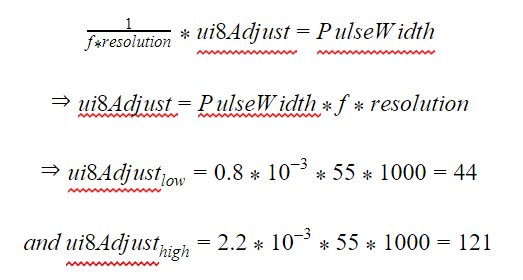
**Lab06:**

Task01:

Because this part of the lab requires for the motor to automatically make a full sweep of motion, all of the button functionalities that were in the original code for task 0 have been removed. Additionally, the Power HD 3001HB servo motor does not have a full 180 degree motion. Instead, the motor has 165 degree motion, which was used to complete this part of the lab. Datasheet information of this is found below:



So, the calculation was performed to find the ui8Adjust value to get the 800(0.8mS) and 2200(2.2mS) pulses required for full 165 degree motion.



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"

// Using a 55Hz control signal

#define PWM\_FREQUENCY 55

// Used to define the delay between pulse changes

#define delay 150000

int main(void)

{

volatile uint32\_t ui32Load;

volatile uint32\_t ui32PWMClock;

volatile uint8\_t ui8Adjust;

// Start with center position for 1.5mS pulse

ui8Adjust = 83;

// Clock at 40MHz

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

// PWM at 625kHz

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

// Enable PWM

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

// Use PWM0

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

// Use Pin1 as PWM output

ROM\_GPIOPinConfigure(GPIO\_PD0\_M1PWM0);

// Set to 55Hz

ui32PWMClock = SysCtlClockGet() / 64;

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

// Configure PWM as a down-counter

PWMGenConfigure(PWM1\_BASE, PWM\_GEN\_0, PWM\_GEN\_MODE\_DOWN);

// Load clock value

PWMGenPeriodSet(PWM1\_BASE, PWM\_GEN\_0, ui32Load);

// Set PWM resolution

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

// Set PWM as output

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_0\_BIT, true);

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_0);

// Simple delay between calibrating the motor and running the loop

ROM\_SysCtlDelay(100\*delay);

while(1)

{

// Count down the pulse width to 800uS

while(ui8Adjust>43)

{

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

ROM\_SysCtlDelay(delay);

ui8Adjust--;

}

// 44 is the min, so once the value updates to 43, go back to 44

if(ui8Adjust == 43)

ui8Adjust++;

ROM\_SysCtlDelay(20\*delay);

// Count up the pulse width to 2200uS

while(ui8Adjust<122)

{

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

ROM\_SysCtlDelay(delay);

ui8Adjust++;

}

// 121 is the max, so once the loop has been run and the value reaches 122, go back to 121

if(ui8Adjust == 122)

ui8Adjust--;

ROM\_SysCtlDelay(20\*delay);

}

}

Task02:

By applying PWM to the red LED and changing the pulse width of the PWM, I varied the duty cycle from 10% to 90%, changing the brightness of the LED. I used a variable called fDutyCycle to define the duty cycle of the LED, so all calculations involving fDutyCycle require using fDutyCycle/100 because the duty cycle is read as a percentage.

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"

// Using a 55Hz control signal

#define PWM\_FREQUENCY 55

// Resolution of duty cycle, defining how long the full LED cycle takes

#define resolution 0.0003

int main(void)

{

volatile uint32\_t ui32Load;

volatile uint32\_t ui32PWMClock;

volatile float fDutyCycle;

// Begin with 10% duty cycle

fDutyCycle = 10;

// Clock at 40MHz

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

// PWM at 625kHz

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

// Enable PWM and LEDs

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

// Use PWM0 and set red LED to output

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

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

// Use red LED as PWM output

ROM\_GPIOPinConfigure(GPIO\_PF1\_M1PWM5);

// Set to 55Hz

ui32PWMClock = SysCtlClockGet() / 64;

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

// Configure PWM as a down-counter

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

// Load clock value

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

// Set PWM resolution

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, fDutyCycle / 100 \* ui32Load);

// Set PWM as output

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

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

while(1)

{

// Reset LED to 10% duty cycle

fDutyCycle = 10;

// Count up to 90% duty cycle

while(fDutyCycle<90+resolution)

{

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, fDutyCycle \* ui32Load / 100);

fDutyCycle = fDutyCycle + resolution;

}

}

}

Task03:

By using nested for loops, the red LED will to a full cycle from 90% duty cycle to 10% duty cycle before the green LED updates its duty cycle. After the green LED goes through a full cycle, the blue LED will update its duty cycle. Because all LED are on together, the light produced mostly looks white, but the tone of the light can be seen to become less and less blue as blue’s duty cycle dims gradually.

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"

// Using a 55Hz control signal

#define PWM\_FREQUENCY 55

// Resolution of duty cycle, defining how long the full LED cycle takes

#define resolution 0.5

int main(void)

{

volatile uint32\_t ui32Load;

volatile uint32\_t ui32PWMClock;

volatile float fDutyCycleR;

volatile float fDutyCycleG;

volatile float fDutyCycleB;

// Begin with 90% duty cycle

fDutyCycleR = 90;

fDutyCycleG = 90;

fDutyCycleB = 90;

// Clock at 40MHz

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

// PWM at 625kHz

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

// Enable PWM and LEDs

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

// Use PWM0 and set LEDs to PWM pins

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

ROM\_GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3);

// Configure red, green, and blue LEDs to PWM pins

ROM\_GPIOPinConfigure(GPIO\_PF1\_M1PWM5);

ROM\_GPIOPinConfigure(GPIO\_PF2\_M1PWM6);

ROM\_GPIOPinConfigure(GPIO\_PF3\_M1PWM7);

// Set to 55Hz

ui32PWMClock = SysCtlClockGet() / 64;

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

// Configure PWM as a down-counter. GEN\_2 handles red LED, and GEN\_3 handles the blue and green

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

PWMGenConfigure(PWM1\_BASE, PWM\_GEN\_3, PWM\_GEN\_MODE\_DOWN);

// Load clock value

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

PWMGenPeriodSet(PWM1\_BASE, PWM\_GEN\_3, ui32Load);

// Set PWM resolution

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, fDutyCycleR / 100 \* ui32Load);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_6, fDutyCycleB / 100 \* ui32Load);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_7, fDutyCycleG / 100 \* ui32Load);

// Set PWM as output and begin cycle

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

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_6\_BIT, true);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_7\_BIT, true);

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

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_3);

while(1)

{

// Cycle through each LED fading

for (fDutyCycleR = 90; fDutyCycleR > 9; fDutyCycleR-=resolution)

for (fDutyCycleG = 90; fDutyCycleG > 9; fDutyCycleG-=resolution)

for (fDutyCycleB = 90; fDutyCycleB > 9; fDutyCycleB-=resolution)

{

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, fDutyCycleR / 100 \* ui32Load);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_6, fDutyCycleG / 100 \* ui32Load);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_7, fDutyCycleB / 100 \* ui32Load);

ROM\_SysCtlDelay(100000);

}

}

}