**Lab04:**

Task01:

To get a 75% duty cycle with 2Hz clock, I switched between the clock being 1.5Hz (75% of 2Hz) when the LED is on, and 0.5Hz (25% of 2Hz) when the LED is off. Also, to graph the resulting waveform, the value of the LED was fed to a variable that was then graphed.

Code:

#include <stdint.h>

#include <stdbool.h>

#include "inc/tm4c123gh6pm.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/sysctl.h"

#include "driverlib/interrupt.h"

#include "driverlib/gpio.h"

#include "driverlib/timer.h"

int main(void)

{

// Use to define the period of LED blinking

uint32\_t ui32Period;

// Used to generate waveform graph of LED

uint32\_t BlueLED;

// Set clock to 40MHz SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_XTAL\_16MHZ|SYSCTL\_OSC\_MAIN;

// Enable LED's as outputs

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

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

// Enable timer0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

// Clock period of 2Hz with 75% duty cycle

ui32Period = (SysCtlClockGet() / 2) \* 3/4;

TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period -1);

// Enable timer0 overload as interrupts

IntEnable(INT\_TIMER0A);

TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

IntMasterEnable();

// Start timer

TimerEnable(TIMER0\_BASE, TIMER\_A);

while(1)

{

// Store the value of the blue LED

BlueLED = GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

}

}

// When timer0 overloads, cycle the LED

void Timer0IntHandler(void)

{

// Redefined for use in interrupt handler

uint32\_t ui32Period;

// Clear the timer interrupt

TimerIntClear(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

// Read the current state of the GPIO pin and

// write back the opposite state

if(GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_2))

{

// Turn off LEDs

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 0);

// Set timer0 to have 25% duty cycle while LED is off

ui32Period = (SysCtlClockGet() / 2)\* 1/4;

TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period -1);

}

else

{

// Turn on blue LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4);

// Set timer0 to have 75% duty cycle while LED is on

ui32Period = (SysCtlClockGet() / 2)\* 3/4;

TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period -1);

}

}

Task02:

To add the functionality of holding the LED on for 1.5sec when the button was pressed, the code for Task 1 was used to create the regular 75% duty cycle with 2Hz. When the pushbutton is pressed, an interrupt fires that turns timer0’s interrupt off and turns timer1 on, as well as turns on the LED. Timer1 is set to count for 1.5sec, then an interrupt fires when timer1 overloads which turns off timer1 and re-enables timer0’s overload interrupt to instruct for the TIVAC to go back to the normal 2Hz blinking. These two additional interrupts were also added to the .ccs file for task 2, which is why there are two .ccs files shown with the source code: one for task 1, and one for task 2.

Code:

#include <stdint.h>

#include <stdbool.h>

#include "inc/tm4c123gh6pm.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/sysctl.h"

#include "driverlib/interrupt.h"

#include "driverlib/gpio.h"

#include "driverlib/timer.h"

#include "inc/hw\_gpio.h"

int main(void)

{

// Use to define the period of LED blinking

uint32\_t ui32Period;

// Used to generate waveform graph of LED

uint32\_t BlueLED;

// Set clock to 40MHz

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

// Enable LED's as outputs

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

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

// Unlock GPIOCR for register F

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

// Free up sw2

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_CR) = 0x01;

// Set sw2 to be an input

GPIOPinTypeGPIOInput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

// Enable sw2 as an interrupt

GPIOIntEnable(GPIO\_PORTF\_BASE, GPIO\_INT\_PIN\_0);

GPIOIntTypeSet(GPIO\_PORTF\_BASE, GPIO\_INT\_PIN\_0, GPIO\_RISING\_EDGE);

IntEnable(INT\_GPIOF);

// Enable timer0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

// Clock period of 2Hz with 75% duty cycle

ui32Period = (SysCtlClockGet() / 2) \* 3/4;

TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period -1);

// Enable timer0 overload as interrupts

IntEnable(INT\_TIMER0A);

TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

IntMasterEnable();

// Start timer

TimerEnable(TIMER0\_BASE, TIMER\_A);

while(1)

{

// Store the value of the blue LED

BlueLED = GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

}

}

// When timer0 overloads, cycle the LED

void Timer0IntHandler(void)

{

// Redefined for use in interrupt handler

uint32\_t ui32Period;

// Clear the timer interrupt

TimerIntClear(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

// Read the current state of the GPIO pin and

// write back the opposite state

if(GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_2))

{

// Turn off LEDs

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 0);

// Set timer0 to have 25% duty cycle while LED is off

ui32Period = (SysCtlClockGet() / 2)\* 1/4;

TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period -1);

}

else

{

// Turn on blue LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4);

// Set timer0 to have 75% duty cycle while LED is on

ui32Period = (SysCtlClockGet() / 2)\* 3/4;

TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period -1);

}

}

// When sw2 is pushed, hold for 1.5 sec

void PortFIntHandler(void)

{

// Disable timer0 interrupt, but keep timer0 running for later use

TimerIntDisable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

// Clear sw2 interrupt

GPIOIntClear(GPIO\_PORTF\_BASE, GPIO\_INT\_PIN\_0);

// If sw2 has been pressed, turn on the blue LED

if (GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_0))

{ // Turn on blue LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2,4);

// Enable timer1

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

TimerConfigure(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

// Setup timer1 as 2/3Hz (3/2sec)

uint32\_t ui32Period = (SysCtlClockGet()/2\*3);

TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period -1);

// Enable overload of timer1 as interrupt

IntEnable(INT\_TIMER1A);

TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

IntMasterEnable();

// Start timer1

TimerEnable(TIMER1\_BASE, TIMER\_A);

}

}

// When timer1 overloads, go back to oscillating with timer0

void Timer1IntHandler(void)

{

// Clear the timer interrupt

TimerIntClear(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

// Turn off timer1 and disable its interrupt

TimerDisable(TIMER1\_BASE, TIMER\_A);

IntDisable(INT\_TIMER1A);

// Turn off LED to set to beginning of blinking cycle

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 0);

// Re-enable timer0 overload interrupt to restart blinking cycle

TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

}