MANHATTAN COLLEGE

Lab 6: Interrupt Programming with GPIO

School of Engineering

Electrical and Computer Engineering Department

ECEG 721-61 Embedded Systems

Jamie Quinn & Lucia Rosado-Fournier

November 25th, 2020

Table of Contents

1.1 Component Requirements

1.1.1 Software Requirements

1. Keil uVision4
2. TivaWare_C_Series

1.1.2 Hardware Requirements

1. Tiva LaunchPad EK-TM4C123GXL LaunchPad
2. USB Cable

1.2 Software

The software is written in the C language using the Keil uVision4 Integrated Development Environment (IDE).

The software enables the clock and GPIO Port F of the microcontroller to configure pin 3. This is done because it is connected to the green LED and it will be used as an output. The timer is then configured, and the timer interrupt is enabled. When on the interrupt, the interrupt service routine (ISR) reads pin 3. Based on the value, HIGH or LOW, the opposite is written. This toggles the LED based on the timer interrupt. The frequency for this is stored in a temporary variable, ui32Period.

Using this variable, the timer will count for every cycle based on the systems clock frequency. In this program, the GPIO is at 10HZ and 50% duty cycle. The number of counts needed is calculated as follows.

$$Number\ of\ Clock\ Cycles = \frac{System\ Clock\ Frequency}{Desired\ Frequency}$$

$$ui23Period = Number\ of\ Clock\ Cycles * Duty\ Cycle$$

$$= \frac{40\ MHZ}{10\ HZ} * \frac{50}{100}$$

$$= 2 * 10^6\ counts$$

Flowchart

The flowchart for the C code is shown in the figure below.

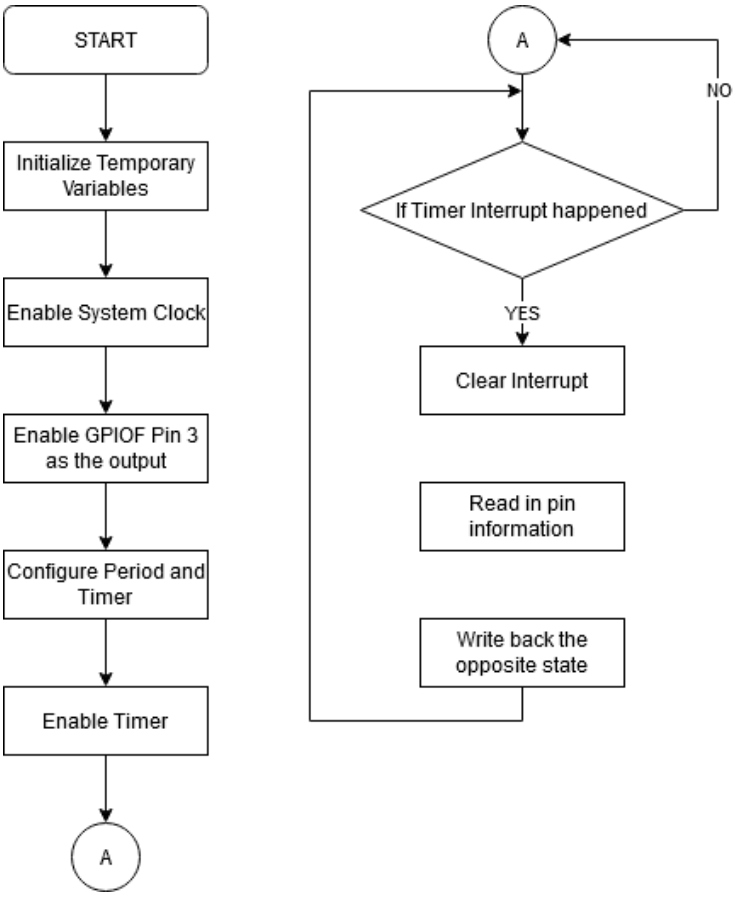

Figure *– Flowchart of the C code to enable the interrupt.*

C code

The C code can be seen below.

```
#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"
// includes all necessary library and drivers to execute this program
int main(void) {
	uint32_t ui32Period;
	SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN); //enabling system clock
	//Enable GPIO F
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|
GPIO_PIN_3); //enabling GPIO Pin 3 as Output
	//Enable Timer
	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
	//Configure Timer
	TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
	ui32Period = (SysCtlClockGet() / 10) / .5; //configuring the period
	TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
	IntEnable(INT_TIMER0A);
	TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
	IntMasterEnable();
	TimerEnable(TIMER0_BASE, TIMER_A);
	while(1) { }
}
void TIMER0A_Handler(void) //Interrupt Service Routine
{ // 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_3))
	{
		GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3,
0); //writing 0 to GPIO PF3
	}
	else {
		GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8); //writing 8 in
hex to the GPIO PF3
	}
}
```

1.3 Procedure

The procedure to run the software is as follows.

1. Connect the microcontroller to the PC using the USB cable.
2. Build the program and Load onto the microcontroller.
3. Observe the results.

1.4 Observation

As expected, the LED blinks green at a specific rate, which can be changed by changing the temporary variable specified earlier. When the reset switch is held, the LED stops flashing. There were no problems when working with the lab.

1.5 Summary

This lab was a good introduction to interrupts and how they can be used to make the hardware do different functionalities when the interrupt is triggered. This can be seen in the image below. Overall, the lab took about two hours to complete.

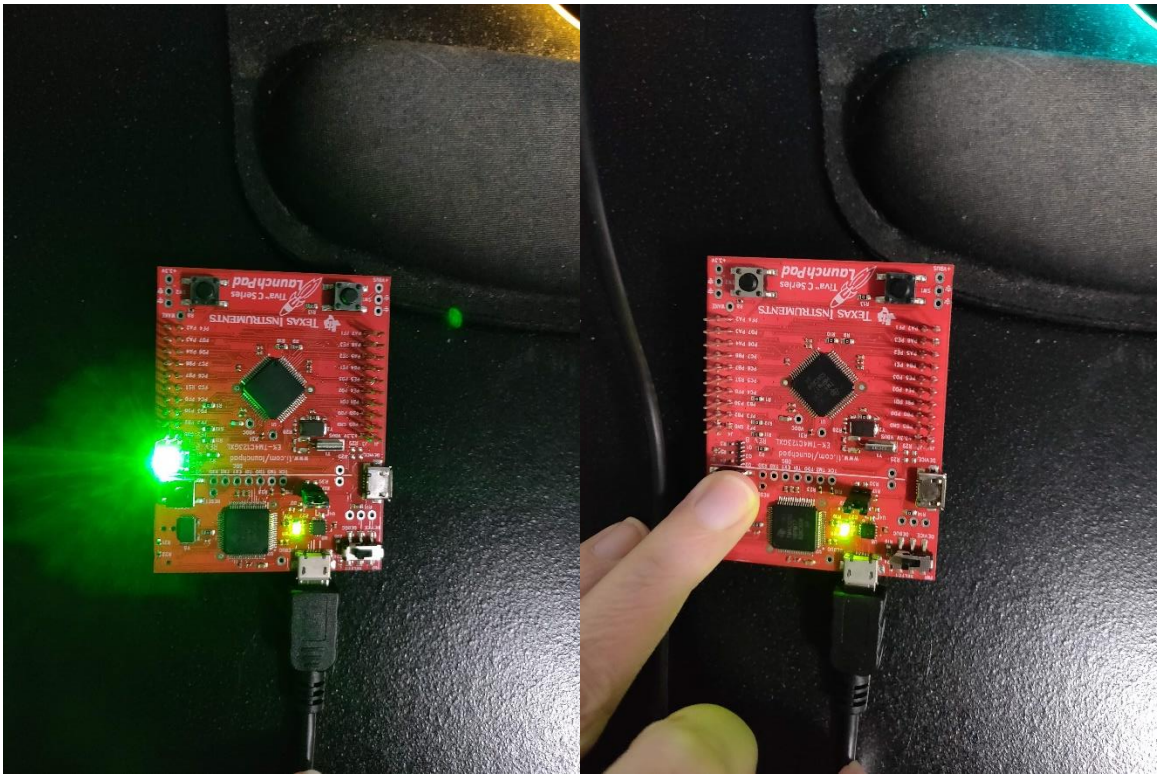

Figure *– Microcontroller blinking and not blinking when the reset is pressed.*