

### Task 00: Execute the supplied 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)
{
    //variables to program the PWM
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;

    //run CPU at 40MHz

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

    //enable PWM1 and GPIO modules
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    //port D pin 0(PD0) must be configured as a PWM
    ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);

    //unlock the GPIO commit control register
    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;
    //configures PF0 & 4 as inputs
    ROM_GPIODirModeSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0,GPIO_DIR_MODE_IN);
    //configures the internal pull-up resistors on both pins
    ROM_GIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

    //PWM clock is SYSCLK/64 then divide it by frequency to be loaded
    ui32PWMClock = SysCtlClockGet()/64;
    ui32Load = (ui32PWMClock/PWM_FREQUENCY)-1;
    //Configure module 1 PWM generator 0 as a down-counter and load value
    PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);
```

```
//enable PWM settings setting the pulse-width
ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load/1000);
//PWM module 1, generator 0 needs to be enabled
ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0);

while(1)
{
    //read the PF4 pin to see if SW1 is pressed
    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 to increment the pulse width
    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);
    }
    //determines the speed of the loop
    ROM_SysCtlDelay(100000);
}
}
```

**Task 01:** Change PWM duty cycle from 10% to 90% to control the brightness of the LED at PF1.

```
#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)
{
```

```
//variables to program the PWM
volatile uint32_t ui32Load;
volatile uint32_t ui32PWMClock;
volatile uint8_t ui8Adjust;
ui8Adjust = 83;

//run CPU at 40MHz

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

//enable PWM1 and GPIO modules
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

//port D pin 0(PD0) must be configured as a PWM
ROM_GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
ROM_GPIOPinConfigure(GPIO_PF1_M1PWM5);

//unlock the GPIO commit control register
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;
//configures PF0 & 4 as inputs
ROM_GPIODirModeSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0,GPIO_DIR_MODE_IN);
//configures the internal pull-up resistors on both pins
ROM_GIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

//PWM clock is SYSCLK/64 then divide it by frequency to be loaded
ui32PWMClock = SysCtlClockGet()/64;
ui32Load = (ui32PWMClock / PWM_FREQUENCY)-1;
//Configure module 1 PWM generator 0 as a down-counter and load value
PWMPGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
PWMPGenPeriodSet(PWM1_BASE, PWM_GEN_2, ui32Load);

//enable PWM settings setting the pulse-width
ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, ui8Adjust * ui32Load/1000);
//PWM module 1, generator needs to be enabled
ROM_PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_2);

while(1)
{
    //read the PF4 pin to see if SW1 is pressed
    if(ROM_GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)==0x00)
    {
        //decrement until 1ms limit--- from 10% duty cycle
        ui8Adjust--;
        if(ui8Adjust < 100)
        {
            ui8Adjust = 100;
        }
        ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, ui8Adjust * ui32Load/1000);
    }
}
```

```
}
//read the PF0 pin to see if SW2 is pressed to increment the pulse width
if(ROM_GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0)==0X00)
{
    //increment until 1.9ms ---to 90% duty cycle
    ui8Adjust++;
    if(ui8Adjust > 900)
    {
        ui8Adjust = 900;
    }
    ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, ui8Adjust * ui32Load/1000);
}
//determines the speed of the loop
ROM_SysCtlDelay(100000);

}

}
```

**Task 02:** Change PWM duty cycle from 10% to 90% to control the brightness of all three LED at PF1, PF2, and PF3 using three nested “for loops”.

```
#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)
{
    //variables to program the PWM
    volatile uint32_t ui32Load;
    volatile uint32_t ui32PWMClock;
    volatile uint8_t ui8Adjust;
    ui8Adjust = 83;

    //run CPU at 40MHz

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

    //enable PWM1 and GPIOD modules
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    //port D pin 0(PD0) must be configured as a PWM
```

```
ROM_GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
ROM_GPIOPinConfigure(GPIO_PF1_M1PWM5);
ROM_GPIOPinConfigure(GPIO_PF2_M1PWM6);
ROM_GPIOPinConfigure(GPIO_PF3_M1PWM7);

//PWM clock is SYSCLK/64 then divide it by frequency to be loaded
ui32PWMClock = SysCtlClockGet()/64;
ui32Load = (ui32PWMClock / PWM_FREQUENCY)-1;
//Configure module 1 PWM generator 0 as a down-counter and load value
PWMPGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
PWMPGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN);
PWMPGenPeriodSet(PWM1_BASE, PWM_GEN_2, ui32Load);
PWMPGenPeriodSet(PWM1_BASE, PWM_GEN_3, ui32Load);

//enable PWM settings setting the pulse-width
ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5_BIT, ui8Adjust * ui32Load/1000);
ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6_BIT, ui8Adjust * ui32Load/1000);
ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7_BIT, ui8Adjust * ui32Load/1000);

//PWM module 1, generator needs to be enabled
ROM_PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT|PWM_OUT_6_BIT|PWM_OUT_7_BIT, true);
ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_2);
ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_3);

while(1)
{
    //variables for for loops
    uint16_t red;
    uint16_t green;
    uint16_t blue;;

    for(red = 100; red < 900; red++)
    {
        ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, red * ui32Load/1000);
        ROM_SysCtlDelay(10000);
        for(blue = 100; blue < 900; blue++)
        {
            ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, blue * ui32Load/1000);
            ROM_SysCtlDelay(10000);
            for(green = 100; green < 900; green++)
            {
                ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, green *
ui32Load/1000);
                ROM_SysCtlDelay(10000);
            }
        }
    }

    for(red = 100; red >= 100; --red)
    {
        ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, red * ui32Load/1000);
        ROM_SysCtlDelay(10000);
    }
    for( blue = 100; blue >= 100; --blue)
    {
```

```
        ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, blue *ui32Load/1000);  
        ROM_SysCtlDelay(10000);  
    }  
    for(green = 100; green >= 100; --green)  
    {  
        ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, green *ui32Load/1000);  
        ROM_SysCtlDelay(10000);  
    }  
}
```