Blinker Frequency

STM32F466RE

June 29, 2024

*CPU pipelining allows instructions to be processed in parallel, making it difficult to confidently determine how many loop iterations occur per clock cycle, even for simple programs; Intel describes this as "a form of parallelization where multiple iterations of a loop execute concurrently" [1]. Disentangling the effects of pipelining from these computations is impossible at this level of depth.

Default and Delayed Frequencies

If a firmware's only instruction is to blink an onboard LED, blinks will occur at around* the core frequency (f) of 84MHz. We want to slow the blinking such that it's naturally visible, so we'll need to to add a number of cycle delays (N_d) to our application to target an easily-observable frequency (f').

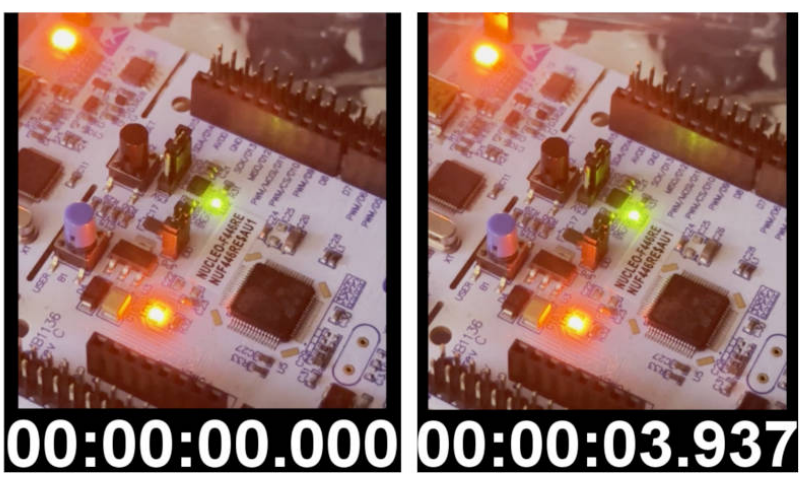
$$f = \frac{N}{T} = 84000000 \ \frac{cycles}{second}$$

$$f' \propto N' = \frac{N}{N_d}$$

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

Initial implementation of cycle delays could be achieved by a *for loop* that executes only one 'no-operation' instruction. At first glance, one iteration of this loop requires one cycle, so iterating $(i = 0; \ i < N; \ i++)$ predicts a requirement of $N_d = N_{loop}$ iterations, where $N_{loop} = N = 84 \ million \ cycles$; this corresponds to our desired $f' = 1 \ Hz$. Consider the following code excerpt:

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



Flashing this firmware and observing the LED, it's clear that the order of magnitude is approximately correct, yet f' is 4 times smaller than expected. Why is this true if the application runs at 84MHz and each $for\ loop$ iteration executes only one *nop* assembly instruction?

- The application runs at N cycles per second, but each iteration of the $for\ loop$ costs more than 1 CPU cycle.
- N_d was over-estimated; the application is processing is slower than expected.
- N iterations of this $for\ loop$ (N_{loop}) costs cN cycles ($c > 1$).
- $N_{loops} < N$ 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:

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

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

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

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

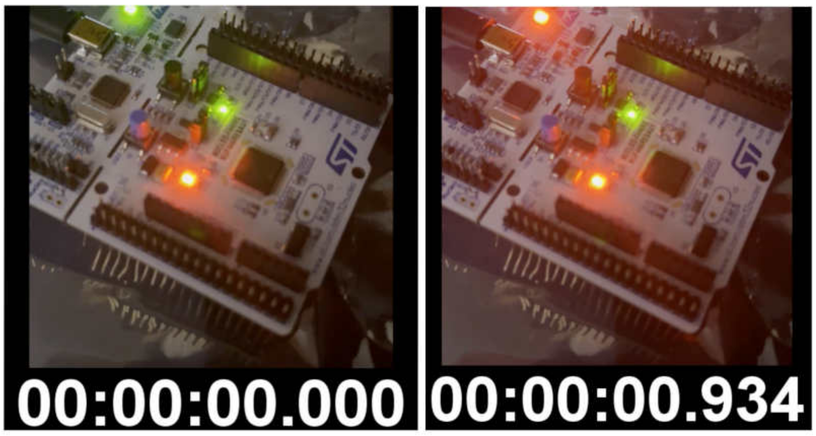
This coefficient represents the number of cycles required to complete one iteration of the $for\ loop$, and we may calculate it using the frequency recorded above:

$$c_i = \frac{1}{0.2540\ \frac{cycles}{second}} \frac{1}{1\ second} \approxeq 4\ cycles$$

Unsurprisingly, the observed frequency (f') is inversely proportional to the number of cycles required for one iteration. Our firmware over-estimated this requirement, 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 $f' = 1$. That is, $f' = 1$ when $N_{loop} = \frac{1}{c_i} N$

Update the firmware to reduce N_{loop} by a factor of c_i. If the proposed relationship between f' and c_i is correct, f' will approach 1 Hz.

```
    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 $f' = 0.934\ Hz \approxeq 1\ Hz$ (6.6 percent error). Although inefficient, we determined the time (or cycles) required to execute simple machine instructions, without reading any corresponding assembly code.

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) {
    rcc_periph_clock_enable(RCC_GPIOA);
    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) {
        // Toggle the LED state at freq (f')
        gpio_toggle(LED_PORT,LED_PIN);
        maintain_frequency_standard(84000000/4);
```

```
        }
        return 0;
    }
```