香港中文大學
The Chinese University of Hong Kong

CENG2400 Embedded System Design

Lab 03: Interrupt and Timer

Kezhi LI

kzli24@cse.cuhk.edu.hk

Outline

- **Step 1:** Learning the interrupt
- **Step 2:** Learning the calculation of delay
- **Step 3:** Doing your assignment (upload your code and video on blackboard before next lab)

Step 1: Learning the interrupt

What is an Interrupt?

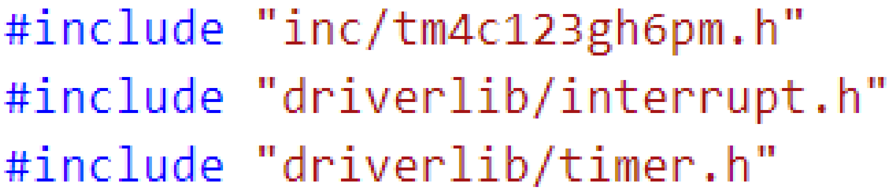
- Event that pauses the normal program execution and run an interrupt service routine (ISR).
- Types: hardware interrupts (e.g., switch press) and software interrupts (e.g., timer)

Why use Interrupts?

- After lab2, although you can change the mode of LED after each RGB color ends, the response is not efficient enough.
- Interrupt can help you change the mode immediately after pressing the button.

Step 1: Learning the interrupt

1. Include 3 extra header files in your “main.c” file.



```
#include "inc/tm4c123gh6pm.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
```

- **tm4c123gh6pm.h**: Definitions for the interrupt and register assignments on the Tiva C Series device on the LaunchPad board

- **interrupt.h**: Defines macros for NVIC Controller(Interrupt) API of *driverLib*. This includes API functions such as *IntEnable* and *IntPrioritySet*.

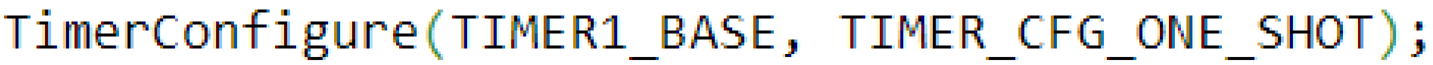
- **timer.h**: Defines macros for Timer API of *driverLib*. This includes APIfunctions such as *TimerConfiqure* and *TimerLoadSet*.

Step 1: Learning the interrupt

For Timer Interrupt

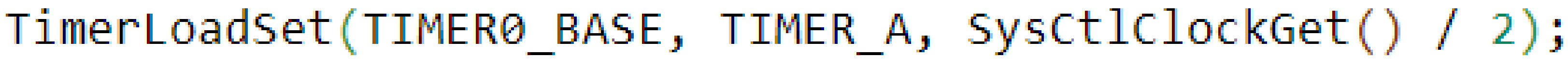
2. Timer Configuration

One shot or ***Periodic***



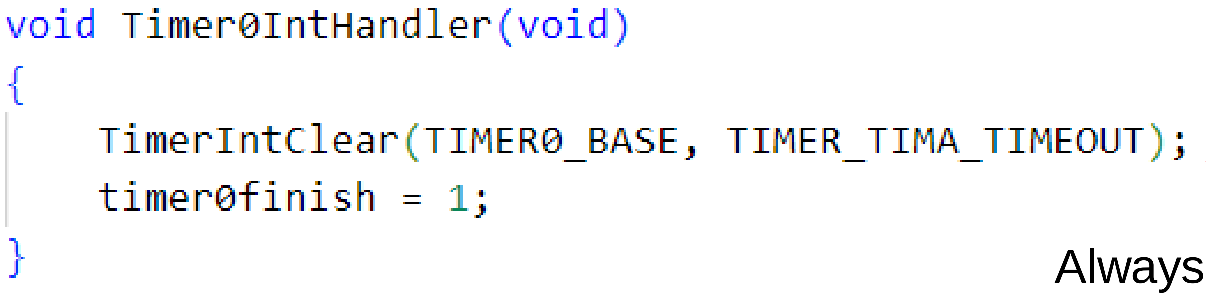
```
TimerConfigure(TIMER1_BASE, TIMER_CFG_ONE_SHOT);
```

3. Set Timer Delay



```
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 2);
```

4. Define Timer Interrupt Handler

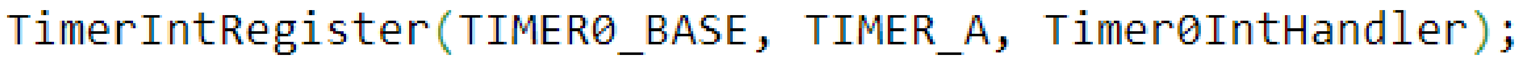


```
void Timer0IntHandler(void)
{
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    timer0finish = 1;
}
```

Always remember to clear the interrupt at the beginning of the handler.

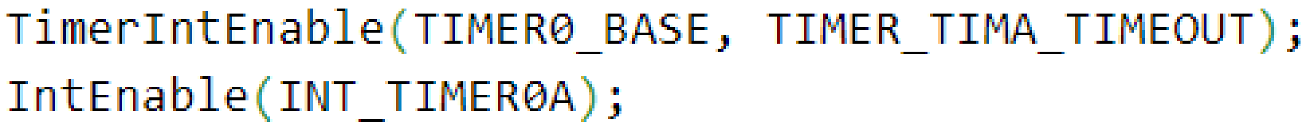
Step 1: Learning the interrupt

5. Register Timer Handler



```
TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0IntHandler);
```

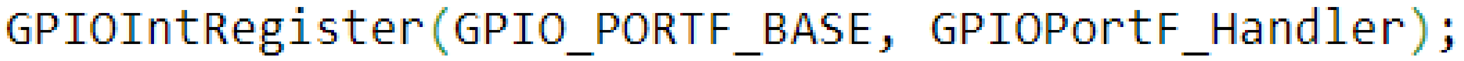
6. Enable Interrupt and Timer



```
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
IntEnable(INT_TIMER0A);
```

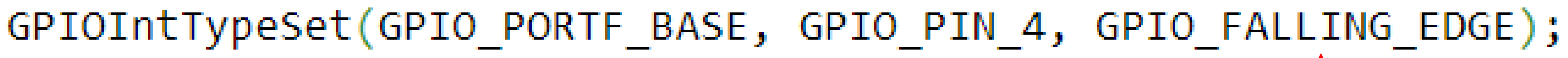
For GPIO Interrupt

2. Register GPIO Handler



```
GPIOIntRegister(GPIO_PORTF_BASE, GPIOPortF_Handler);
```

3. Set GPIO Interrupt Type

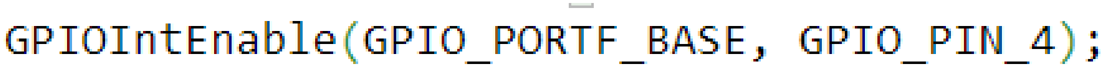


```
GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_FALLING_EDGE);
```

Falling Edge or ***Rising Edge*** or ***Both***

Step 1: Learning the interrupt

4. Enable GPIO Interrupt

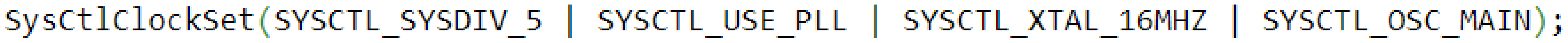


```
GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);
```

Step 2: Learning the delay

You should be aware of the time used in each setup

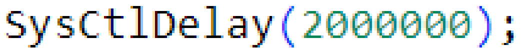
1. Define the clock using this function



```
SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);
```

The default frequency is 400M *Hz*, and if we set “SYSCTL_SYSDIV_5”, the frequency is divided by 5, which is 80M *Hz* instead.

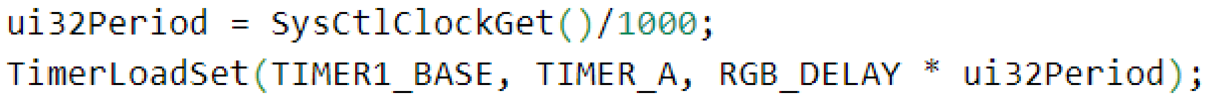
2. In Lab2, we set up the delay using



```
SysCtlDelay(2000000);
```

Since the frequency 80M *Hz*, and we set the delay to be 2M, the delay should be 2 / 80 * 3 = 0.075s. Here we times 3 because the each iteration takes 3 clock cycles by default.

3. In Lab3, we set up the delay using



```
ui32Period = SysCtlClockGet()/1000;
TimerLoadSet(TIMER1_BASE, TIMER_A, RGB_DELAY * ui32Period);
```

The unit of the clock is *Hz*, there ***ui32Period*** defines the frequency per millisecond. Then, the timer load is set to be ***RGB_DELAY * ui32Period***, so that the delay of the timer is exactly ***RGB_DELAY*** ms.

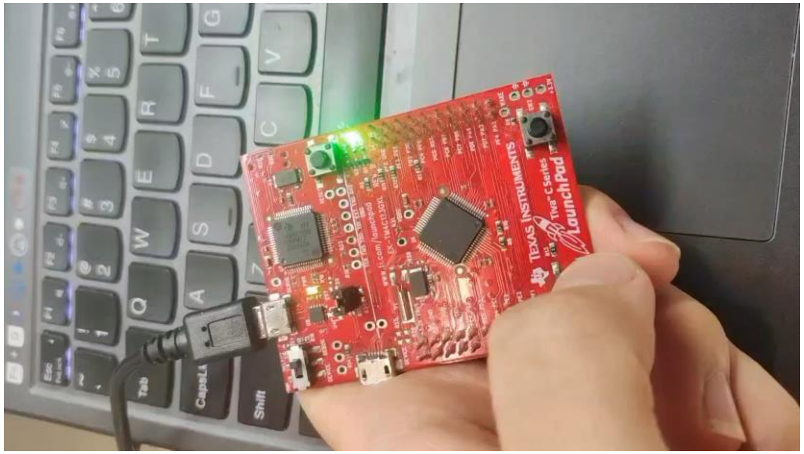
Step 3: Assignments

Before doing the assignments, please carefully reading the provided code.

Assignment 1: Mimic the ***RGB_Timer()*** in the provided code, finish ***Flash_Timer()***.

Assignment 2: Replace ***Read_Switches_Timer()*** with the GPIO interrupt.

Your final demo should look like this:



Q&A

Thanks for listening!

Q & A