EE 14 Lab 2: Blinky blink!

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

1 Introduction

In this lab, you'll write C code from scratch to control an LED. This is the "hello, world" of microcontrollers, and is the fundamental demonstration of "general-purpose input/output" (GPIO).

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

- Find your way around a 1600-page datasheet without panicking
- Write C code to set memory-mapped GPIO registers
- Use an oscilloscope to measure the speed of code running on a microcontroller
- Capture screenshots from an oscilloscope

Instead of a full lab report, each lab in EE 14 will have a specific "documentation focus", which is one particular thing we want you to learn how to do very well. Each one will probably require both some learning and some tweaking, so don't try to rush it; take the time to do it well.

Documentation focus: Oscilloscope captures — you will capture a waveform from an oscilloscope and annotate it to demonstrate how your code is running.

Unfortunately, only the newer Rigol oscilloscopes can save images to a flash drive, so sit at a bench with one of those if you have the choice. You may need to take turns with a classmate.

2 Prelab

Look at the Nucleo L432KC pin diagram (link on the course website), and pick a pin that you'd like to connect an LED to. It should have a board identifier (like "D3" or "A2") which is printed on the Nucleo board, and a corresponding pin/port identifier (like "PB_0" or "PA_3"). The letter in the identifier signifies which GPIO port the pin is controlled by (either PORT A or PORT B), and the number is the index within that port. Note that the board identifiers are not used at all in the software; they're just a concise way to identify the connections on the PCB.

> **L1**: What port and pin number did you choose?

Open the STM32L41xxx reference manual (link on the course website). Observe that it has 1628 pages. Take a deep breath; you *can* read and understand this; it'll just take some patience and practice!

You'll need to find information on three registers:

- The GPIO port mode register (whether a particular pin is an input or output)
- The GPIO port output register (the value of the pin, if it is configured as an output)
- The AHB peripheral clock enable register (which enables/disables whole features of the chip)

For each one, we need to know 1) the register's address (i.e., what memory address to write to), and 2) which bits in that register correspond to the function we want (i.e., what data to write).

Look in the GPIO registers subsection.

L2: What is the address offset for the GPIO **port mode** register?

L3: The port mode register has 32 bits. Which bit(s) of the register correspond to your chosen pin? What should those bits be set to in order to enable general-purpose output mode?

L4: What is the address offset for the GPIO **output data** register?

L5: The output data register has 32 bits. Which bit(s) of the register correspond to your chosen pin?

Look in the reset and clock control (RCC) registers subsection.

L6: What is the address offset for the **AHB2 peripheral clock enable** register?

L7: Which bits of the register correspond to the enable for your GPIO port (either GPIOA or GPIOB)?

Now we need to find the memory offset of the GPIO port and clock control. Find Table 2, in the "Memory organization" section of the document.

L8: What is the base address (i.e., the starting "boundary address") for the GPIO port corresponding to your chosen pin (either GPIOA or GPIOB)?

L9: What is the base address (i.e., the starting "boundary address") for the RCC port?

With all those numbers in hand, you're ready to write some code! You don't actually have to be in lab to get started; you've already got everything you need to give it a try.

3 In lab

L10: Create an empty PlatformIO project with the same settings as last time:

- **Name**: Whatever you want, but hopefully something descriptive like `EE14_lab2`!
- **Board**: ST Nucleo L432KC
- **Framework**: CMSIS

L11: Create a new `.c` file in the `src` folder. You can use the following pseudocode as an outline.

```
// You may want to use #define to set up the register offsets as constants

int main()
{
  // 1. Configure the RCC to enable the GPIO clock
  // 2. Configure the GPIO port/pin as an output
  // 3. Set the GPIO pin to be a 1 (turn on the LED)

  while(1) { } // Loop forever
}
```

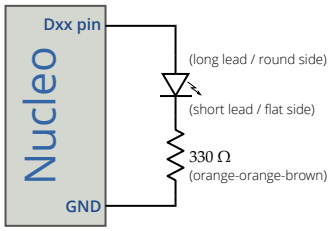
To set a value of a register, first create a pointer with the address, and then dereference it:

```
#define SECRET_REGISTER 0x20001400

volatile unsigned int* reg = (unsigned int*) SECRET_REGISTER;

*reg = 0x8041;
```

L12: Fill in the pseudocode above with the appropriate C code to set the GPIO value high, and wire up an LED to the pin on your breadboard.



L13: Run your code. If everything is correct, your LED should turn on! If not, check that you're writing the correct registers and values. You may also find it helpful to use the debugger (see section 5).

Debugging note: If you get an error "unable to connect to target" when trying to flash your code, this is usually caused by your code overwriting some of the PORT A register bits which correspond to the programming pins on the chip. To flash your code if this happens, hold down the reset button and click "upload". When you see the message `hla_swd`, then release the reset button quickly, and it should complete the upload process.

4 But how fast is this?

L14: Put code inside the `while` loop to turn the LED on and off repeatedly. It'll be blinking too fast for your eyes to see, but you should be able to tell that it's on dimly.

L15: Use an oscillscope to examine the voltage on your chosen GPIO pin. How long does it take for one iteration of the loop?

L16: How long do you think it takes the processor to execute each instruction? Capture an oscillscope screenshot using the USB flash drive (not a phone picture), and annotate it to show your answer.

Some things you may find useful in your investigation:

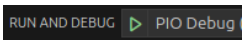
- You can "unroll" the loop by writing code to turn the LED on and off multiple times inside each loop iteration. This will let you see the difference between the time to execute the on/off instructions and the loop branch instruction.
- You can examine the assembly code and see how many instructions are actually executed by the processor. Read the following section to learn how.
- Bit masking takes multiple instructions (read, perform the AND/OR operation, and write). The STM32L4 chips include a "GPIO port bit set/reset register" which performs bit masking for you and only requires one instruction. Essentially, writing a 1 to a particular bit in the set/reset register sets the corresponding output bit to 1, while writing a 0 leaves it unchanged (rather than making it 0). You can read about this in section 8.5.7 of the reference manual.

5 Debugging and viewing the generated assembly code

1. If you want to see the assembly code, open the `platformio.ini` file and add the line

```
debug_build_flags = -O1
```

This will turn on compiler optimization during debugging. If you're just trying to debug your C code, don't do this.

2. Open the debug pane by clicking "Run and Debug" in the activity bar.
3. Start a debug session by clicking the "Start Debugging" button at the top. 

4. This should open a window with the code, with one line highlighted in yellow.
5. If you wish, you can run the code one line at a time by clicking the "step over" button  . You should be able to see the exact instructions that cause the LED to turn on and off, and the branch that makes the loop.
6. If you're trying to get your code to work, it can be helpful to poke the peripheral registers directly using the "Peripherals" pane on the left side. To change a value, right-click and select "update value".

   If the GPIO registers aren't updating, it's probably because the GPIO clock is disabled — enable it in the AHB peripheral clock register, and try again.

6 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 1-2 sentence description of the lab.
- Your annotated oscilloscope screenshot. A technical reader (e.g., another EE 14 student) should be able to look at your figure and understand exactly how you determined how fast the microcontroller is executing code.
- 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" — I have about a dozen 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.