EE 14 Lab 6: Interrupts

Lab report due a week after your lab session (3-7 March 2025)

1 Introduction

So far, we've used the microcontroller to do a variety of input/output tasks, but we had no good way to keep track of time in code. The only way to delay has been to spin in a loop for many cycles, which makes it hard for the processor to do anything else in the meantime.

To build more complex systems, it will be important for the processor to respond quickly to events, without having to watch them constantly. In this lab you will implement *interrupts*, which give us tools to track time and to respond quickly to external events.

After successfully completing this lab, you should be able to:

- Use the `SysTick` timer and interrupt to keep track of time
- Use a GPIO pin to trigger an interrupt from an external source
- Drive a multiplexed display (specifically a dual seven-segment display)

Documentation focus: Writing for a non-technical audience. See the guidelines at the end of this handout.

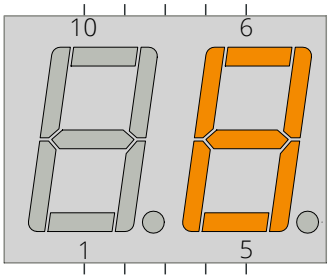
2 Prelab

If you've taken ES 4, this should be very *familiar. You're welcome to copy your previous work.*

P1: Look at the datasheet for the LTD-4608JR 7-segment display contained in your lab kit (posted on the course website), and determine which segments (A-G) should be on for each decimal digit 0-9. It will help to write this out like a truth table:

digit	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
...	...						

P2: Label the pins on the diagram below according to which segment they control (A-G). Pin 1 is the bottom left, and they are numbered sequentially counter-clockwise.



3 In lab

SysTick

L1: Create an empty PlatformIO project with the same settings as usual:

- **Board**: ST Nucleo L432KC
- **Framework**: CMSIS

L2: Download the framework starter files from the course website (`gpio.c`, `ee14lib.h`, and `main.c`) and add them to your project.

The SysTick timer is a "core peripheral" of the CPU (i.e., it's part of all Cortex-M4 processors, and not just our particular STM32 chip), so you'll find details about it in the **Cortex-M4 Programming Manual** (also linked on the course website). Section 4.5 describes what each register does.

L3: The following code (in `main.c`) initializes the SysTick timer.

Look at the register map for the SysTick peripheral and annotate each line of code with a brief comment explaining what it does.

```
void SysTick_initialize(void) {
    SysTick->CTRL = 0;
    SysTick->LOAD = 0; // TODO: your value here
    NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) - 1);
    SysTick->VAL = 0;
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
}
```

L4: Replace the 0 in `SysTick->LOAD = 0` with an appropriate value so that SysTick will generate an interrupt once per millisecond (i.e., at 1kHz). The input clock is the same 4 MHz clock we know and love from previous labs.

L5: Create a variable to hold a count of the number of times the SysTick interrupt has triggered (which will also be the number of milliseconds elapsed since turning on). Remember to declare it `volatile`!

Write the SysTick handler function. It should have the signature `void SysTick_Handler(void)`, and it should increment the counter variable each time it runs.

L6: Write a simple `delay_ms()` function which takes a number of milliseconds to delay. It should record the current millisecond count, add the delay, and then spin in a loop until the count exceeds that number.

> **L7**: Write some code to test your `delay_ms()` function, perhaps by turning an LED on and off at a given interval.

Multiplexed displays

> **L8**: Write code to drive the two digits of your display.

Since the two digits share the cathode lines, you cannot control them independently — instead, turn on one anode with the corresponding cathodes for one digit, and then switch to the other digit. If you alternate faster than about 50 Hz, your eyes will not perceive the flicker and it will apear that both digits are on.[1]

The main loop should look something like this:

```
while(1) {
  // Set up segments (cathodes) for tens digit
  // Turn on tens digit anode
  // Wait for 5 ms or so
  // Turn off the tens digit anode

  // Repeat the process for the ones digit!
}
```

You probably want to write a reusable function to set the cathode GPIO pins to display particular digit. If you store your segments truth table (from Prelab 1) in an array, your code to do this can be very simple.

> **L9**: Display the timer value on your seven-segment display.

External interrupts

For the final part, let's wire up an external button to trigger a second interrupt.

> **L10**: Pick a GPIO pin, and figure out which port/pin number it has. Connect a switch between that pin and ground.

> **L11**: Write code to set the pin as an input and enable the pullup resistor. You might find it helpful to write some code to display the value of the pin or something to confirm it's working.

> **L12**: Read section 9.2.3-9.2.6 of the STM32L4xxx reference manual (on EXTI interrupt mapping) to figure out which bits need to be set in order to route your particular GPIO pin to the EXTI (external interrupt controller).

> **L13**: Read section 13 of the manual to determine which bits of the EXTI FTSR and IMR registers need to be set in order to enable the interrupt. An example function is shown below; you just need to replace the elipsis with the specific bits.

```
void config_gpio_interrupt(void)
{
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[...] ...
```

[1] This isn't just a clever trick to do in lab; nearly all LED displays actually work this way.

```
    EXTI->FTSR1 |= ...
    EXTI->IMR1 |= ...

    NVIC_SetPriority(..., 2); // (IRQ number, priority)
    NVIC_EnableIRQ(...);
}
```

L14: Write your interrupt handler, and make it do something interesting. One option is to set up a counter which tracks the number of button presses (times the interrupt handler has run) and display this. You could also continue to display the SysTick timer, and use the button to pause or reset it.

Your interrupt handler should be called `EXTI*_IRQHandler` (where * is replaced by the number). Interrupts 5-9 and interrupts 10-15 are grouped together into one EXTI channel; their handlers should be named `EXTI9_5_IRQHandler` and `EXTI15_10_IRQHandler` respectively.

Note that you may need to check the pending interrupt register (PR1) to see which event actually occured.

You will also need to reset the appropriate bit of PR1 somewhere in your interrupt handler. Otherwise the processor will keep running your interrupt handler over and over. Note that you can reset it by writing a 1, so this can be a simple assignment statement instead of a `|=` (resulting in 1 CPU instruction instead of 3).

```
    EXTI->PR1 = ...
```

4 Report for this lab

If you cannot explain something in simple terms, you don't understand it well enough.

— attributed to Richard Feynman

For this week's "report", you will write a 2-page explanation of this lab for a non-technical audience (i.e., someone who is not an engineer and does not know how to code).

This has two purposes. First, engineers often need to communicate complex ideas to non-technical people, including managers, marketing, customers, policy makers, and the general public. Doing so in a way that is clear, friendly, and accessible is a huge boost to your career, and elevates the public perception of engineering as a whole.

Second, as the quote above says, the process of trying to explain something precisely and without technical jargon often forces you to clarify your own understanding. I cannot begin to count the number of times I've tried to explain something in a class, only to realize that I had major gaps in my knowledge of basic concepts!

Although this sort of writing can be challenging, it is a skill you can develop with practice, even to the point of enjoying it.

Report guidelines

Your reader is an intelligent, thoughtful person who is not an engineer. Think of a family member or a friend majoring in something outside of STEM. Your "report" should be a 2-page document explaining to this person how your completed lab project works, including the SysTick timer, the multiplexed digit display, and the button with the GPIO interrupt.

As you write your report, make sure to:

- Explain why your project matters, and how it fits into the larger context of the world. Before you try to give details about a multiplexed display, help your reader understand what the point of such a thing is.
- Eliminate technical jargon wherever possible, and explain terms that might not be clear even if they're obvious to you. For example, your audience probably doesn't know what voltage is, except as a number on batteries.
- Use drawings and diagrams to explain your ideas. In fact, nobody says you have to have primarmily text with some figures; your document could just as easily be organized like a poster or presentation slide with one or more figures and succinct explanatory text.
- Pay attention to the visual formatting of the document. Does it look imposing and technical, or inviting and accessible? Think about how websites and other graphics are designed to convey information to general audiences.

What to turn in

Submit your report as a PDF on Gradescope like usual, but **do not put your name on it.**

For the first round of feedback, another student will review your draft and make comments. The course website has a link to a draft rubric which you can use to evaluate your own work before submitting.