

Exercise - 02b

Flash LED with blocking delay (GPIO)

Andreas Habegger | Adrian Steiner
BTS3230 | Version 1.0.0 of 02.09.2026

Please be aware that the content is subject to change at any time. For the latest version, please check the website.

In this exercise you will create your first autonomous application. Instead of directly mirroring the state of the user button, the MCU shall toggle a LED periodically. To generate a visible delay, a simple software loop is used.

Objectives

- ▶ Generate an output signal without user interaction.
- ▶ Understand the concept of a superloop.
- ▶ Implement a simple software delay using a for loop.
- ▶ Observe the impact of blocking code execution.

Outcomes

- ▶ Configure a GPIO output in software.
- ▶ Toggle an LED periodically.
- ▶ Understand how a blocking delay affects the execution flow.
- ▶ Learn the limitations of software-based timing.

Description

In the previous exercise, the LED state depended on the state of the push-button. In this exercise, the MCU shall repeatedly switch the LED on and off at a fixed interval.

The delay between two state changes is generated entirely in software. A simple for loop executes a large number of instructions and thereby occupies the CPU for a certain amount of time.

While the CPU is executing this delay loop, no other application code can run. This behaviour is called a blocking delay.

Note

Only the Nucleo-64 board is used in this task.

Tasks

In the previous exercise, you implemented the configuration of a GPIO input and output. You may either reuse your previous implementation or configure the GPIOs again as a refresher. Use the green LD2 LED on the Nucleo-64 board as the output.

Inside the superloop, toggle the LED with a frequency of **0.5 Hz**. The delay between two state changes shall be implemented using a simple software loop.

 Example Loop Delay

```
for(volatile uint32_t count = DELAY_COUNT; count > 0; count--)  
{  
    asm("NOP");  
}
```

 Question

- Why is the variable declared as `volatile`?
- What is the purpose of the expression `asm("NOP");`?
- Can you calculate the macro `DELAY_COUNT` required to achieve the desired frequency of 0.5 Hz?

 Solution

- The `volatile` keyword prevents the compiler from optimizing accesses to the variable. The compiler must read and write the variable exactly as specified in the source code. Without `volatile`, the compiler may recognise that the loop has no observable effect and remove parts of it or even eliminate the entire loop. In general, `volatile` is required for variables that may be modified by hardware, interrupts, or other program contexts outside the current execution flow.
- `asm("NOP");` inserts the processor instruction *No Operation (NOP)*. The CPU executes the instruction but performs no operation. This ensures that each loop iteration consumes processor cycles and contributes to the delay.
- An exact value cannot be calculated solely from the C code. The actual delay depends on the CPU clock frequency, the generated machine instructions, and the compiler optimisation settings. Only an approximate value can be determined theoretically. For precise timing, a hardware timer should be used instead of a software delay loop.

Implementation

 Exercise

1. Enable the clock of the GPIO port connected to the LED.
2. Configure the LED pin as a GPIO output.

3. Implement a superloop that:

- Waits using the software delay
- Toggles the LED

4. Verify that the LED blinks with the desired frequency. Modify your loop delay time if it does not match

Results

If all is well, the LED should blink continuously with a frequency of approximately **0.5 Hz**.



Measure Frequency

5. Measure the blinking frequency using a stopwatch (for example, on your smartphone).

- Do you observe any irregularities?
- How accurate is your measurement method?

6. Measure the timing more precisely using the Saleae logic analyser.

- Connect the Saleae input to the LED Morpho pin.
- Measure the frequency and period of the signal.
 - Do you still observe any irregularities?
 - How accurate is the generated frequency?
 - Is the measured value equal to the expected value?
 - Can you determine the timing resolution of the software delay?



Discussion

Discuss the limitations of software-based delays with your classmates.

- ▶ Is the generated frequency precise?
- ▶ What happens if the CPU clock frequency changes?
- ▶ How does compiler optimisation influence the delay?
- ▶ Can the MCU perform other tasks while executing the delay loop?
- ▶ How reproducible is the delay across different compiler settings?
- ▶ Which hardware peripheral could be used to generate a more accurate timing signal?

 Non-Blocking delay

Implement the same exercise using a non-blocking delay mechanism.

1. Replace the blocking software loop with a solution that allows other parts of the application to execute while the delay is running.
2. Extend the application so that the state of the user button is evaluated continuously and the LED can be enabled or disabled immediately when the button is pressed.

Compare both implementations:

- ▶ Which implementation is more responsive?
- ▶ Which implementation uses the CPU more efficiently?
- ▶ Which implementation is more precise?
- ▶ What are the advantages and disadvantages of each approach?