Homework 3 - Timing stuff

First steps

Set up your development board to blink an LED at 1Hz.

For the ESP32-C3 using Espressif IDF (IoT development framework), you can use the following code:

```
#include "freertos/FreeRTOS.h"
#include "freertos/task.h" // Contains thread delay function
#include "driver/gpio.h" // GPIO pin controls

#define BLINK_GPIO 33 // Change this to whatever GPIO pin you're using

void app_main() {
    gpio_reset_pin(BLINK_GPIO);
    gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);

    while(1){
        gpio_set_level(BLINK_GPIO, 1);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        gpio_set_level(BLINK_GPIO, 0);
        vTaskDelay(500 / portTICK_PERIOD_MS);
    }
}
```

Timing the loop

Remove the timing delays from the code, so that the “blink” code runs as fast as possible. Use an oscilloscope or logic analyzer to measure the frequency at which the pin is being toggled.

- How long does it take to set the GPIO pin using `gpio_set_level()`? Record some notes, oscilloscope shots, or drawings of the waveform to explain how you got this answer.

Running faster

We saw in class that abstractions often get between us and maximum performance. Replace the `gpio_set_level()` calls with direct writes to the appropriate registers in the chip.

You’ll need to read the ESP32-C3 technical reference manual for details on the registers. (And yes, this is a 739-page document. Learning to navigate such monstrosities is one of the objectives of this course!)

Note: The ESP32-C3 has special “set” and “clear” registers associated with the GPIO register, so you should be able to toggle an individual GPIO pin without having to read/write the whole register.

You may find the header files `soc/gpio_reg.h` and `soc_gpio_struct.h` helpful. The macro `REG_WRITE(ADDR, VALUE)` may also be useful.

- How fast can you set a GPIO pin now? Is this surprising?
- The ESP32-C3 processor core runs at 160 MHz. Does your GPIO speed make sense relative to this number? Why or why not?

Optional: Even faster?

The Espressif documentation describes a "dedicated GPIO" mode, where the GPIO is controlled by a *special CPU instruction* rather than being memory mapped. Figure out how to do this. How fast can you go now?

Note that you don't actually have to write any assembly code; there are wrappers around the instructions in `hal/dedic_gpio_cpu_ll.h`.

Printing debug information

Use the `printf()` function to print out a message to the serial port, and use the PlatformIO serial monitor to confirm that the message is printed. You may find it easiest to print out the message repeatedly so that you don't miss it right when the board boots up.

Timing printf()

- How long does a single call to `printf()` take?
- Try printing some messages of different lengths. Can you write an equation relating the function time versus the length of the string? Make sure to explain how you figured out your answer!
- Suppose you have the following code:

  ```
  while(1){
      gpio_set_level(BLINK_GPIO, 1);
      printf("on!\n");
      gpio_set_level(BLINK_GPIO, 0);
      printf("off!\n");
  }
  ```

  Based on everything you've done so far, how fast will this loop run? Try it and check whether it matches your prediction!

What to turn in

Put a document (Google doc, Word doc, PDF) in the class Google Drive folder with your answers to the bullet-point questions and any notes you took along the way. Create a subfolder with your name under homework 3, and put the file(s) in your folder.

It doesn't need to be a fancy writeup, but it should be clear to other engineers (i.e., myself and your classmates) what exactly you did and where your numbers came from.

Make sure to report your results in "engineering units" with the appropriate SI prefixes (micro, kilo, mega,etc). Saying 120 MHz immediately means something to an engineer; I have lots of reference points for that number and can quickly put it in context. 1.2×10^8 Hz just doesn't click as fast.