.5mS pulse-width.
Take care though to
be sure that your numbers will fit within the 16-bit registers.
 */

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

    /*
      Let's run the CPU at 40MHz. The PWM module is clocked by the
system clock through
      a divider, and that divider has a range of 2 to 64.
      By setting the divider to 64, it will run the PWM clock at 625
kHz.
      Note that we're using the ROM versions to reduce our code size.
     */


ROM_SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSC
TL_XTAL_16MHZ);
    ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_64);

    /*
     We need to enable the PWM1 and GPIOD modules (for the PWM output
on PD0) and
     the 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 (PD0) must be configured as a PWM output
    pin 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.
    (The BUTTONSPOLL API could do this for us, but that API checks for
individual button
    presses rather than a button being held down). Pulling up a GPIO
pin is normally pretty
    straight-forward, but PF0 is considered a critical peripheral
since it can be configured to
    be a NMI input. Since this is the case, we will have to unlock the
GPIO commit control
    register to make this change

    The first three lines below unlock the GPIO commit control
register,
    the fourth configures PF0 & 4 as inputs and the fifth configures
the
    internal pull-up resistors on both pins. The drive strength
setting is merely
    a place keeper and has no function for an input.
     */

    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);

    /*
    The PWM clock is SYSCLK/64 (set in step 12 above). Divide the PWM
clock by the desired frequency (55Hz)
    to determine the count to be loaded into the Load register. Then
subtract 1 since the counter down-counts to zero.
    Configure module 1 PWM generator 0 as a down-counter and load the
count value.
     */

    ui32PWMClock = SysCtlClockGet() / 64;
    ui32Load = (ui32PWMClock / PMW_FREQUENCY) - 1;
    PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);

    /*
     Now we can make the final PWM settings and enable it. The first
line sets the pulsewidth.
```

```
     The PWM Load value is divided by 1000 (which determines the
minimum resolution for the servo)
     and the multiplied by the adjusting value. These numbers could be
changed to provide more or less resolution.
     In lines two and three, PWM module 1, generator 0 needs to be
enabled as an output and enabled to run.
     */

    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(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00) // if
sw1 is pressed, will go 0 to 180 degrees
        {
            while(ui8Adjust > 20)
            {
                ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust *
ui32Load / 1000);
                ui8Adjust--;
            }
        }

        /*
         The next code will read the PF0 pin to see if SW2 is pressed
to increment
         the pulse width. The maximum limit is set to reach 2.0mS.
         */
        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_0)==0x00) // if
sw2 is pressed, will go 0 to 180 degrees
        {
            while(ui8Adjust < 115)
            {
                ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust *
ui32Load / 1000);
                ui8Adjust++;
            }
        }


        /*
         This final line determines the speed of the loop.
         If the servo moves too quickly or too slowly for you,
```

```
         feel free to change the count to your liking.
         */

        ROM_SysCtlDelay(100000);


    }



    return 0;
}
```

Task 02:

Youtube Link: https://youtu.be/AFTTT5uO_lU

Modified Code:

```
// Insert code here
// task 02
#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"

int main(void)
{

    int i; //for loop temp variable
    SysCtlPWMClockSet(SYSCTL_PWMDIV_1);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIOF
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); //enable PWM1

    GPIOPinConfigure(GPIO_PF1_M1PWM5); //assign pin to PWM
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3); //set pins as output
```

```
    PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN);

    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, 400); //set period to 400
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 400);

    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, 100);

    PWMGenEnable(PWM1_BASE, PWM_GEN_2);
    PWMGenEnable(PWM1_BASE, PWM_GEN_3);

    PWMOutputState(PWM1_BASE, (PWM_OUT_5_BIT | PWM_OUT_6_BIT |
PWM_OUT_7_BIT), true);
    //output PWM pins

    while(1)
    {
        for(i = 40; i < 360 ; i+=5){ //increase from 10% (40) duty
cycle to 90%  (360)
          ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, i); //PF1
          SysCtlDelay(100000);
        }
        for(i = 360; i > 40; i-=5 ){ //decrease from 90% (360) duty
cycle to 10%  (40)
          ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, i); //PF1
          SysCtlDelay(100000);
        }

    }

}
```
