

Есть ли у вас вопросы?

# Предупреждение

Все электрические схемы,  
представленные здесь и далее, являются  
условными!

В них могут отсутствовать важные  
компоненты!

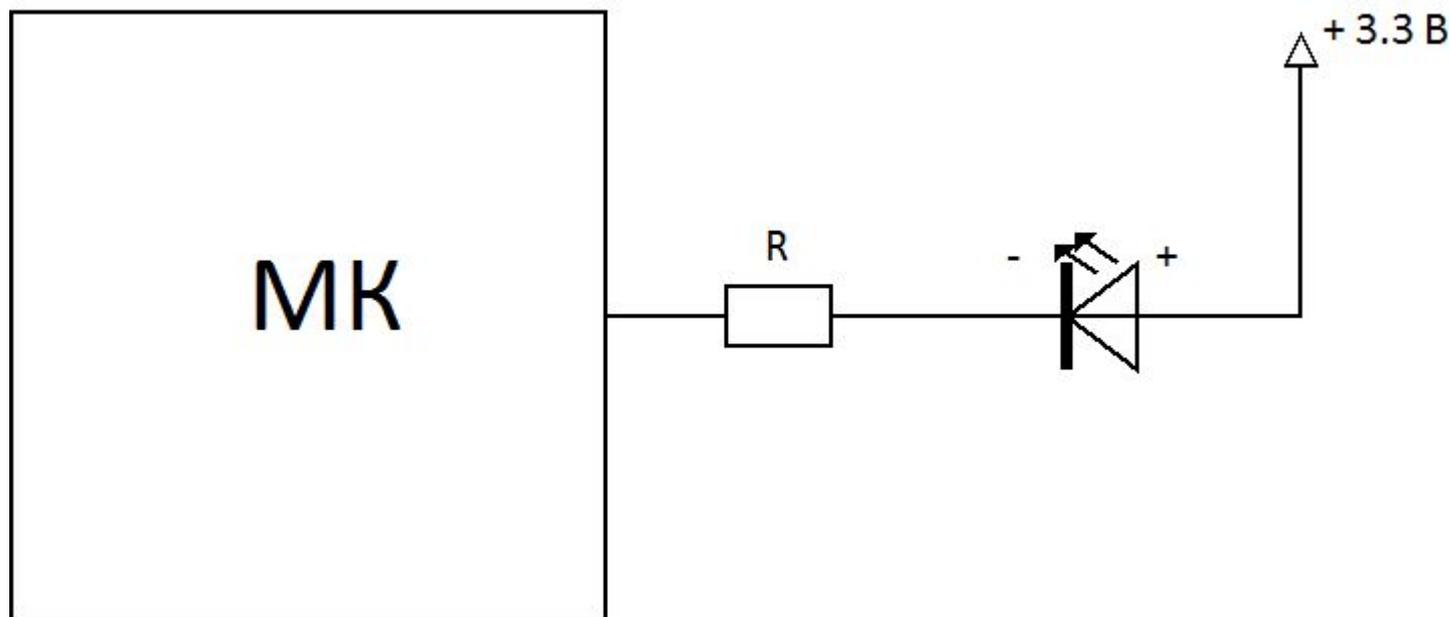
# Цели на сегодня:

1. Зажечь светодиод
1. Погасить светодиод
1. Помигать светодиодом
1. Помигать по нажатию кнопки (bonus level)

# Как зажечь светодиод?

Это зависит от того, как он подключен.

# Как зажечь светодиод?

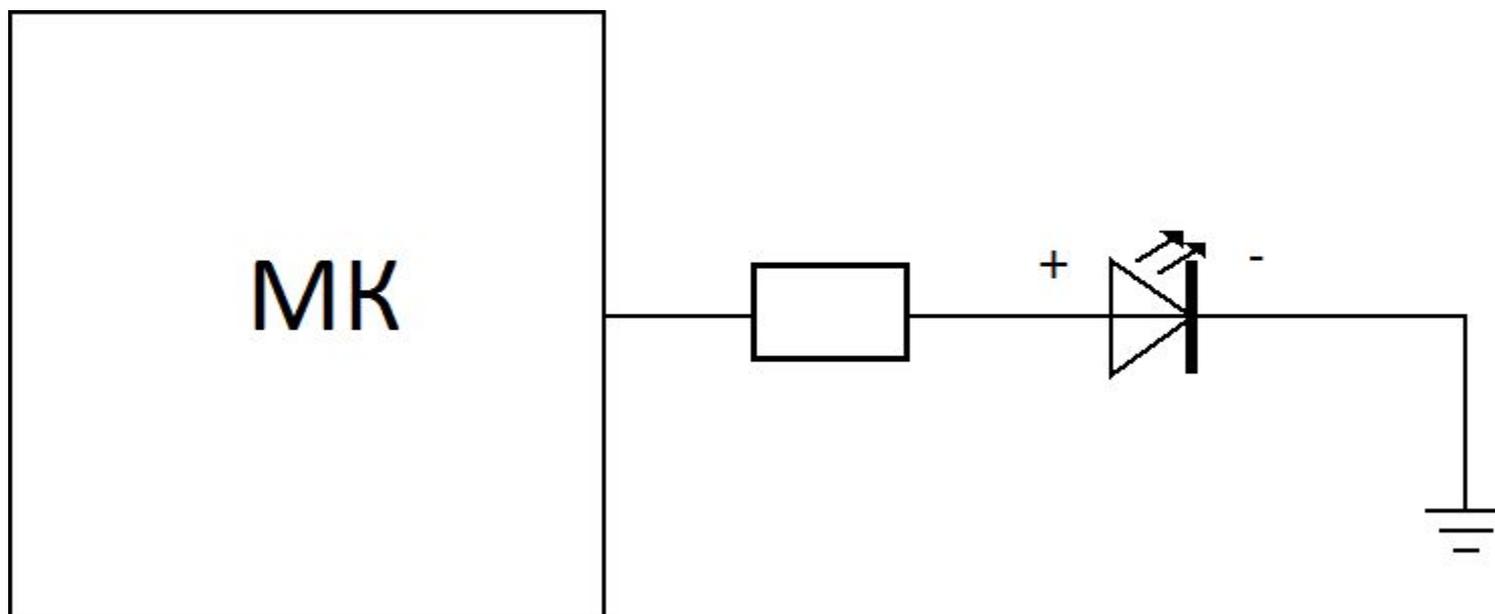


Как зажечь светодиод при таком подключении?

Нужно на вывод МК подать **низкий** уровень напряжения – светодиод загорится.

Если подать **высокий** уровень – светодиод погаснет.

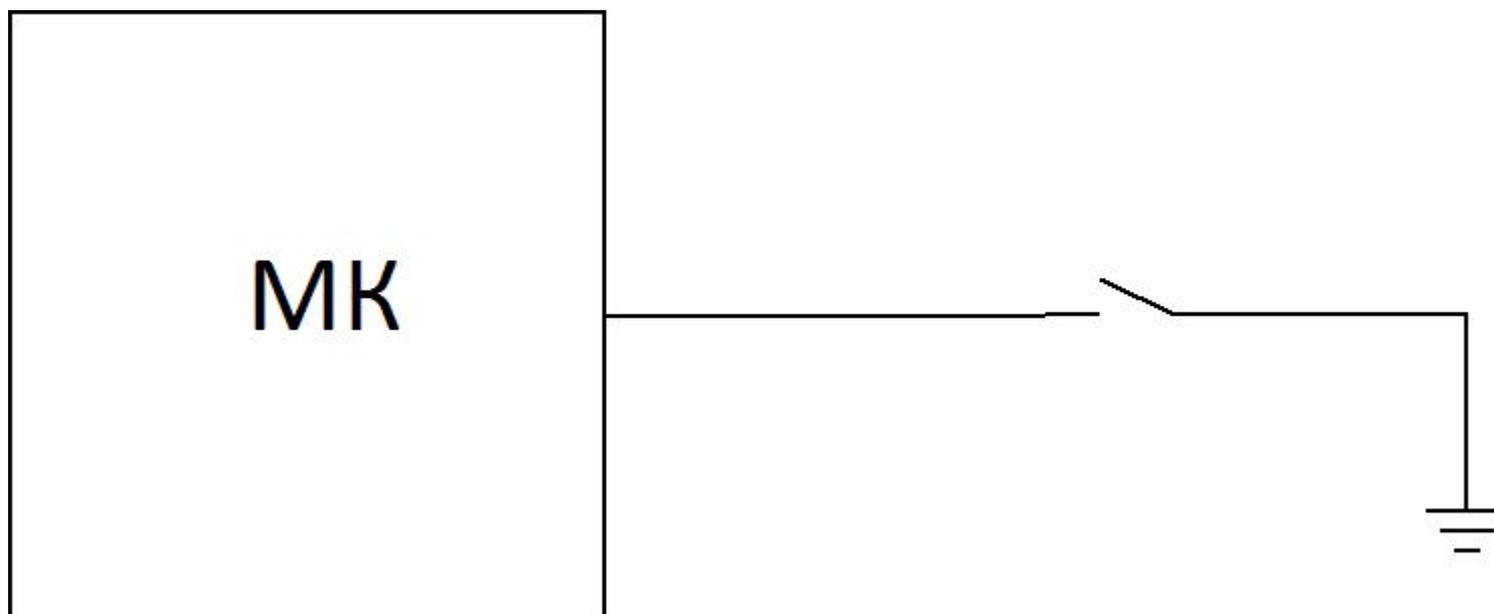
# Как зажечь светодиод?



Как зажечь светодиод при таком подключении? Очевидно, все наоборот. Нужно на вывод МК подать **высокий** уровень напряжения – светодиод загорится.

Если подать **низкий** уровень – светодиод погаснет.

# Как узнать, нажата ли кнопка?

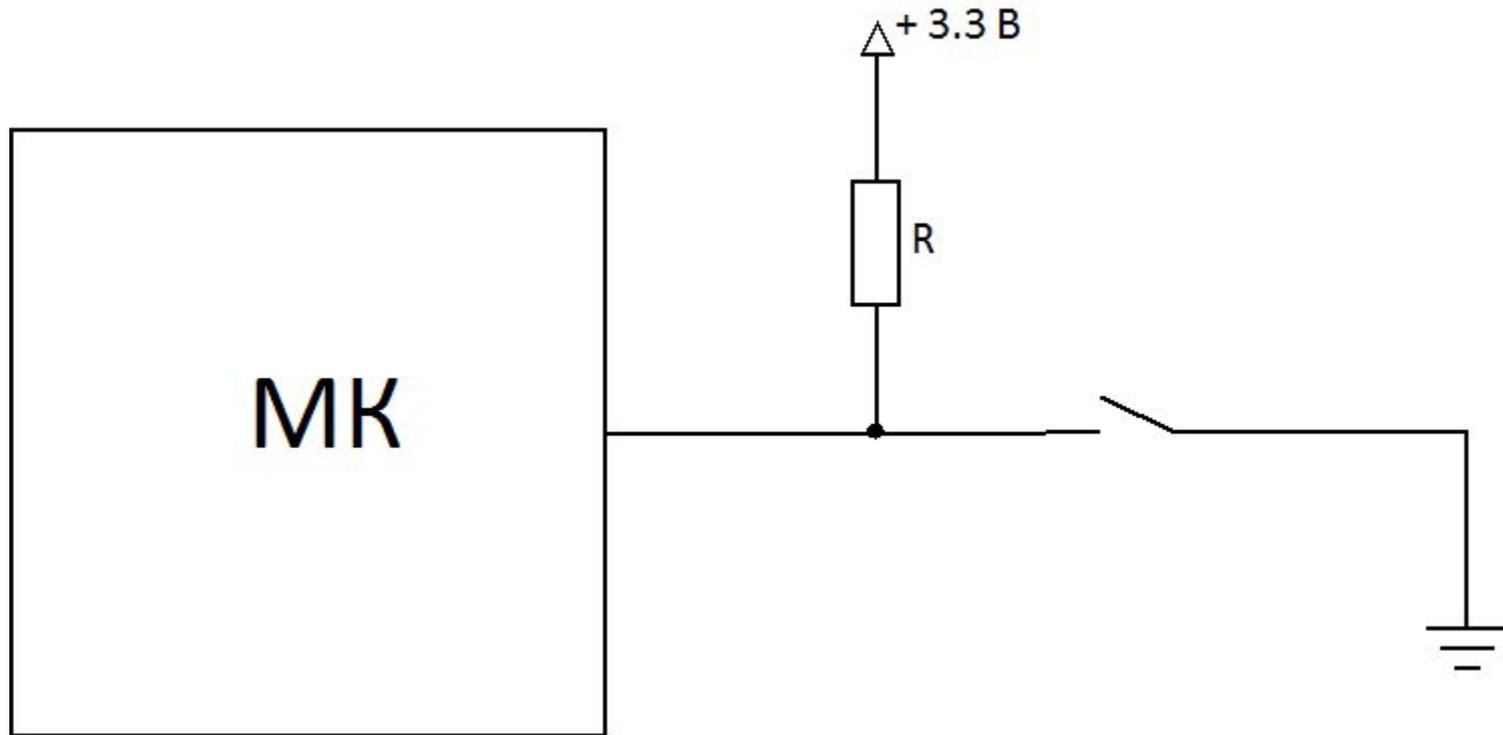


Нужно измерить уровень напряжения на входе.  
Если кнопка нажата – то уровень на входе будет низким.

А если не нажата?

Неопределенность. Это плохо. Что делать?

# Как правильно подключать кнопку



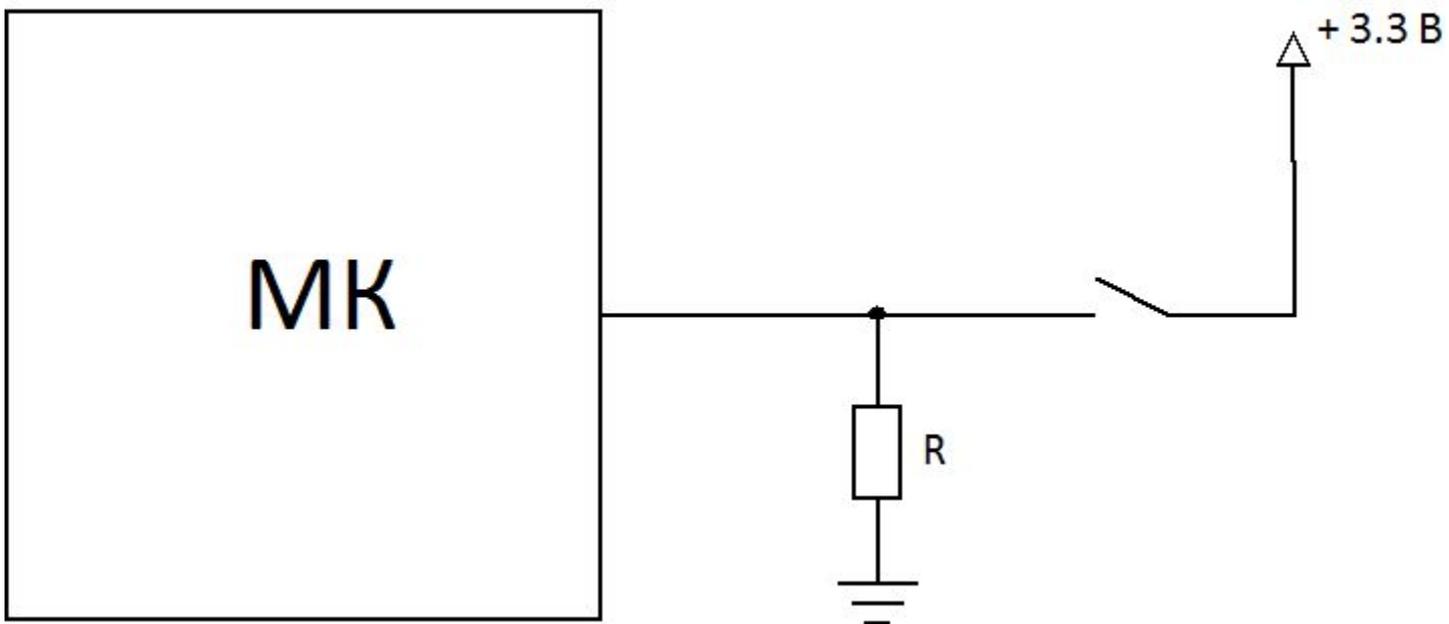
Теперь, когда кнопка не нажата, на входе будет высокий уровень.

Это называется «подтяжка к питанию» - Pull Up.

А зачем нужен резистор?

Чтобы не было КЗ, когда кнопка нажата.

# Но можно и наоборот



Это называется «подтяжка к земле» - Pull Down.

Теперь, когда кнопка нажата, на входе МК будет высокий уровень,

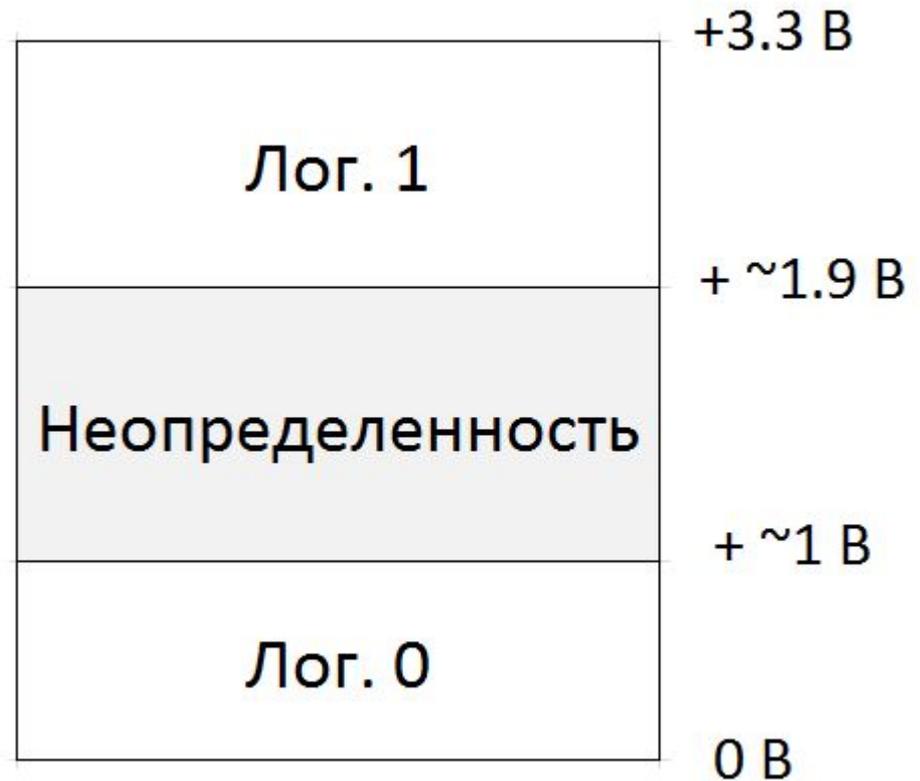
а когда не нажата - низкий

**Подтяжка может быть ВНУТРИ МК!**

# Логические уровни

Для stm32f103:

Для других устройств уровни могут быть другими; кодирование может быть инверсным и т.д.



# Контакты микроконтроллера

(они же «пины», «ноги», «выводы»)

## Тип:

- цифровой
- аналоговый

## Направление:

- ВХОД
- ВЫХОД

## Режим:

- Ввод/вывод общего назначения (GPIO)
- Альтернативный

## Режим входа:

- С подтяжкой (вверх/вниз)
- Без подтяжки (floating)

## Режим выхода:

- Комплементарный (Push pull)
- Открытый сток (Open drain)

## Текущее состояние:

- входа (только чтение)
- выхода (чтение/запись)

# Работа с периферийными устройствами

- Специальные команды ассемблера
- Ввод/вывод, отображенный на память (memory mapped IO) – регистры доступны по фиксированным адресам.

В последнем случае у каждого периферийного устройства есть набор регистров (не путать с регистрами ЦПУ).

Каждый регистр настраивает определенную функциональность.

Каждый бит в регистре что-то означает.

# Что из этого нам сегодня нужно?

Чтобы зажечь светодиод на плате discovery, нам нужна ножка в режиме комплементарного выхода (output push-pull).

Чтобы считать состояние кнопки – вход без подтяжки (input floating).

# Как же всем этим управлять?

Нужно как-то выбирать все эти режимы и состояния для каждого контакта! Как? Как должен выглядеть API?

С помощью специальных функций, которые кто-то уже написал за нас?

Но что делают эти функции?

# Работа с GPIO

Контакты МК логически объединяются в группы – «порты».

В stm32f10x в каждом порту 16 контактов.

Порты обозначаются буквами – PORTA, PORTB, PORTC...

Контакты обозначаются числами от 0 до 15:

PC.12 – 12-й контакт в порту C.

Количество **доступных** контактов зависит от корпуса МК; некоторые порты могут отсутствовать целиком или частично.



# STM32 VL Discovery

Два светодиода, подключенные к земле и МК:

- PC.8
- PC.9

Две кнопки:

- Черная – это reset
- Синяя – PA.0 – просто кнопка с внешней подтяжкой к питанию

# Регистры GPIO

Регистр CRL (control low) – режим работы пинов с 0 по 7.

Регистр CRH (control high) – режим работы пинов с 8 по 15.

Регистр IDR (input data register) – чтение состояния входов.

Регистр ODR (output data register) – чтение и запись состояния выходов.

И еще несколько.

У каждого порта есть такой набор регистров.

Где про них читать? reference manual, глава 9.2 (стр. 166)

# Доступ к регистрам периферии в языке C

Через указатели на «волшебные» структуры:

GPIOA->ODR – доступ к регистру ODR порта A, словно это обычная глобальная переменная.

(в других МК бывают «волшебные» указатели сразу на регистры, без структур)

# Почему ничего не работает?!

Практически всю периферию в МК нужно сначала включить (подать питание и тактирование).

Это нужно сделать через регистры подсистемы тактирования. Все GPIO включаются через регистр RCC->APB2ENR (ref. man. стр. 142)

# Как же зажечь светодиод

1. Подать питание на нужный порт
  - регистр RCC->AP2ENR
1. Настроить режим нужного контакта в нужном порту (нужен режим output push pull)
  - регистр GPIOx->CRH или CRL
1. Вывести на контакт высокий уровень
  - регистр GPIOx->ODR

# Битовые манипуляции

- Установка одного бита:
  - `a |= 1<<7; // установить седьмой бит`
- Сброс одного бита:
  - `a &= ~(1<<3); // сбросить третий бит`
- Инверсия одного бита:
  - `a ^= 1<<5; // инверсия пятого бита`

# Доступ к регистрам, отображенным на память

Допустим, адрес нужного мне регистра - 0x4001 0800.

И регистр этот размером в 4 байта.

Как мне в него что-нибудь записать, если я пишу на C?

Например, можно создать указатель:

```
uint32_t * ptr = 0x40010800;
```

Но указателю нельзя присвоить число, будет ошибка компиляции.

Нужно сделать *приведение типа*:

```
uint32_t * ptr = (uint32_t *)0x40010800;
```

Теперь почти все ок, можно записать число по нужному адресу:

```
*ptr = 1;
```

# Доступ к регистрам, отображенным на память

А как сделать то же самое, не создавая указатель?

```
*((volatile uint32_t *) (0x40010800)) = 1
```

Ужас какой. И что, каждый раз вот так писать?

Можно и так.

Но как правило, этот ужас прячут за #define:

```
#define REGISTER *((volatile uint32_t *)0x40010800)
```

И потом можно писать так, словно REGISTER – это заранее созданная глобальная переменная:

```
REGISTER = 5;
```

# Доступ к регистрам, отображенным на память

В последние годы, дефайнить каждый регистр – это уже не модно.

Теперь модно объявить структуру из нескольких регистров, относящихся к одному периферийному устройству.

А потом задефайнить адрес с приведением типа к этой структуре.

Тогда доступ к регистрам выглядит как доступ к *глобальному указателю на структуру*:

```
GPIOC->ODR = 1;
```