Blinker Frequency

STM32F466RE

June 23, 2024

Default and Delayed Frequencies

If the firmware's only instruction is to blink an LED on the board, the blinking won't be visible at its core frequency (f) of 84MHz. We want to observe the blinking with natural vision, so the solution is to add a number of cycle delays (N_d) to produce a new, significantly-reduced frequency (f').

$$f = \frac{N}{T} = 84000000 \ \frac{cycles}{second}$$

$$f' = \frac{N'}{T} = \frac{N/N_{delays}}{t}$$

*f' is controllable, as opposed to f which is determined by the chip's architecture.

Initial implementation of cycle delays could be achieved with a for loop that executes only one, null instruction. At first glance, one iteration requires one cycle, so iterating ($i = 0;\ i < N;\ i++$) predicts a requirement of N_{loop} iterations: $N_{loop} = N = 84\ million\ cycles$. Consider the following:

```
for (uint32_t i = 0; i < N; i++) {
      __asm__("nop");
}
gpio_toggle(led_port,led_pin);
```

$$N_{loop} = \sum_{i=0}^{N} 1 = 84000000\ iterations \stackrel{?}{=} 84000000\ cycles$$

This summation assumes that 1 iteration of the *for loop* is performed within one CPU cycle. However, observing the LED after a firmware update, it's clear that the order of magnitude is approximately correct, yet f' is 4-5 times smaller than expected.

Why is this the case if our application runs at 84MHz and our for loop executes only one command?

- The application is running on the system at N cycles per second, but each iteration of our for loop costs more than 1 cycle.

- N iterations of this $for\ loop$ (N_{loops}) costs cN cycles, where $c > 1$.
- $N_{loops} < N$ must be satisfied to achieve $f' \leq 1$.

Error is generated solely from the following assumption:

$$N_{loop} = \sum_{i=0}^{N} 1 = N = 84000000\ cycles$$

Empirically, it is clear that the above equality is incorrect and the following is true:

$$N_{loop} = \sum_{i=0}^{N} c_i > N$$

Since N_{loop} is chosen by us, and N is known, we can accurately determine the nunber of CPU cycles consumed by each iteration.

Re-arrange eq. 2 to obtain the number of cycles per $for\ loop$ iteration, denoted c_i:

$$N_{loop} = c_i \sum_{i=0}^{N} 1 = \frac{N}{f' * 1\ second}$$

$$c_i = \frac{1}{\sum_{i=0}^{N} 1} \frac{N}{f' * 1\ second}$$

$$c_i = \frac{1}{N} \frac{N}{f' * 1\ second}$$

$$c_i = \frac{1}{f' * 1\ second}$$

Unsurprisingly, the observed frequency (f') is inversely proportional to the number of cycles used per loop-iteration. The required number of iterations is over-estimated in our firmware, resulting in the LED blinking *too slowly*. Reduce the number of iterations and the blinker rate will increase; reducing them by a factor of c_i should yield our target of $f' = 1$. This is our correction factor for N_{loop}.

Update the firmware and measure the blinker frequency. If the proposed relationship between f' and c_i is correct, c_i will be an integer. Likeewise, machine instructions should not cost *partial* cycles; computing operations are mostly discrete. If f' and c_i aren't integers, there would be a problem.

```
uiint32_t Nloop = 84000000 / 4;
for (uint32_t i = 0; i < Nloop; i++) {
     __asm__("nop");
}
gpio_toggle(led_port,led_pin);
```

Re-assuring – a blink frequency of $f' \approxeq 0.25$ *cycles per second*. The inverse of this frequency gives the number of cycles required to complete one iteration of the *for loop*:

$$c_i \approxeq \frac{1}{0.25\frac{cycles}{second}} \frac{1}{1\ second} = 4\ cycles$$

At 60fps, I measured $f' = 1.00\frac{cycles}{second}$; it is safe to conclude that c_i is highly accurate. This is probably not accurate down to the milisecond scale, and that can be explored deeper during a later module.

Although tedious and inefficient, we determined how much time (or how many cycles) are required to execute certain machine instructions, without having to read any corresponding assembly code. Reduce the implemented delay by a factor of c_i and re-measure f' to confirm that it approaches $1\ \frac{cycle}{second}$.

Firmware Source Code:

```
//Blinker Firmware 2024.06.18
#include <libopencm3/stm32/f4/rcc.h>
#include <libopencm3/stm32/f4/gpio.h>

#define LED_PORT (GPIOA)
#define LED_PIN (GPIO5)

static void rcc_setup(void) {
    rcc_clock_setup_pll(&rcc_hsi_configs[RCC_CLOCK_3V3_84MHZ]);
}

static void gpio_setup(void) {
    // Turn on the clock for a given periphereal
    rcc_periph_clock_enable(RCC_GPIOA);

    // Configure the LED pin. Send output to Port A
    gpio_mode_setup(LED_PORT,GPIO_MODE_OUTPUT,GPIO_PUPD_NONE,LED_PIN);
}

static void maintain_frequency_standard(uint32_t cycles) {
    for (uint32_t i = 0; i < cycles; i++) {
        __asm__("nop");
    }
}

int main(void) {
    rcc_setup();
    gpio_setup();

    while (1) {
        // Turn the LED on and then off at freq (f')
        gpio_toggle(LED_PORT,LED_PIN);
```

```
        maintain_frequency_standard(84000000/4);
    }

    return 0;
}
```