**Date Submitted: September 27, 2019**

**Task 00: Execute provided code**

**Youtube Link: https://youtu.be/JkNfsvaS3hg**

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

**Task 01:**

Youtube Link: https://youtu.be/vN7L4JZnA88

**Modified Schematic (if applicable):**

**Modified Code:**

uint32\_t ui32PeriodHigh;

uint32\_t ui32PeriodLow;

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

{

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

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

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

ui32PeriodHigh = (**SysCtlClockGet**() / 10) \* 0.43; // = 40MHz -> To get 43% duty cycle multiply by 0.43

ui32PeriodLow = (**SysCtlClockGet**() / 10) \* 0.57; // = 40MHz -> To get 43% duty cycle multiply by 0.43

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32PeriodHigh -1);

**IntEnable**(INT\_TIMER0A);

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

**IntMasterEnable**();

**TimerEnable**(TIMER0\_BASE, TIMER\_A);

**while**(1)

{

}

}

**void** **Timer0IntHandler**(**void**)

{

// 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))

{

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

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32PeriodLow -1);

}

**else**

{

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

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32PeriodHigh -1);

}

}

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

**Task 02:**

Youtube Link: https://youtu.be/PjFIY-6eInA

**Modified Schematic (if applicable):**

**Modified Code:**

**#include** "inc/hw\_gpio.h"

**#define** LED\_PERIPH SYSCTL\_PERIPH\_GPIOF

**#define** LED\_BASE GPIO\_PORTF\_BASE

**#define** RED\_LED GPIO\_PIN\_1

**#define** BLUE\_LED GPIO\_PIN\_2

**#define** GREEN\_LED GPIO\_PIN\_3

**#define** Button\_PERIPH SYSCTL\_PERIPH\_GPIOF

**#define** ButtonBase GPIO\_PORTF\_BASE

**#define** Button GPIO\_PIN\_0

**#define** ButtonInt GPIO\_INT\_PIN\_0

uint32\_t ui32PeriodHigh;

uint32\_t ui32PeriodLow;

**void** **timer1A\_delayMs**(**int** ttime);

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

{

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

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

**SysCtlDelay**(3);

//Switch Config

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;

**GPIOPinTypeGPIOInput**(GPIO\_PORTF\_BASE, Button);

**GPIOPadConfigSet**(GPIO\_PORTF\_BASE,Button,GPIO\_STRENGTH\_2MA,GPIO\_PIN\_TYPE\_STD\_WPU);

//LED Config

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

//GPIO INT Config

**GPIOIntEnable**(GPIO\_PORTF\_BASE, Button);

**GPIOIntTypeSet**(GPIO\_PORTF\_BASE, Button, GPIO\_FALLING\_EDGE);

**IntEnable**(INT\_GPIOF);

//Timer Config

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

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

ui32PeriodHigh = (**SysCtlClockGet**() / 10) \* 0.43; // = 40MHz -> To get 43% duty cycle multiply by 0.43

ui32PeriodLow = (**SysCtlClockGet**() / 10) \* 0.57; // = 40MHz -> To get 43% duty cycle multiply by 0.43

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32PeriodHigh -1);

//Timer INT config

**IntEnable**(INT\_TIMER0A);

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

**IntMasterEnable**();

**TimerEnable**(TIMER0\_BASE, TIMER\_A);

**while**(1)

{

}

}

**void** **Timer0IntHandler**(**void**)

{

// 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))

{

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

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32PeriodLow -1);

}

**else**

{

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

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32PeriodHigh -1);

}

}

**void** **PortFIntHandler**(**void**)

{

//Clear the GPIO interrupt

**GPIOIntClear**(GPIO\_PORTF\_BASE, ButtonInt);

//Read the current state of the GPIO pin and write back the opposite state

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GREEN\_LED, GREEN\_LED);

timer1A\_delayMs(1000);

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

**SysCtlDelay**(2000000); //delay of .05ms to resolve debouncing. 2M was chosen as 40MHz \* .05 = 2M

}

**void** **timer1A\_delayMs**(**int** ttime)

{

**int** i;

SYSCTL\_RCGCTIMER\_R |= 2; //enable clock to Timer Block 1

TIMER1\_CTL\_R = 0; //disable Timer before initialization

TIMER1\_CFG\_R = 0x04; //16-bit option

TIMER1\_TAMR\_R = 0x02; //periodic mode and down-counter

TIMER1\_TAILR\_R = 40000 - 1; //TimerA interval load value reg

TIMER1\_ICR\_R = 0x1; //clear the Timer A timeout flag

TIMER1\_CTL\_R |= 0x01; //enable Timer A after initialization

**for**(i = 0; i < ttime; i++) {

**while**((TIMER1\_RIS\_R & 0x1) == 0)

; //wait for TimerA timeout flag

TIMER1\_ICR\_R = 0x1; //clear TimerA timeout flag

}

}

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