**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);

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

\*/

#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.

\* 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

\* 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);

}

}

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

**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);

}

}