Saul Alejandro Mendoza Guzman
Github root directory: https://github.com/mendos1/CPE_403/Lab6

Date Submitted: 10/29/2019

Task 00: Execute provided code

Youtube Link:
https://www.youtube.com/watch?v=ehvJkJt1WtU

Task 00:

```
/*
 * main.c
 *
 *  Created on: Oct 14, 2019
 *      Author: rexaul
 */



#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);
```

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

```
  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(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);
  }
}
```

Task 01:

Grap
Youtube Link:
https://www.youtube.com/watch?v=iQ7Ki3OEpjw
Modified Schematic (if applicable):

Modified Code:

```
/*
 * main.c
 *
 *  Created on: Oct 14, 2019
 *      Author: rexaul
 */
```

Saul Alejandro Mendoza Guzman
Github root directory: https://github.com/mendos1/CPE_403/Lab6

```
#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 // This sets the base frequency to control the servo: STEP 1

int main(void)
{
    /*  STEP 2: lines 31-34 are defined as volatile to ensure compiler wont eliminate
them regardless of optimization settings
     *          the ui8Adjust var will allow us to adjust the position of the servo.
     *          83 is the center position to create a 1.5ms pulse
     *          (PWM_FREQUENCY / 1000) * ui8Adjust
     */
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;

    /*  STEP 3: lines 40-41 here we will 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 divder to 64
it will run the PWM clock at 625kHz.
     */

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

    /*  STEP 4: lines 45-47 here we need to enable the PWM1 and the 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);

    /*  STEP 5: lines 52-54 here port D pin 0 (PD0) must be configured as a PWM
output pin for module 1, PWM generator 0 (check out the schematic)
     *
     */
    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);

    /*  STEP 6: lines 63-67 here PORT F pin 0 and pin 4 are connected to the S2 and
S1 switches on the LaunchPad.
```

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

```
     *          in order for the state of the pins to be read the pins must be pulled
up. 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. We have to
     *          unlock the GPIO commit control register to make this change. This
feature was mentioned in chapter 3 of the workshop.
     *          The first three lines unlock the GPIO commit control register and the
fourth line configures PF0 and PF4 as inputs
     *          and the fifth configures the internal pull-up register on both the
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);

    /*  STEP 7: lines 74-77 here the PWM clock is SYSCLK/64. Divide the PWM clock b
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 / PWM_FREQUENCY) - 1;
    PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);

    /*  STEP 8: lines 85-87 here we make the final PWM settings and enable it.
     *          First Line: sets the pulse-width. The PWM Load Value is divided by
1000 (wich determines the min resolution for the
     *                      servo) and then multiplied by the adjusting value. These
numbers could be changed to provide more or less resolution
     *          Next 2 Lines: PWM module 1 generator 0 needs to be set 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){
        // ************************************* CONTROLLING THE SERVO
*****************************************
        /*  STEP 9: lines 100-106 here we read PF4 to see if SW1 was pressed. No
debouncing is needed since we're not
         *          looking for individual key presses. Each time this code runs it
will decrement the adjust variable
         *          by one unless it reaches the lower 1ms limit. This number like
the center and upper positions was
```

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

```
         *          determined by measuring the output of the PWM pulse width
register with the new value. This load is
         *          done asynchronously to the output. In a more critical design you
might want to consult the databook
         *          concerning making this load differently
         *
         */
        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00){
            ui8Adjust--;
            if (ui8Adjust < 1){
                ui8Adjust = 1;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }

        /*  STEP 10: lines 112-118 here we will read 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){
            ui8Adjust++;
            if (ui8Adjust > 111){
                ui8Adjust = 111;
            }
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
        }

        /*  STEP 11: line 123 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);
    }
}
```

Saul Alejandro Mendoza Guzman
Github root directory: https://github.com/mendos1/CPE_403/Lab6

Task 02:

Youtube Link:
https://www.youtube.com/watch?v=31bLIyMJj40

Modified Schematic (if applicable):

Modified Code:

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

void delayMS(int ms) {
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*ms ) ;
}

int
main(void)
{
    //Set the clock
   SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC |   SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

   //Configure PWM Clock to match system
   SysCtlPWMClockSet(SYSCTL_PWMDIV_1);

   // Enable the peripherals used by this program.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);  //The Tiva Launchpad has two modules
(0 and 1). Module 1 covers the LED pins

    //Configure PF1,PF2,PF3 Pins as PWM
    GPIOPinConfigure(GPIO_PF1_M1PWM5);
    GPIOPinConfigure(GPIO_PF2_M1PWM6);
    GPIOPinConfigure(GPIO_PF3_M1PWM7);
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

    //Configure PWM Options
    //PWM_GEN_2 Covers M1PWM4 and M1PWM5
    //PWM_GEN_3 Covers M1PWM6 and M1PWM7 See page 207 4/11/13 DriverLib doc
    PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);

    //Set the Period (expressed in clock ticks)
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, 320);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 320);
```

```
    //Set PWM duty-50% (Period /2)
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5,100);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6,100);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7,100);

    // Enable the PWM generator
    PWMGenEnable(PWM1_BASE, PWM_GEN_2);
    PWMGenEnable(PWM1_BASE, PWM_GEN_3);

    // Turn on the Output pins
    PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT | PWM_OUT_6_BIT | PWM_OUT_7_BIT, true);

    //Fade
    bool fadeUp = true;
    unsigned long increment = 10;
    unsigned long pwmNow = 100;
    while(1)
    {
        delayMS(20);
        if (fadeUp) {
            pwmNow += increment;
            if (pwmNow >= 320) { fadeUp = false; }
        }
        else {
            pwmNow -= increment;
            if (pwmNow <= 10) { fadeUp = true; }
        }
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5,pwmNow);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6,pwmNow);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7,pwmNow);
    }

}
```

Task 02:

Youtube Link:
https://www.youtube.com/watch?v=6XN70W53hlg

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 // This sets the base frequency to control the servo: STEP 1
void delayMS(int ms) {
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*ms ) ;
}
int main(void)
{
    /*  STEP 2: lines 31-34 are defined as volatile to ensure compiler wont eliminate
them regardless of optimization settings
     *          the ui8Adjust var will allow us to adjust the position of the servo.
     *          83 is the center position to create a 1.5ms pulse
     *          (PWM_FREQUENCY / 1000) * ui8Adjust
     */
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;

    /*  STEP 3: lines 40-41 here we will 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 divder to 64
it will run the PWM clock at 625kHz.
     */

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

    /*  STEP 4: lines 45-47 here we need to enable the PWM1 and the 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);

    //Configure PF1,PF2,PF3 Pins as PWM
    GPIOPinConfigure(GPIO_PF1_M1PWM5);
    GPIOPinConfigure(GPIO_PF2_M1PWM6);
    GPIOPinConfigure(GPIO_PF3_M1PWM7);
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

    /*  STEP 5: lines 52-54 here port D pin 0 (PD0) must be configured as a PWM
output pin for module 1, PWM generator 0 (check out the schematic)
     *
     */
    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);

    /*  STEP 6: lines 63-67 here 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 the pins must be pulled
up. 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. We have to
```

```
     *          unlock the GPIO commit control register to make this change. This
feature was mentioned in chapter 3 of the workshop.
     *          The first three lines unlock the GPIO commit control register and the
fourth line configures PF0 and PF4 as inputs
     *          and the fifth configures the internal pull-up register on both the
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);

    /*  STEP 7: lines 74-77 here the PWM clock is SYSCLK/64. Divide the PWM clock b
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 / PWM_FREQUENCY) - 1;
    PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);

    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, 320);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 320);

    /*  STEP 8: lines 85-87 here we make the final PWM settings and enable it.
     *          First Line: sets the pulse-width. The PWM Load Value is divided by
1000 (wich determines the min resolution for the
     *                      servo) and then multiplied by the adjusting value. These
numbers could be changed to provide more or less resolution
     *          Next 2 Lines: PWM module 1 generator 0 needs to be set as an output
and enabled to run.
     *
     */
    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5,100);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6,100);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7,100);

    ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT | PWM_OUT_6_BIT | PWM_OUT_7_BIT, true);

    ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0);
    PWMGenEnable(PWM1_BASE, PWM_GEN_2);
    PWMGenEnable(PWM1_BASE, PWM_GEN_3);

    //Fade
    bool fadeUp = true;
```

```
    unsigned long increment = 10;
    unsigned long pwmNow = 100;

    while(1){
        delayMS(20);
        // ************************************* CONTROLLING THE SERVO
*****************************************
        /*  STEP 9: lines 100-106 here we read PF4 to see if SW1 was pressed. No
debouncing is needed since we're not
         *          looking for individual key presses. Each time this code runs it
will decrement the adjust variable
         *          by one unless it reaches the lower 1ms limit. This number like
the center and upper positions was
         *          determined by measuring the output of the PWM pulse width
register with the new value. This load is
         *          done asynchronously to the output. In a more critical design you
might want to consult the databook
         *          concerning making this load differently
         *
         */
        if(ROM_GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0x00){
            if (fadeUp == 1) {
                pwmNow += increment;
                if (pwmNow >= 320) { fadeUp = false; }
            }
        }

        /*  STEP 10: lines 112-118 here we will read 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 (fadeUp == 0) {
                pwmNow -= increment;
                if (pwmNow <= 10) { fadeUp = true; }
            }
        }

        /*  STEP 11: line 123 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.
         */
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5,pwmNow);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6,pwmNow);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7,pwmNow);
    }
}
```

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