**Date Submitted: 10/25**

**Task 01:**

Youtube Link: [**https://youtu.be/zA6AKIJ3z7A**](https://youtu.be/zA6AKIJ3z7A)

**Modified Code:**

**Changed the code in the while loop. Replaced the *if* structure from task 00 with motor position and delays. The first motor position is 0 degrees followed by a 3-second delay. Then, the second motor position is 165 degrees followed by a 3-second delay. Although instructed to do 180 degrees, the PowerHD 3001 Servo can only go to 165 degrees max as specified by the datasheet.**

**#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

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

{

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint8\_t ui8Adjust;

ui8Adjust = 83; // Adjusts the position of the servo

// Sets up the system clock and PWM clock

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

// Enables PWM1, GPIOD, GPIOF

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

// Configure PD0 as a PWM

ROM\_GPIOPinTypePWM(GPIO\_PORTD\_BASE, GPIO\_PIN\_0);

ROM\_GPIOPinConfigure(GPIO\_PD0\_M1PWM0);

// Unlock PF0 for SW

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;

// Configure pull-up for PF0 and PF4

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);

// Configure PWM clock

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);

// Adjust width and enable PWM

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)

{

// 0 degrees

ui8Adjust = 44;

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

ROM\_SysCtlDelay(40000000); // 3-second delay

// 165 degrees [Maximum angle specified by PowerHD 30001HB datasheet]

ui8Adjust = 121;

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

ROM\_SysCtlDelay(40000000); // 3-second delay

}

}

**------------------------------------------------------------------------------------**

**Task 02:**

Youtube Link: [**https://youtu.be/K-zCS-PIG7g**](https://youtu.be/K-zCS-PIG7g)

**Modified Code:**

**Using task 00 as a template, the code was modified to enable PF1’s PWM. Instead of PD0’s M1PWM0 and generator 0, PF1 will be using M1PWM5 and generator 2. For the PWM output control logic, instead of port 0, port 5 is used to output the PWM of PF1. Modify all PWM functions to fit these specifications. Finally, refine the delay function and increment/decrement ui16Adjust in the *if* structure for the LED to increase/decrease brightness with a 10-90% duty-cycle.**

**#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

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

{

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint16\_t ui16Adjust;

ui16Adjust = 1136; // Adjusts the brightness of the LED

// Sets up the system clock and PWM clock

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

// Enables PWM1, GPIOD, GPIOF

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

// Configure PF1 as a PWM

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

ROM\_GPIOPinConfigure(GPIO\_PF1\_M1PWM5);

// Unlock PF0 for SW

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;

// Configure pull-up for PF0 and PF4

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);

// Configure PWM clock

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

**PWMGenConfigure**(PWM1\_BASE, PWM\_GEN\_2, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(PWM1\_BASE, PWM\_GEN\_2, ui32Load);

// Adjust width and enable PWM

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, ui16Adjust);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_5\_BIT, **true**);

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_2);

**while**(1)

{

// Decrease DC

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_4)==0x00)

{

ui16Adjust = ui16Adjust - 5;

// Min DC = 10%

**if** (ui16Adjust < 1136)

{

ui16Adjust = 1136;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, ui16Adjust);

}

// Increase DC

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_0)==0x00)

{

ui16Adjust = ui16Adjust + 5;

// Max DC = 90%

**if** (ui16Adjust > 10225)

{

ui16Adjust = 10225;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, ui16Adjust);

}

ROM\_SysCtlDelay(10000);

}

}

**------------------------------------------------------------------------------------**

**Task 03:**

Youtube Link: <https://youtu.be/Qf9Zcm0QhTk>

**Modified Code:**

**The only piece of code that was carried from task 03 is the PWM. First modification is implementing an ADC that would read the value from the potentiometer. Next, interface the DC motor which required PA2 and PA3 to control the direction and stop the motor. Speed is controlled by the potentiometer.**

**#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"

**#include** "driverlib/adc.h"

**#define** PWM\_FREQUENCY 55

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

{

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32PotValue;

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint16\_t ui16Adjust = 0;

// Sets up the system clock and PWM clock

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

**SysCtlPWMClockSet**(SYSCTL\_PWMDIV\_64);

// Enable PE2 [ADC input], PWM0 [PE4 is the PWM], PA2, and PA3

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOE); // ADC input and PWM0

// PA2/PA3 controls direction of motor

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTA\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3);

**GPIOADCTriggerEnable**(GPIO\_PORTE\_BASE, GPIO\_PIN\_2); // PE2 is ADC input

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_PWM0); // Enable PWM0 for PE4

// Configure PE4 as a PWM

**GPIOPinTypePWM**(GPIO\_PORTE\_BASE, GPIO\_PIN\_4);

**GPIOPinConfigure**(GPIO\_PE4\_M0PWM4);

// Configure PWM clock

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

**PWMGenConfigure**(PWM0\_BASE, PWM\_GEN\_2, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(PWM0\_BASE, PWM\_GEN\_2, ui32Load);

// Adjust width and enable PWM

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, ui16Adjust);

**PWMOutputState**(PWM0\_BASE, PWM\_OUT\_4\_BIT, true);

**PWMGenEnable**(PWM0\_BASE, PWM\_GEN\_2);

// Enable ADC0 [Code from lab 5]

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 0, ADC\_CTL\_CH1);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 1, ADC\_CTL\_CH1);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 2, ADC\_CTL\_CH1);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 3, ADC\_CTL\_CH1 | ADC\_CTL\_IE | ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 1);

**while**(1)

{

**ADCIntClear**(ADC0\_BASE, 1);

**ADCProcessorTrigger**(ADC0\_BASE, 1);

// Poll for the ADC flag

**while**(!**ADCIntStatus**(ADC0\_BASE, 1, false));

**ADCSequenceDataGet**(ADC0\_BASE, 1, ui32ADC0Value);

// Average out the sampled analog signal

ui32PotValue = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

// Adjust the speed of the motor by multiplying the pot value by 3

ui16Adjust = ui32PotValue \* 3;

// Motor is off

**if**(ui16Adjust < 1136)

{

**GPIOPinWrite**(GPIO\_PORTA\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

ui16Adjust = 1136;

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, ui16Adjust);

}

// Motor is on

**else**

{

**GPIOPinWrite**(GPIO\_PORTA\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_PIN\_3);

// Adjust the speed from 10% DC to 90%

**if**(ui16Adjust < 10225)

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, ui16Adjust);

**else**

// Maximum DC of the PWM is 90%

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, 10225);

}

**SysCtlDelay**(10000);

}

}

**------------------------------------------------------------------------------------**

**Task 04:**

Youtube Link: <https://youtu.be/Ih0OGhEUm4M>

**Modified Code:**

**The primary change from task 03 code is implementing the provided QEI code and displaying the value to the terminal using UART.**

**#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"

**#include** "driverlib/adc.h"

**#include** "driverlib/qei.h"

**#include** "utils/uartstdio.h"

**#include** "driverlib/uart.h"

**#define** PWM\_FREQUENCY 55

**volatile** **int** qeiPosition;

**volatile** **int** qeiVelocity;

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

{

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32PotValue;

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint16\_t ui16Adjust = 0;

// Sets up the system clock and PWM clock

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

**SysCtlPWMClockSet**(SYSCTL\_PWMDIV\_64);

// Enable PE2 [ADC input], PWM0 [PE4 is the PWM], PA2, and PA3

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOE); // ADC input and PWM0

// PA2/PA3 controls direction of motor

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTA\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3);

**GPIOADCTriggerEnable**(GPIO\_PORTE\_BASE, GPIO\_PIN\_2); // PE2 is ADC input

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_PWM0); // Enable PWM0 for PE4

// Configure PE4 as a PWM

**GPIOPinTypePWM**(GPIO\_PORTE\_BASE, GPIO\_PIN\_4);

**GPIOPinConfigure**(GPIO\_PE4\_M0PWM4);

// Configure PWM clock

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

**PWMGenConfigure**(PWM0\_BASE, PWM\_GEN\_2, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(PWM0\_BASE, PWM\_GEN\_2, ui32Load);

// Adjust width and enable PWM

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, ui16Adjust);

**PWMOutputState**(PWM0\_BASE, PWM\_OUT\_4\_BIT, true);

**PWMGenEnable**(PWM0\_BASE, PWM\_GEN\_2);

// Enable ADC0 [Code from lab 5]

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 0, ADC\_CTL\_CH1);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 1, ADC\_CTL\_CH1);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 2, ADC\_CTL\_CH1);

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 3, ADC\_CTL\_CH1 | ADC\_CTL\_IE | ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 1);

// Enable QEI Peripherals

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOD);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_QEI0);

//Unlock GPIOD7 - Like PF0 its used for NMI -

HWREG(GPIO\_PORTD\_BASE + GPIO\_O\_LOCK) = GPIO\_LOCK\_KEY;

HWREG(GPIO\_PORTD\_BASE + GPIO\_O\_CR) |= 0x80;

HWREG(GPIO\_PORTD\_BASE + GPIO\_O\_LOCK) = 0;

//Set Pins to be PHA0 and PHB0

**GPIOPinConfigure**(GPIO\_PD6\_PHA0);

**GPIOPinConfigure**(GPIO\_PD7\_PHB0);

//Set GPIO pins for QEI. PhA0 -> PD6, PhB0 ->PD7.

**GPIOPinTypeQEI**(GPIO\_PORTD\_BASE, GPIO\_PIN\_6 | GPIO\_PIN\_7);

//DISable peripheral and int before configuration

**QEIDisable**(QEI0\_BASE);

**QEIIntDisable**(QEI0\_BASE,QEI\_INTERROR | QEI\_INTDIR | QEI\_INTTIMER |

QEI\_INTINDEX);

// Configure quadrature encoder, use an arbitrary top limit of 1000

**QEIConfigure**(QEI0\_BASE, (QEI\_CONFIG\_CAPTURE\_A\_B | QEI\_CONFIG\_NO\_RESET

| QEI\_CONFIG\_QUADRATURE | QEI\_CONFIG\_NO\_SWAP), 1000);

**QEIVelocityConfigure**(QEI0\_BASE, QEI\_VELDIV\_1, **SysCtlClockGet**());

// Enable the quadrature encoder.

**QEIEnable**(QEI0\_BASE);

//Set position to a middle value so we can see if things are working

**QEIPositionSet**(QEI0\_BASE, 500);

**QEIVelocityEnable**(QEI0\_BASE);

// Set up GPIOA for UART

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

// Enable UART0 so that we can configure the clock.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

// Use the internal 16MHz oscillator as the UART clock source.

**UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);

// Select the alternate (UART) function for these pins.

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Initialize the UART for console I/O.

UARTStdioConfig(0, 115200, 16000000);

**while**(1)

{

**ADCIntClear**(ADC0\_BASE, 1);

**ADCProcessorTrigger**(ADC0\_BASE, 1);

// Poll for the ADC flag

**while**(!**ADCIntStatus**(ADC0\_BASE, 1, false));

**ADCSequenceDataGet**(ADC0\_BASE, 1, ui32ADC0Value);

// Average out the sampled analog signal

ui32PotValue = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

// Adjust the speed of the motor by multiplying the pot value by 3

ui16Adjust = ui32PotValue \* 3;

// Motor is off

**if**(ui16Adjust < 1136)

{

**GPIOPinWrite**(GPIO\_PORTA\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

ui16Adjust = 1136;

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, ui16Adjust);

}

// Motor is on

**else**

{

**GPIOPinWrite**(GPIO\_PORTA\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_PIN\_3);

// Adjust the speed from 10% DC to 90%

**if**(ui16Adjust < 10225)

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, ui16Adjust);

**else**

// Maximum DC of the PWM is 90%

**PWMPulseWidthSet**(PWM0\_BASE, PWM\_OUT\_4, 10225);

}

// Read the position and speed of the motor using QEI

qeiPosition = **QEIPositionGet**(QEI0\_BASE);

qeiVelocity = **QEIVelocityGet**(QEI0\_BASE);

UARTprintf("QEI Velocity: %d\n", qeiVelocity);

**SysCtlDelay**(10000);

}

}