EE186 Lab 2:
Sorry to Interrupt!

Due: October 16, 2026 (10:00 PM)

For detailed guidance, please watch the **Getting Started Video** on Canvas under **Panopto Course Videos** tab.

1 Introduction

In this lab, you will explore one of the most fundamental concepts in embedded systems: **interrupts**. Unlike polling, where a program continuously checks for changes, interrupts allow the microcontroller to respond immediately to events. In this lab, you will implement two types of interrupts:

- External interrupts: triggered by an external event, such as pressing a button.
- Timer interrupts: triggered at a specific time interval to execute periodic tasks.

As you did in the last lab, please find any relevant information in the datasheet PDFs here: board manual, MCU reference manual, MCU datasheet.

Note: In this lab, you are not allowed to use any Hardware Abstraction Layer (HAL) provided by ST. In other words, after creating a project folder, delete everything in `main.c` and start from:

```
#include <stdint.h>
int main(void)
{
    // your code goes here
    while(1)
    {
        // your code goes here
    };
}
```

2 Push Button

An **external interrupt** allows the microcontroller to react to changes in input signals from external components. When this event occurs, an interrupt triggers a predefined function called the Interrupt Service Routine (ISR).
In this part of the lab, you will configure an external interrupt on a push button press. When the button is pressed, the microcontroller should detect the event and toggle a blue LED.

The extended interrupts and events controller (EXTI) is used to map various interrupt sources such as a GPIO pin to specific interrupts and set the interrupt conditions such as rising or falling edge. There are 39 possible external interrupts that can be generated by GPIO and other peripherals. Table 78 in the Reference Manual lists all these sources. The first 16 EXTI (0-15) are used for GPIO peripherals. Each external

interrupt can be associated with one GPIO pin from banks PA to PI. The pin number and external interrupt number will be the same. For example, EXTI0 can be associated with any GPIO pin PA0 through PI0.

Explain how external interrupts work and how the ISR handles button presses. What happens to the main loop? When does the ISR get executed? What would happen if you did not clear the EXTI pending flag inside the ISR?

Now, implement the push button interrupt such that when the board's push button is pressed, the blue LED is toggled. That is, if the LED is on, the push button press turns the LED off. If the LED is off, the button press turns the LED on. Similar to Lab 1, you are required to find relevant register addresses and manipulate the bits. You may find these sections below in the STM32L4xx Reference Manual useful.

- Section 6: Reset and clock control (RCC).
- Section 8: General-purpose I/Os (GPIO).
- Section 9: System configuration controller (SYSCFG).
- Section 15: Nested vectored interrupt controller (NVIC). See the ARMv7-M Architecture Reference Manual Section B 3.4 for register specific details.
- Section 16: Extended interrupts and events controller (EXTI).

Deliverables

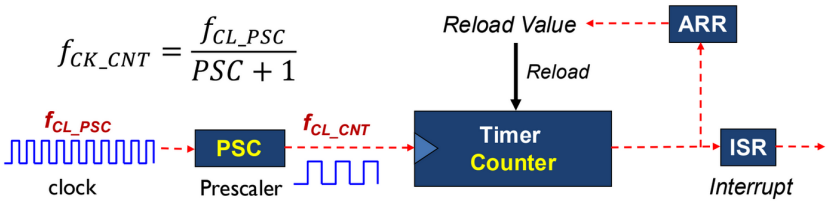
- (3 Points) Explain how external interrupts work.
- (5 Points) Submit a short video demonstrating the push button toggling the blue LED along with your source code.

3 Timer

In Lab 1, you implemented blinking LEDs using a software delay inside a loop. This approach works but is inefficient since it blocks the CPU from performing other tasks while waiting. A better approach is to use a **timer interrupt**, which allows the microcontroller to change the LED color at precise time intervals without blocking execution. In this section, you will use a timer interrupt to cycle through the LED sequence green → blue → red one second per color.

Explain how a timer interrupt works. What function is counting time passed? What function triggers the ISR? How is the timer interrupt different from an external interrupt?

The microcontroller does not inherently understand time intervals like 1 second; instead, it operates based on its clock frequency. The internal timer counts clock cycles, and for it to generate an interrupt every second, we need to determine how many clock cycles correspond to that time interval. Calculate the timer counter frequency using what you've learned in class.



Using the calculated counter frequency, enable timer 4. To do this, you need to enable the corresponding RCC, configure TIM4 (prescaler and auto-reload register), and set TIM4 as an interrupt.

Deliverables

- (3 Points) Explain how timer interrupt works.
- (2 Points) Calculate the timer frequency. Show the process.
- (5 Points) Submit a short video demonstrating the timer interrupt along with the source code.

4 Sorry to Interrupt!

Now that you've learned both external interrupt and timer interrupt, let's put them to work together. Build a system where the push button temporarily stops the LED sequence. Normally, the LEDs should cycle green $\rightarrow$ blue $\rightarrow$ red every second using a timer interrupt. However, when the button is pressed, the currently lit LED should remain on until the button is released. The timer should keep running but should not change the LED while the button is pressed. In order to implement this, you need to use both the rising and falling edges of the signal.

Deliverables

(5 Points) Submit a video demonstrating this interrupt behavior along with the source code.