**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 GPIOD modules

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

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\_GPIOPadConfigSet(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;

//Cconfugure 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.**

**int** **main**(**void**)

{

.

.

.

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

ROM\_GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

ROM\_GPIOPinConfigure(GPIO\_PF1\_M1PWM5);

.

.

.

//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\_2, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(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”.**

**int** **main**(**void**)

{

.

.

.

//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

**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, ui32Load);

**PWMGenPeriodSet**(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);

}

}

}