EE 14 Lab 2: Reaction time!

Lab report due a week after your lab session (10-14 February 2025)

1 Introduction

In this lab, you'll create a reaction-time game using some buttons and LEDs. This will give you additional practice working directly with registers and with GPIO pins, while also building up a larger piece of code. You'll use a state machine to organize the logic of the game, which is a very powerful design pattern for structuring software.

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

- Use the `stm32l432xx.h` header file to simplify register access
- Use GPIO inputs to read buttons, using the internal pull-up resistors
- Make a state machine in C with non-trivial behavior

Documentation focus: State diagrams — your code will implement a state machine, and you will create a pretty diagram to illustrate and explain its behavior.

2 Prelab

Decide what the rules for your game should be. We recommend one of the two variants below, but you are welcome to propose something else to your TA. It is better to start simple and work from there; you'll have plenty of opportunity to show off when we get to the final project!

Single-player game

1. Initially, all LEDs are off
2. A blue LED turns on, and the player attempts to push the button as quickly as possible in response
3. LEDs light up to indicate how well the player did: Green the interval is under 200 ms, yellow under 250 ms, and red over 250 ms.
4. Press the button again to play again

Two-player game

1. Initially, all LEDs are off
2. A blue LED turns on, and each player attempts to push their button as quickly as possible
3. Whichever player presses their button first, their green LED turns on (and the other does not)
4. Pressing a button again turns off all LEDs and starts a new game.

P1: Decide on the rules for your game, and formalize how it will work. This should include:

- A concise English description of the game
- How many buttons you need, and which Nucleo pins they will connect to
- How many LEDs you need, and which Nucleo pins they will connect to
- A state transition diagram showing all of the possible states, and the inputs that cause transitions.

P2: Using the STM32L4xxx reference manual, find the following information:

- What should the bits of the GPIO port mode register be set to to enable *input* mode?
- Which register enables the GPIO pin pull-up resistors? *Our chip also includes pull-down resistors. You're welcome to use those instead if you prefer.*
- Which register holds the port input data? (i.e., which register does your code need to read in order to read a switch?)

As with last week, you don't actually have to be in lab to get started; you've got everything you need in your lab kit!

3 In lab: warmup

L1: Wire up your switches to your Nucleo board. If you are using pull-ups, then the switch will make a connection between the pin and ground (meaning the microcontroller will read a zero when pressed). If you're using pull-downs, then the switch should connect between the pin and 3.3 V (meaning the microcontroller will read a 1 when pressed).

L2: Wire up your LEDs to your board.

L3: Create an empty PlatformIO project with the same settings as last time:

- **Name**: Whatever you want, but hopefully something descriptive like `EE14_lab3`!
- **Board**: ST Nucleo L432KC
- **Framework**: CMSIS

L4: Create a new `.c` file, and write code so that one of your LEDs turns on when the switch is pressed.

You should follow something like the pseudocode below:

```c
#include "stm32l432xx.h"

int main()
{
  // Set up the GPIO pins as inputs/outputs

  while(1) {
    // Read the switch input
    // Set the LED output
  }
}
```

Several things can make your life easier:

- There is a header file which contains all of the register offsets for the chip, which you can include with `#include "stm32l43xxx.h"`. These are set up as `structs` for each register, so you can use the offsets like this:

  ```c
  // Set the GPIO B enable bit in the RCC AHB2 enable register
  RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
  ```

- You may want to define little functions to set and read values from the GPIO pins, so that your code is easier to read.

Implementing the game

See the slides on the course website about how to implement a state machine.

You should also look at this section from Making Embedded Systems (You'll need to sign in to O'Reilly with your Tufts email which will take you to the SSO login).

> **L5**: Implement your game using a state machine! We'll learn later about how to use timers, but for now a reasonable way to track time is simply to count iterations of a loop:

```
volatile int i; // Make it volatile so the optimizer doesn't remove it!
for(i = 0; i < 1000; i++){ }
```

or

```
volatile int i = 0;
while(/*thing that we're waiting for hasn't happened*/){
  i++;
}
// Now we can look at i to see how long we waited
```

To create unpredictable intervals, you have a few options:

1. If you make the delay longer than a couple seconds, it will feel random enough; people will have to work off of their reaction time rather than rhythm.
2. You can count how many loops run while the code is in the waiting state (waiting for the player to restart the game). Since you're counting very fast, and human input is unpredictable, you can use this as a random basis for calculating a delay interval.
3. The STM32 includes a random number generator which is easy to use. Just enable it, and then you can read truly random numbers out of the data register. See section 24 of the reference manual for details!

> **L6**: Show off your game to your TA, and see who has the faster reaction time!

4 What to turn in

Your lab report should contain the following:

- A cover page with your name, date, lab section, and lab TA names, and a concise description of your game.
- An updated state diagram. This must be drawn with a computer in a vector editing tool (no whiteboard drawings or iPad sketches, and no Paint/Photoshop). If you need some tool suggestions, I've seen great things drawn with Graphviz Dot (save as SVG/PDF!), Mermaid (`mermaid.live`), Inkscape, Adobe Illustrator, PowerPoint, and OmniGraffle.
- Answers to the following questions:
  - What do you understand now, that you didn't before the lab?
  - What are you confused or curious about? (You're not allowed to say "nothing" — once again, I have several things I'm confused and curious about after *writing* this lab.)
  - How long did it take you to complete this lab? (This will help us calibrate the workload for future semesters.)
- Your completed code, as text in a monospace font at the end of the report (not a screenshot!). You should be able to copy and paste straight out of VSCode and keep the syntax highlighting. The LaTeX listings package is also a great way to typeset code.