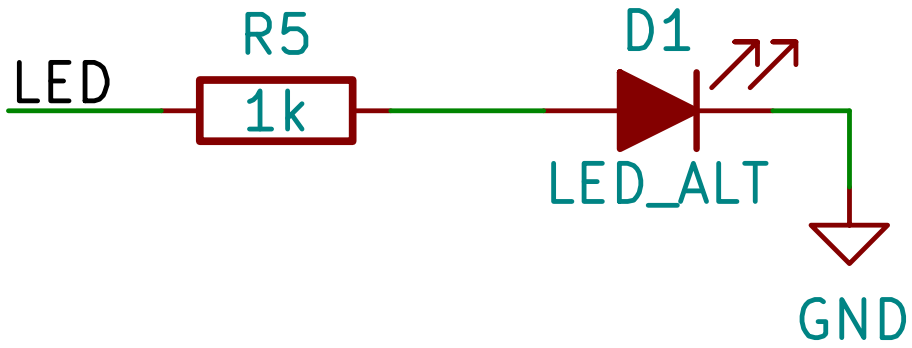
Мигаем светодиодом

В нашем устройстве один светодиод: и он был добавлен не случайно. В мире встраиваемых систем мигание светодиодов, это своего рода «Hello world» программа. Она как минимум поможет понять, что микроконтроллер живой.

Если модуль файлы в проекте под светодиод у вас ещё не созданы, добавьте их: `led.h` и `leds.c`. Самое время применить полученные знания на практике.

Сразу встаёт вопрос, — «А как правильно подключить светодиод к микроконтроллеру?», скажем пока, что через резистор (подробнее в конце курса), сейчас же нас заботит только одно — как заставить его светиться. Впрочем, вы уже должны это знать, сильно абстрагируясь от физики, всё что нам нужно сделать — это выставить высокий логический уровень (т. е. напряжение 3,3 В) на нужной ножке МК. Установив низкий логический уровень (т.е. 0 В) мы, соответственно, светодиод погасим.



Как видите, один вывод (анод, он же «плюс») светодиода подключен к микроконтроллеру через резистор, а второй вывод (катод, он же «минус») подключен к земле. Вам должно быть известно, что ток побежит через светодиод в сторону земли [1].

Сверьтесь со схемотехникой и определите к какой именно ножке он подключен. Общая последовательность работы с портом будет выглядеть примерно так:

- включить тактирование порта;
- настройка нужной ножки в нужный режим;
- управление нужной ножкой.

Для начала, создадим прототипы функций в модуле `led`. Нам нужна функция инициализации порта, включение, выключение светодиода, а так же изменение его текущего состояния на противоположное.

```
// leds.h
oid led_init(void);
void led_turn_on(void);
void led_turn_off(void);
void led_toggle(void);
```

Сразу же добавьте вызов функции инициализации в `init_mcu()`, чтобы не забыть, а затем вернитесь к `led_init()` и реализуйте её. Постарайтесь не смотреть на код ниже и напишите его самостоятельно.

```
void led_init() {
    // turn on GPIOA clocking
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
    // set gpio pin mode
    GPIOA->CRL &= ~GPIO_CRL_MODE3_0; // set as
    GPIOA->CRL |= GPIO_CRL_MODE3_1;  // output with 2 MHz speed
    GPIOA->CRL &= ~GPIO_CRL_CNF3;    // set as push-pull output
}
```

На этом настройка завершена, осталось реализовать функции включения и выключения.

```
void led_turn_on(void) {
    GPIOA->ODR |= GPIO_ODR_ODR3;
}

void led_turn_off(void) {
    GPIOA->ODR &= ~GPIO_ODR_ODR3;
}
```

Подумайте, при помощи какой логической операции можно менять состояние бита в регистре.

▶ Ответ.

Для проверки работы, допишем в главный цикл буквально пару строк, которые будут мигать светодиодом.

```
// int main() {
while (1) {
    led_toggle();
    for (uint32_t i = 0; i < 1e5; i++);
}
```

Ввиду того, что мы пока не знаем, как организовать функцию задержки, воспользуемся бессмысленным циклом — он впустую будет интегрировать переменную и тем самым займёт ядро на некоторое время.

> **Почему стоит использовать `uint32_t` вместо `int` или `unsigned int`?**
>
> Всё очень просто — в разных системах\компиляторах размер `int` или `unsigned int` может отличаться. Для того что бы обезопасить себя лучше использовать чётко заданный размер и тип данных. `uint32_t` — беззнаковый целочисленный, 32 бита. Есть более коротка запись используемая в библиотеке, `stm32f10x.h`:
>
> ```
> typedef uint32_t  u32;
> ```

Запустите программу, светодиод должен мигать с какой-то неопределённой частотой.

Работа с регистрами эффективна с точки зрения выполнения, но не эффективна с точки зрения времени программиста. Все последующие модули мы будем так же рассматривать через регистры, и настраивать их через них. Постарайтесь на время этого курса заставить себя заглядывать в документацию и проверять все утверждения касательно регистров. Это вам поможет привыкнуть работать с ней и кристаллизовать понимание того, как работает микроконтроллер.

Как мы уже отметили раньше, библиотека значительно упрощает работу, взгляните как выглядит функция инициализации с использованием SPL:

```
void led_init(void){
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    GPIO_InitTypeDef gpio;
    gpio.GPIO_Pin   = GPIO_Pin_0;
    gpio.GPIO_Mode  = GPIO_Mode_Out_PP;
    gpio.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIOA, &gpio);
}
```

Код получается более читаемым и его без знания какие нужны регистры можно написать быстрее: открываете файлы библиотеки, находите функцию с говорящим названием и передаёте в неё требуемые аргументы.

Код урока можно найти в репозитории на github: **CMSIS**.

Назад | **Оглавление** | **Дальше**

---

1. В реальной жизни наоборот, но оставим это на совести Бенджамина Франклина. ↩