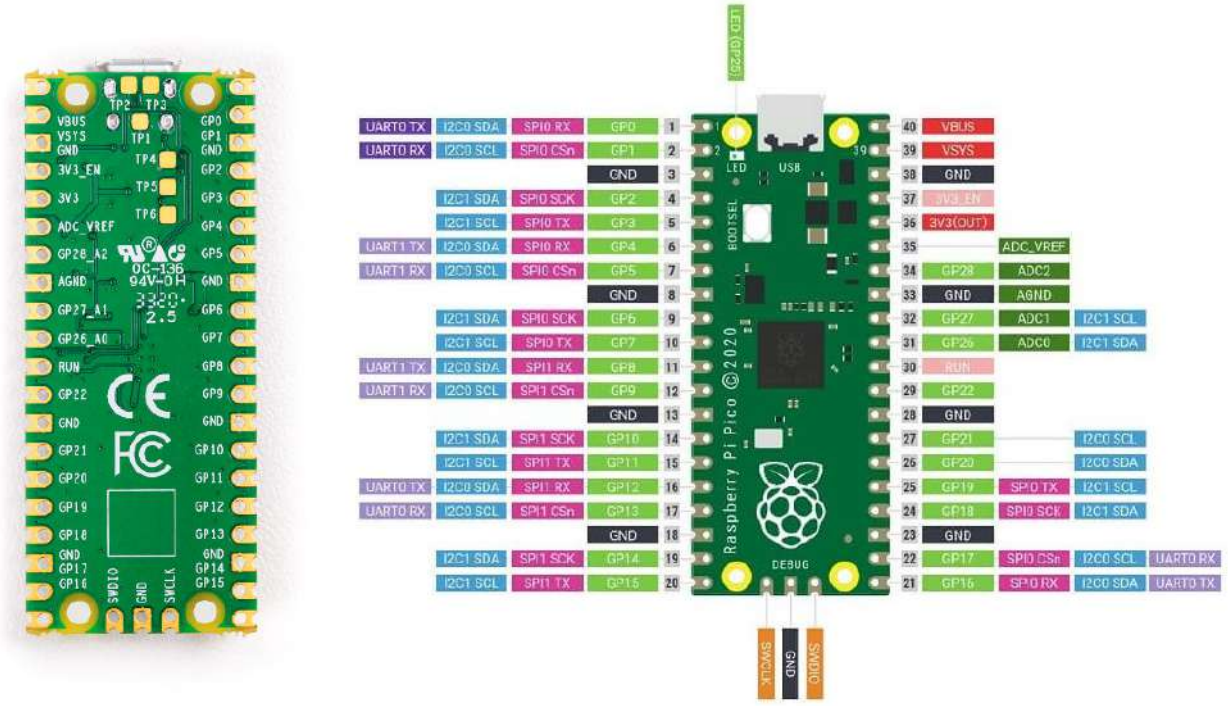
Второе практическое занятие v.2025.1

Пины GPIO

Схема GRIO представлена ниже:



Компактные размеры RP Pico не позволили подписать пины на лицевой стороне, что заставит нас постоянно обращаться к схеме распиновки.

> **ВАЖНО** отметить, что номера пинов указаны в зеленых квадратах - так пин 20 на самом деле 15, а 34 - 28.

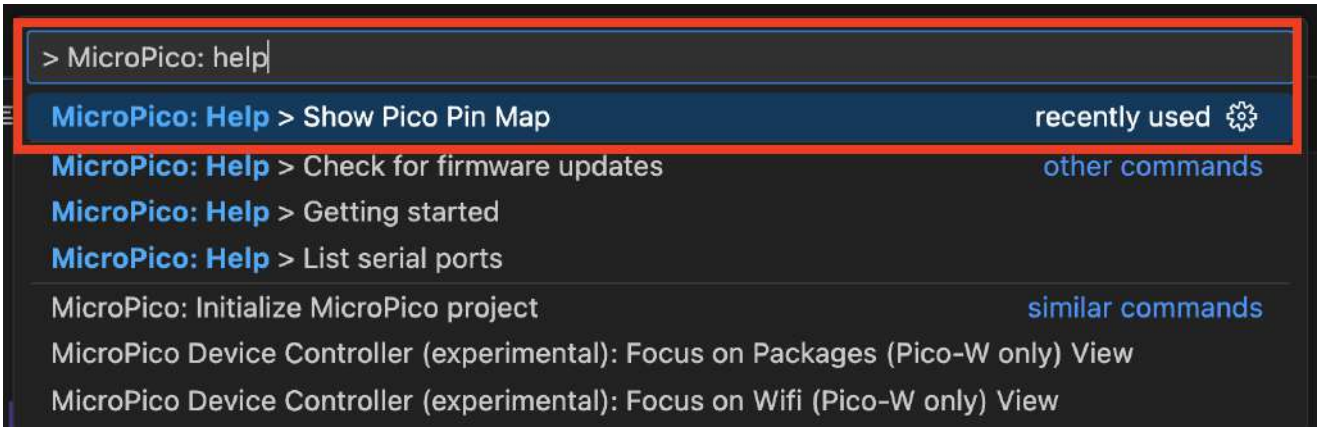
Вывод схемы распиновки в VS Code

В модуле `MicroPico` реализована возможность вывода схемы распиновки.

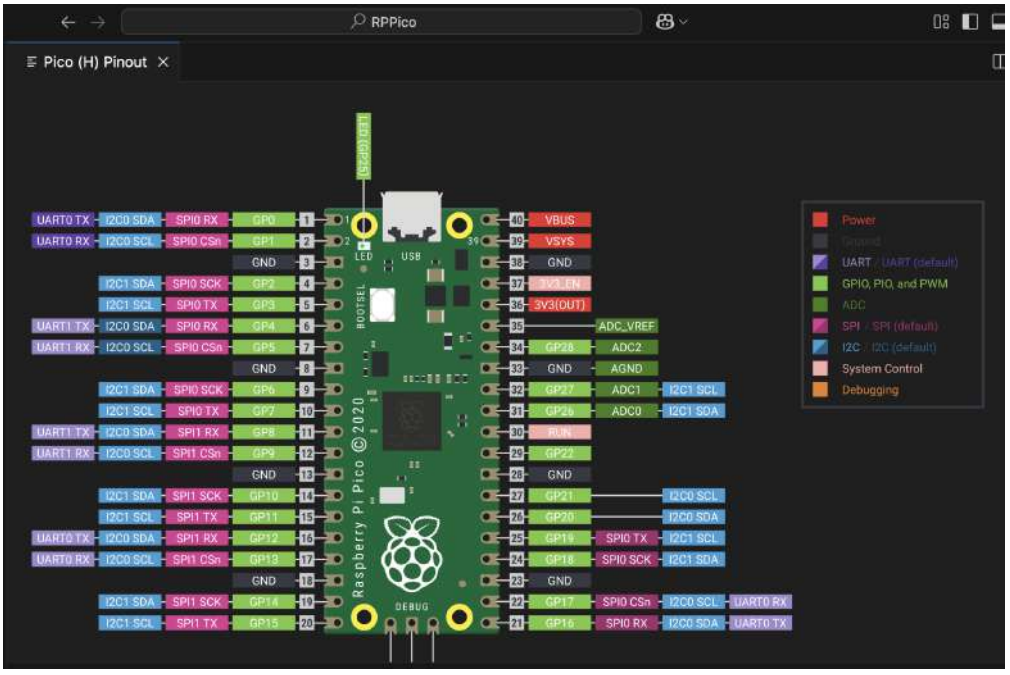
Для этого:

- Вызовите полный список команд (кнопкой в нижней панели или комбинацией `Ctrl + Shift + P`):

- Выберете команду:



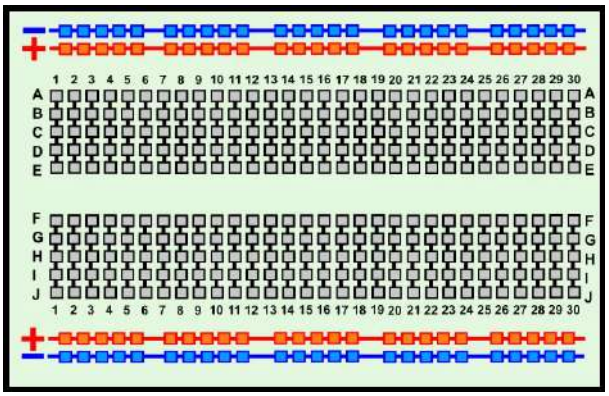
- Тип микроконтроллера:

- Схемаоткроется в новой вкладке:
- 

Breadboard

Подключаться напрямую к пинам GPIO возможно, но неудобно, так это требует "навесного монтажа" электронных компонентов. Для ускорения прототипирования электронных устройств используются макетные платы, сохранившие классическое название breadboard.

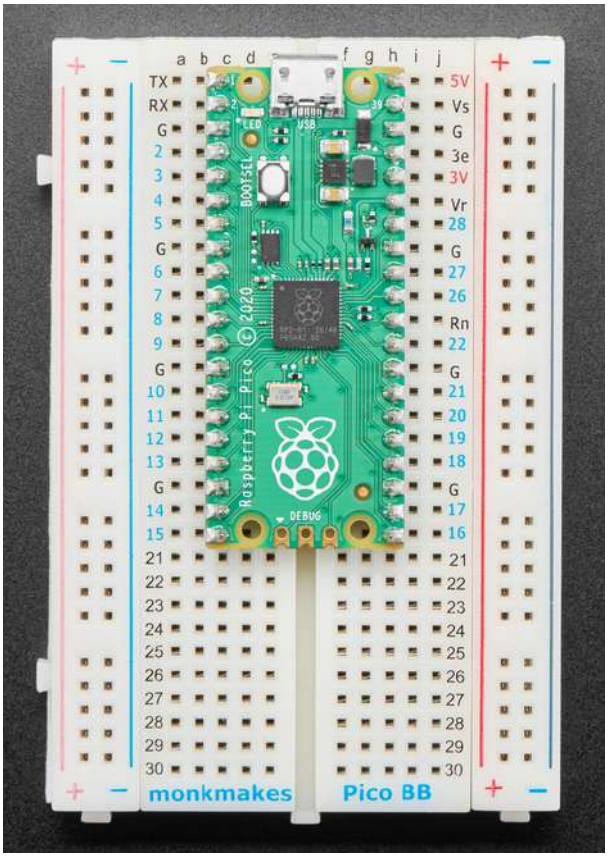
Принцип устройства макетной платы изображен на следующем рисунке:



Центральные ряды отверстий соединенны между собой, так, например, отверстия A1-B1-C1-D1-E1 - имеют общее подключения, а E1-F1 или A1-A2 - нет.

Шины "+" и "-" имеют общее подключение по всей длине макетной платы и используются для питания и соединения компонентов.

Общий вид RP Pico на макетной плате изображен на рисунке:



Что позволяет подключиться к любому пину.

Так 15 пин (нижний левый) имеет общий контакт с отверстиями a15 и b15.

Подключение внешнего светодиода

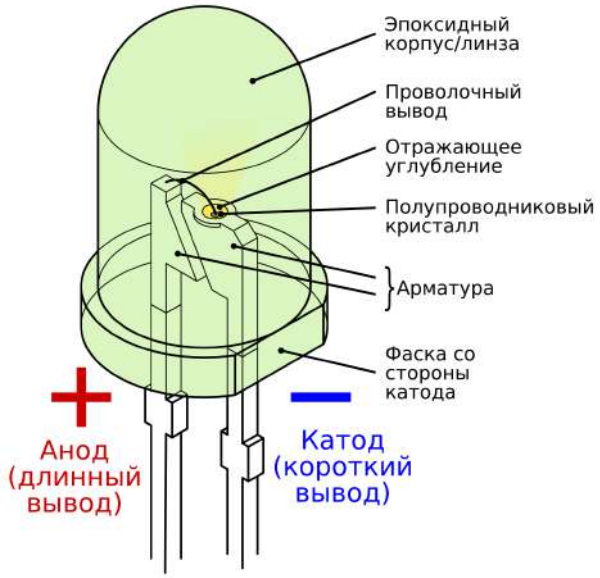
При работе со светодиодами необходимо помнить 2 важных вещи:

1. Диод пропускает ток и светится только в одном направлении.
2. Диод имеет почти нулевое сопротивление электрическому току.

Разберем внимательнее эти условия.

Полярность диода

Чтобы не запутаться с полярностью диода необходимо знать, что все они имеют



стандартный вид:

Тут все просто - короткая нога должно подключаться к земле (GND), а на длинную - подаваться напряжение. Если после пайки лишняя длинна ног была обрезана к земле подключается тот вывод со стороны которого спилена фаска.

Если перепутать -- ничего страшного не произойдет - диод просто не будет светиться.

Сопротивление диода

Второе условие более серьезное:

⚠ ВАЖНО! Если подключить диод без дополнительного сопротивления это приведет к короткому замыканию!

Для лучшего понимания этого вспомним закон Ома:

$$I = \frac{U}{R}$$

Сопротивление R у диода стремится к 0, а значит сила тока I в цепи без дополнительного сопротивления станет стремиться к бесконечности, что, в лучшем случае спалит светодиод, а в худшем - сам микроконтроллер!

> КОРОЧЕ! ВСЕГДА ПОДКЛЮЧАЙТЕ ДИОД ПОСЛЕДОВАТЛЬНО С СОПРОТИВЛЕНИЕМ!

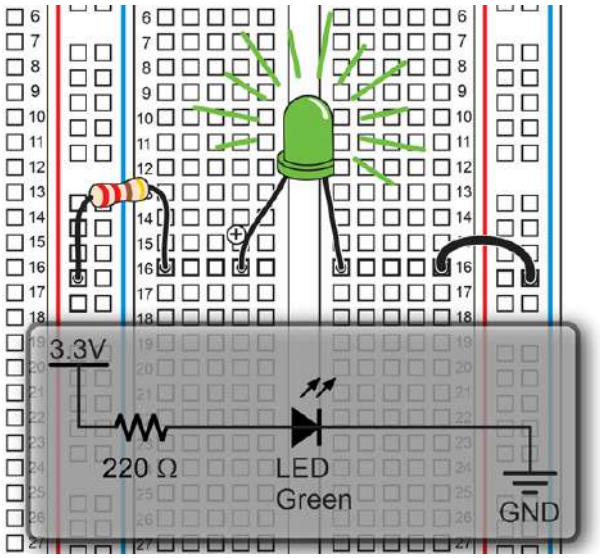
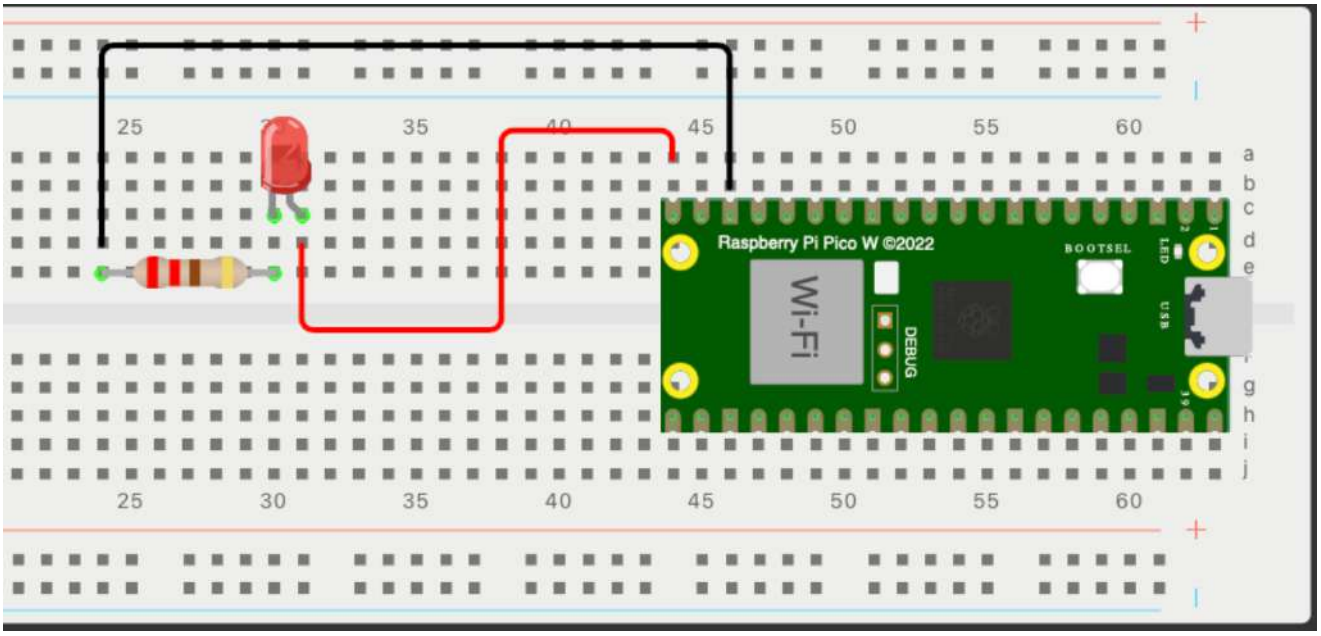


Схема подключение внешнего светодиода

Одна из возможных схем подключения представлена на рисунке:



Код программы

Пример кода для указанной схемы подключения:

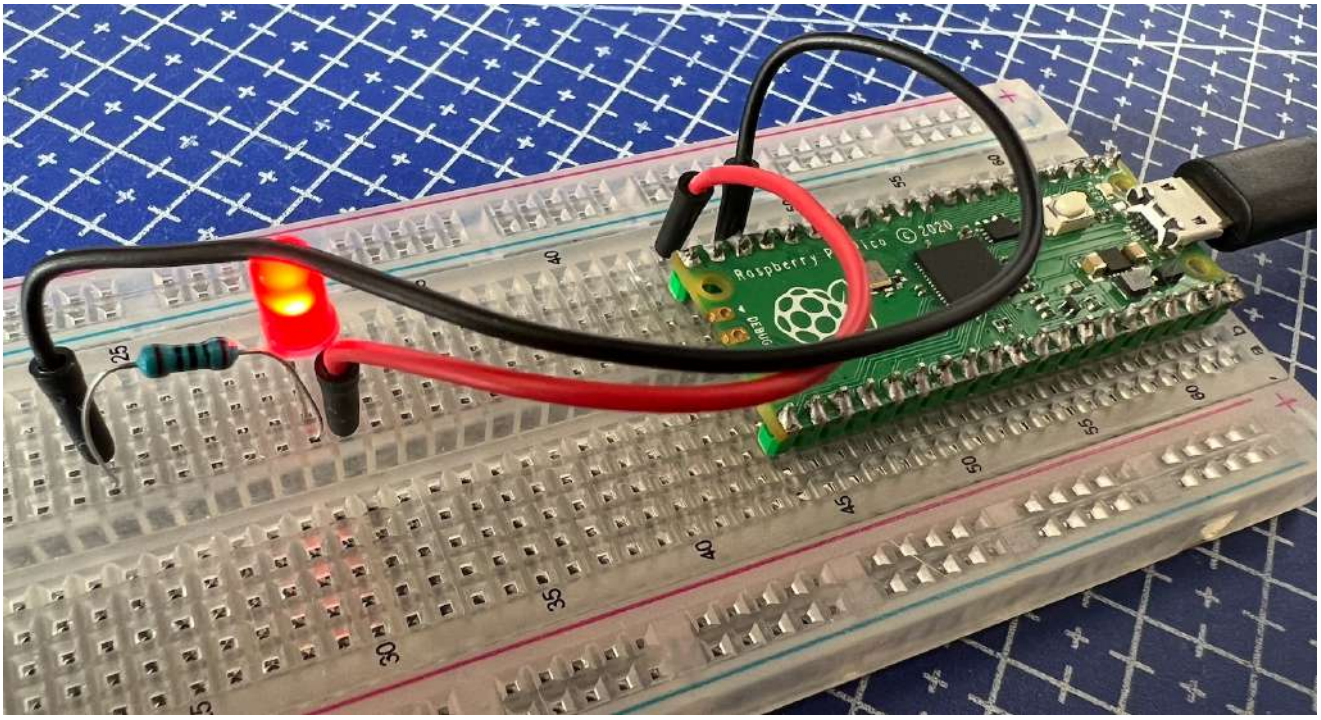
```
from machine import Pin
import time

led_pin = Pin(15, Pin.OUT)

while True:
    led_pin.on()
    time.sleep(3)
    led_pin.off()
    time.sleep(1)
```

В результате диод должен гореть три секунды, гаснуть на одну и повторять этот цикл снова и снова...

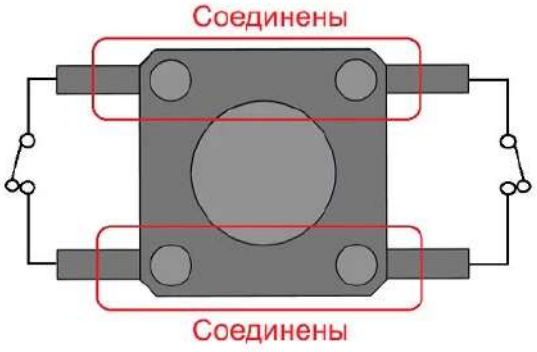
Все это в аналоговом мире будет выглядеть как-то так:



Работа с тактовой кнопкой

Разберем подробнее получение цифрового сигнала на примере обработки сигнала с тактовой кнопки:

Принципиальная схема работы тактовой кнопки показана на рисунке:



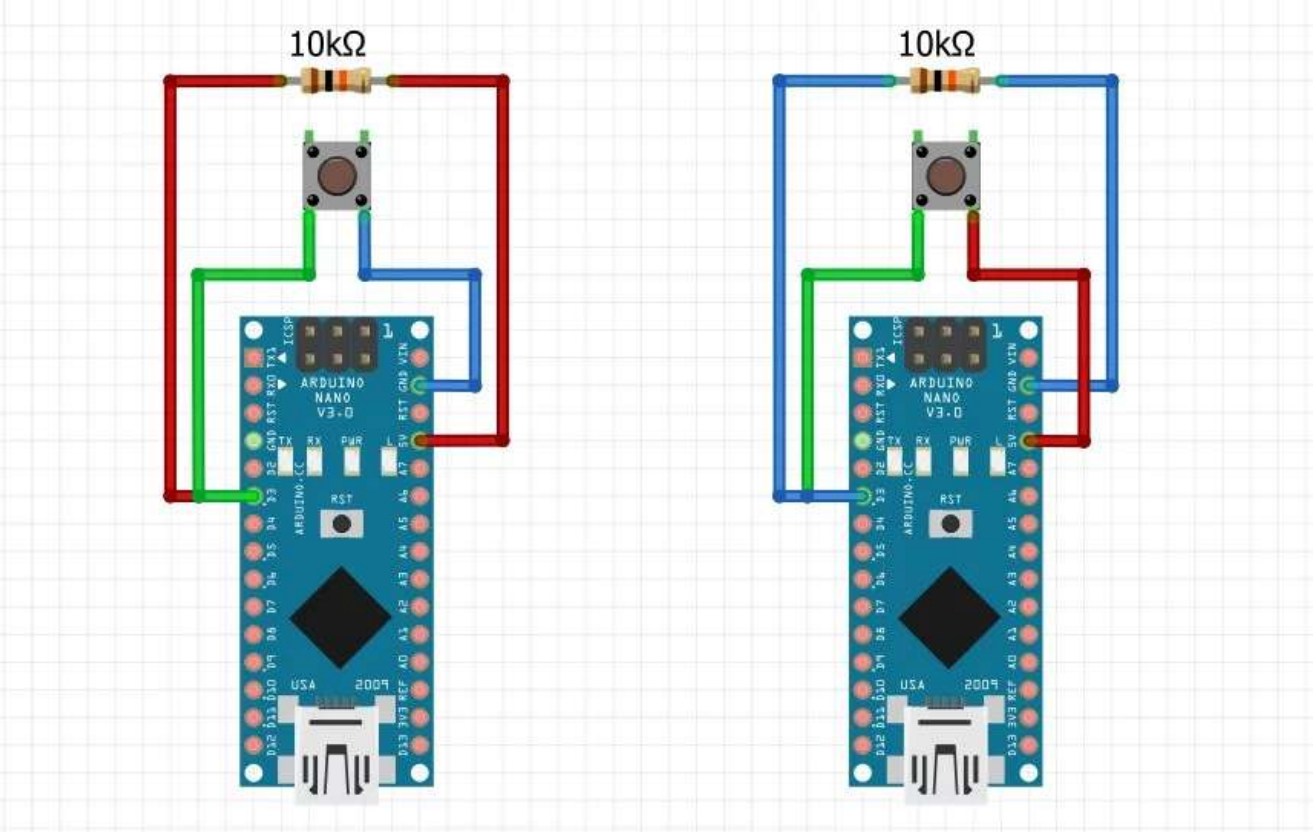
Как видно из схемы - противоположные контакты кнопки замкнуты постоянно, а контакты на одной стороне замыкаются при нажатии.

> ⚠ ВАЖНО! Как и в случае со светодиодом - кнопка сама по себе не создает сопротивления и ее подключение без дополнительного резистора приведет к короткому замыканию и перегоранию платы!

Схемы подключения с "подтяжкой" внешними резисторами

Проблема чтения внешнего сигнала, приходящего на пины, усложняется тем, что провода и прочее подключаемое оборудование может (и будет) работать как антенны и принимать помимо полезного сигнала внешние помехи и шумы.

Решить это могут следующие схемы подключения включающие в себя "подтяжку" пинов через внешнее сопротивление:



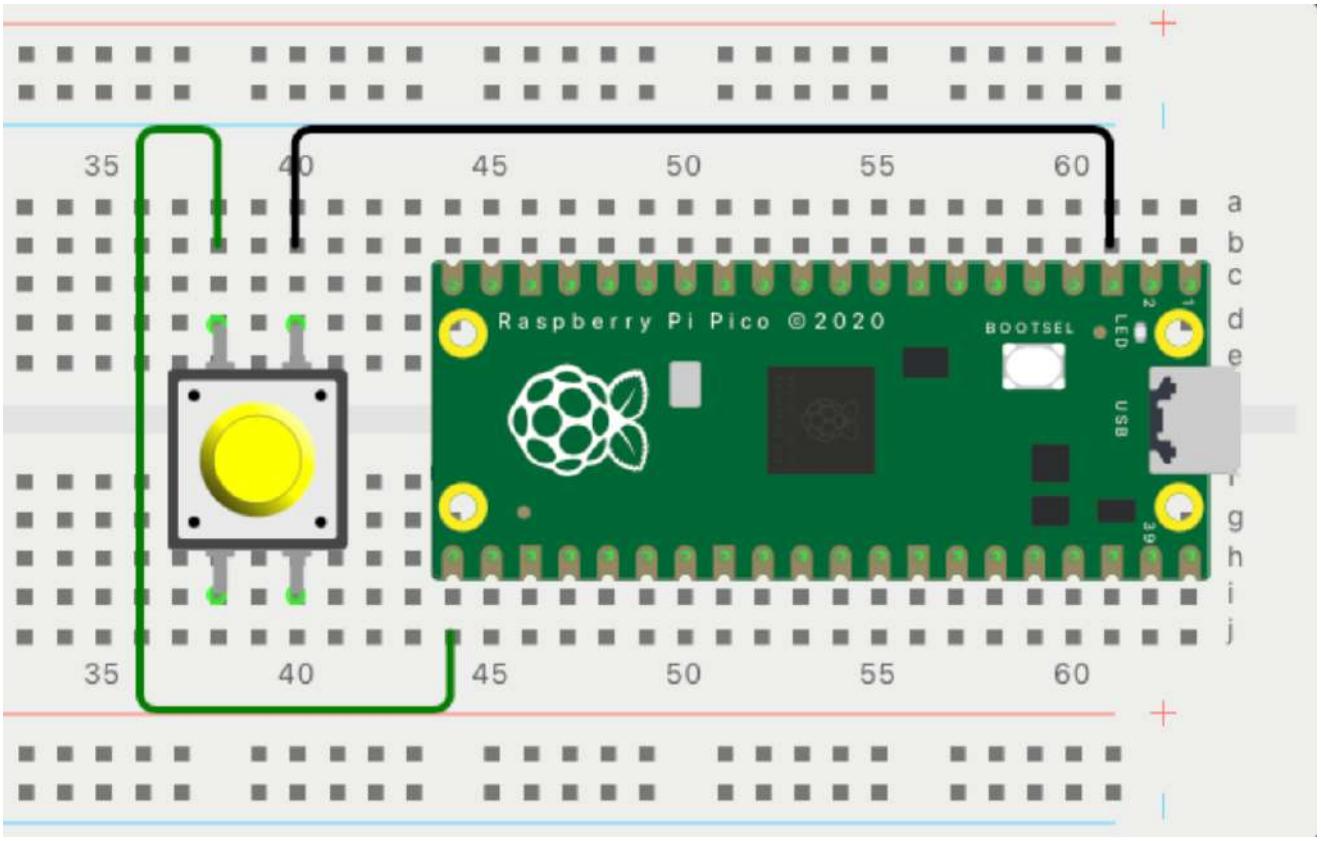
Подтяжка выполняется противоположно принимаемому сигналу, т.е. если нужно ловить высокий сигнал, подтяжка выполняется к земле, если ловить нужно сигнал земли - подтяжка выполняется к питанию.

Как выбирается сопротивление резистора? Тут всё очень просто: при нажатии на кнопку через резистор потечёт ток, так как в любом случае замыкается цепь питание-земля. Чем выше ток, больше потери энергии и нагрев резистора, а это никому не нужно, поэтому сопротивление резистора подтяжки обычно выбирается в диапазоне 5-50 $k\Omega$. Если ставить больше - подтяжка может не обеспечить стабильный уровень

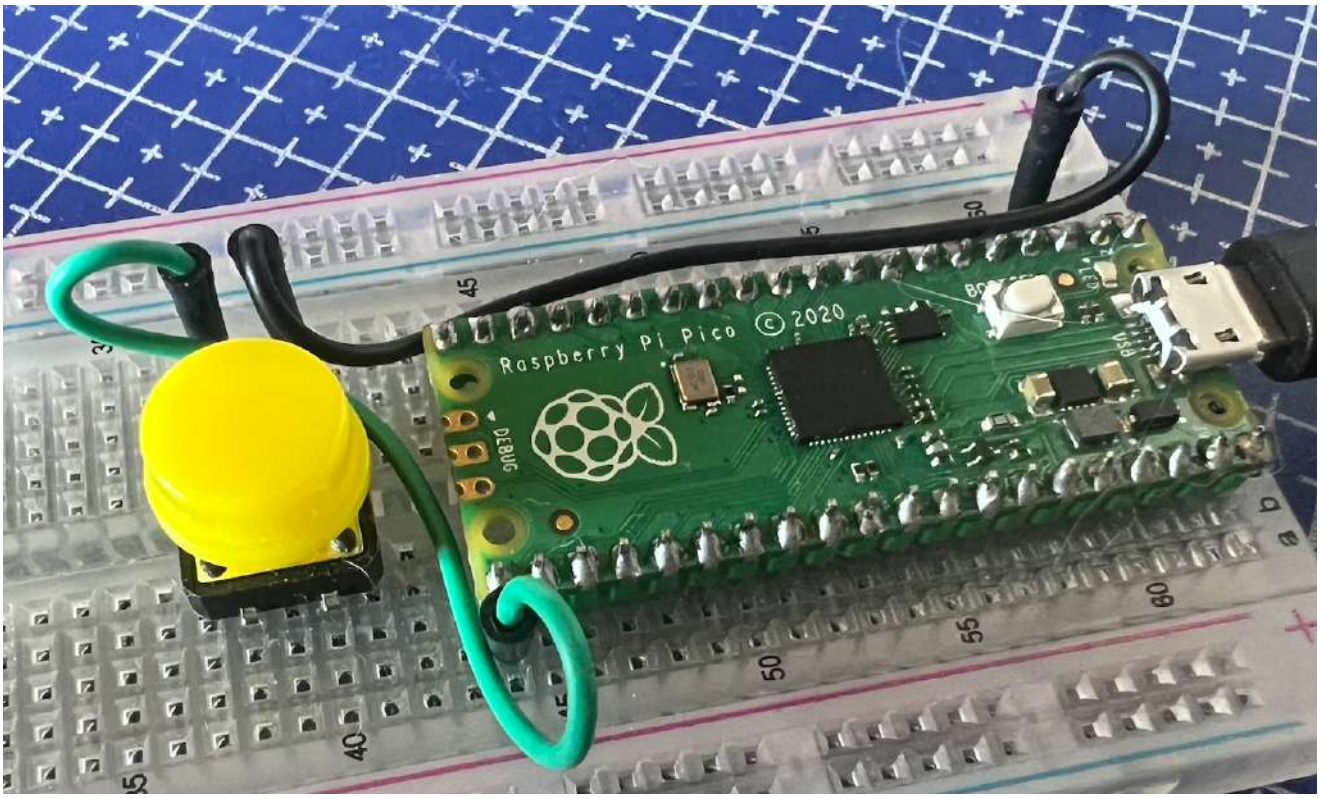
сигнала на пине, а если ставить меньше - будут больше потери энергии в нагрев резистора: при сопротивлении в 1 $k\Omega$ через него потечёт ток величиной 5 V/1000 Ω = 5 мА, для сравнения плата Ардуино с МК в активном режиме потребляет 20-22 мА. Чаще всего для подтяжки используется резистор на 10 $k\Omega$.

Правильное подключение кнопки!

В плате RP Pico есть встроенные "подтягивающие" резисторы позволяющие, при их программном подключении, добиться схемы изображенной выше без внешнего сопротивления.



Или в реальности:



⚠ ПРИ ПОДКЛЮЧЕНИИ КНОПКИ НЕОБХОДИМО ДОПОЛНИТЕЛЬНО ВКЛЮЧИТЬ РЕЖИМ "ПОДТЯЖКИ НА ПИНЕ К КОТОРОМУ ПОДКЛЮЧАЕТСЯ КНОПКА!

Следующий код будет считывать значение кнопки:

```
from machine import Pin
import time

BUTTON_PIN = 16
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)

counter = 0
while True:
    button_value = button.value()
    print(counter, "\tbutton_state", button_value)
    time.sleep(0.2)
    counter += 1
```

Остановимся на двух важных моментах заключающихся в этой части кода:

```
BUTTON_PIN = 16
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)
```

Первое изменение стилистическое - мы инициировали новую константу `BUTTON_PIN` и передали в нее номер конкретного пина. Далее уже именно эта переменная, а не номер пина используется при создании объекта пина для кнопки. Такой подход

позволяет сделать код более гибким и вынести переменные с подключаемыми пинами за пределы основной логики работы программы.

Второе изменение более важное, в контексте решаемой задачи:

```
Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)
```

Мы "включили" на создаваемом пине подтягивающий резистор добавлением параметра "`Pin.PULL_UP`".

Так как на схеме подключения мы подключили кнопку к земле ("`GND`") этим резистором мы будем "подтягивать" сигнал к высокому уровню, то есть "тянуть вверх" (`PULL_UP`).

Проще все это принять и запомнить...

⚠ ВАЖНО! При подключении кнопки напрямую необходимо включать подтягивающий резистор при создании объекта пина! (`Pin(button_pin, Pin.IN, Pin.PULL_UP)`)

В итоге при запуске программы мы можем видеть следующий результат:

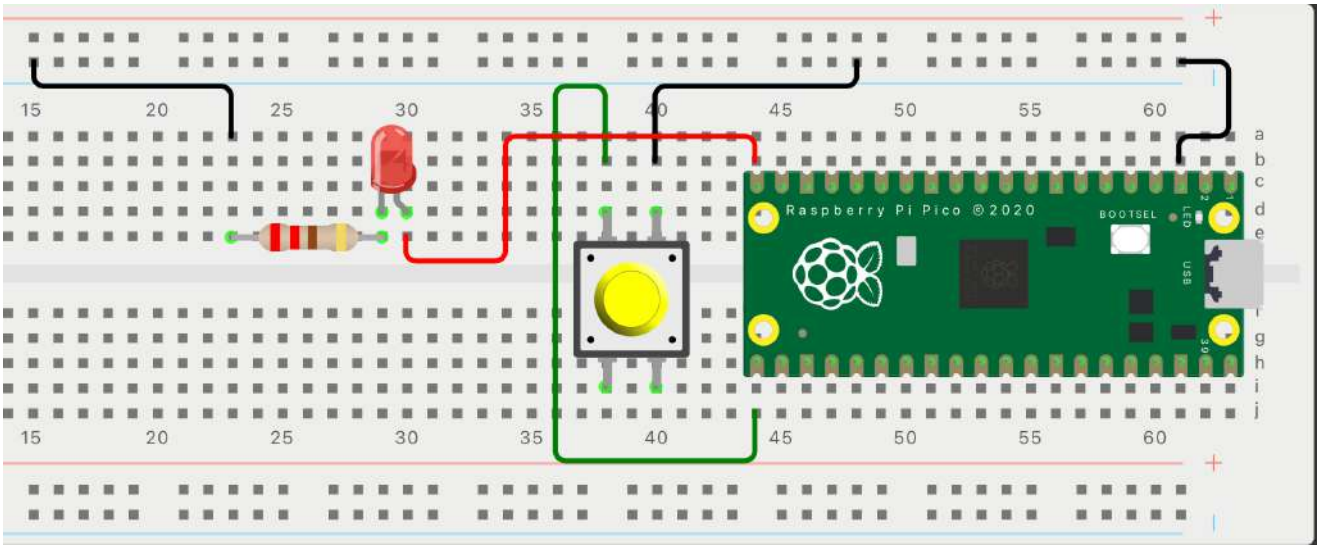
> Так как мы подтянули пин к высокому сигналу в разомкнутой цепи (когда кнопка отпущена) на нем находится высокий сигнал - логическая единица. При нажатии кнопки напряжение "уходит" в землю и пин принимает логический 0. Это необходимо учитывать при разработке алгоритмов использующих кнопку!

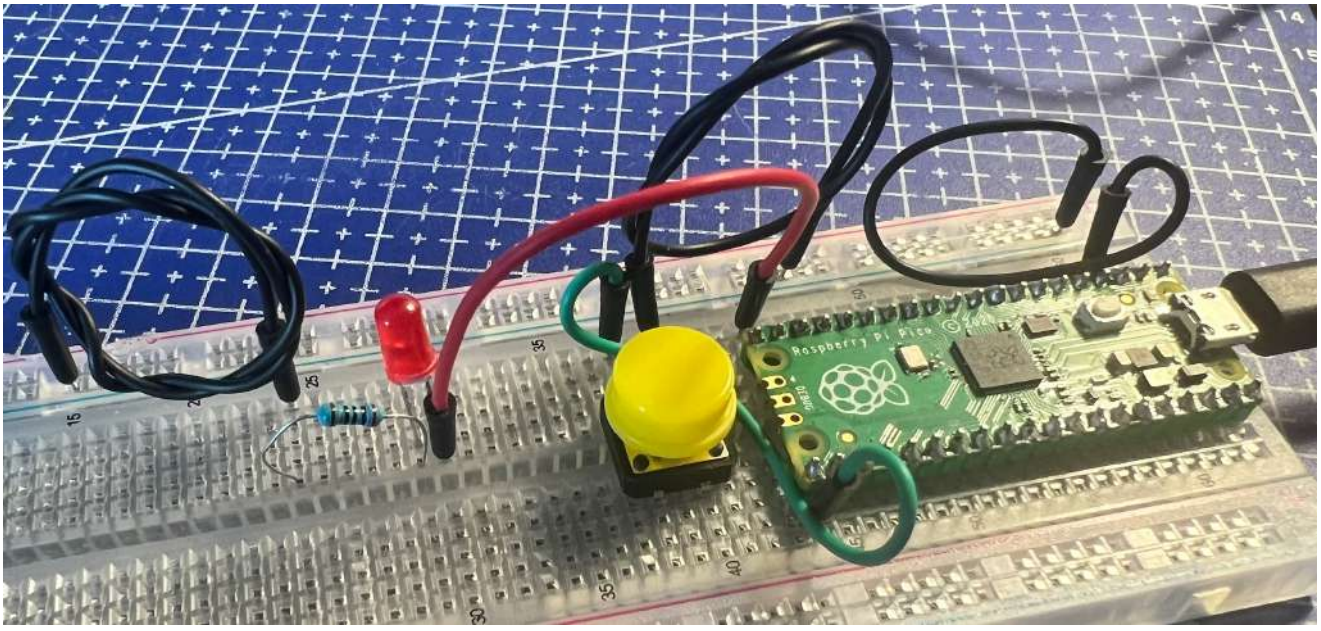
Создание схемы с диодом и кнопкой

Объединим полученные знания о вводе и выводе цифрового сигнала, создав устройство включающие и выключающее светодиод с помощью нажатия кнопки!

Схема подключения

Объединим две ранее изученные схемы в одну:





⚠ Еще раз отметим важность включения в схему резистора при подключении светодиода!

И попробуем заставить все это работать!

Буквальное решение задачи

Если мы попытаемся аккуратно объединить код предыдущих примеров мы получим что-то вроде этого:

```
from machine import Pin
import time

RED_LED_PIN = 15
BUTTON_PIN = 16

red_led = Pin(RED_LED_PIN, Pin.OUT)
```

```
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)

while True:
    button_value = button.value()
    red_led.value(button_value)
```

⚠ Еще раз отметим важность включения подтягивающего резистора для пина кнопки!

Запустив код мы увидим, что в целом задача решена - работа диода зависит от состояния кнопки. При этом эти два устройства не имеют между собой явной связи и все их взаимодействие осуществляется посредством команд микроконтроллера!

Однако перейдем к минусам:

- С отпущенной кнопкой диод постоянно горит и потухает только при нажатии кнопки;
- Чтобы выключить диод, необходимо постоянно держать кнопку нажатой.

Оба эти обстоятельства контр интуитивны.

> С первым попробуйте справиться самостоятельно!

Со вторым справимся вместе программно!

Обработка нажатия кнопки

Внесем исправления в предыдущую версию кода и получим в итоге:

```
from machine import Pin
import time

RED_LED_PIN = 15
BUTTON_PIN = 16

red_led = Pin(RED_LED_PIN, Pin.OUT)
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)

led_state = False
while True:
    red_led.value(led_state)
    button_value = button.value()

    if button_value == 0:
        led_state = not led_state

    time.sleep(0.2)
```

Введем новую переменную `led_state` и инициируем ее стартовым булевым значением `False`.

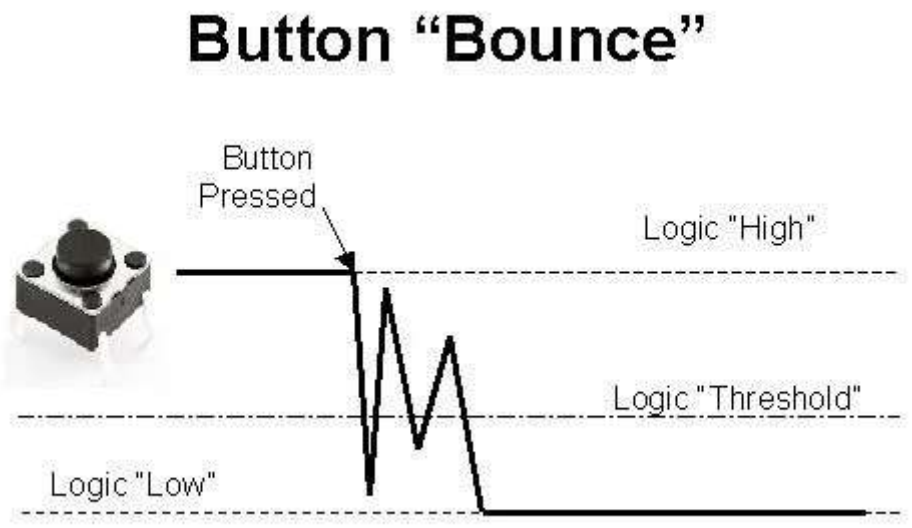
> Булевые значения `False` и `True`, являясь логическим нулем и единицей соответственно могут передаваться в качестве `0` или `1` на ввод пина.

Далее в цикле:

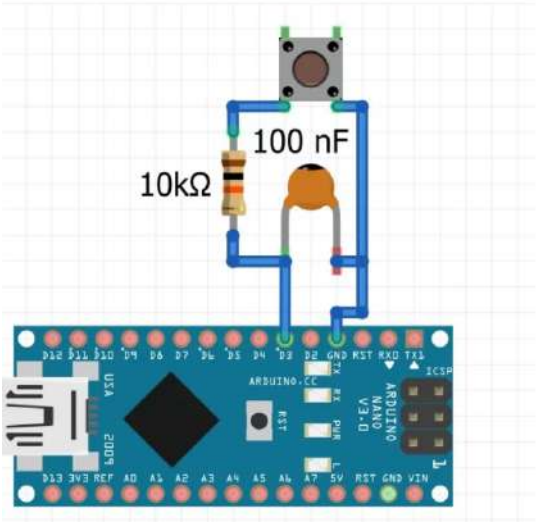
- На диод передается его "состояние" из переменной `led_state`. Если состояние диода совпадает с состоянием пина, то изменений в его состоянии не произойдет, в противном случае он переключится.
- `button_value = button.value()` - читаем значение с пина, подключенного к кнопке;
- Проверяем тождественно ли полученное значение `0` (состоянию с нажатой кнопкой);
- В случае выполнения предыдущего условия - состояние булевой переменной `led_state` меняется на противоположное (`led_state = not led_state` - в переменной присваивается "отрицание" текущего значения);
- Выполняется задержка в 0.2 секунды для устранения "дребезга кнопки".

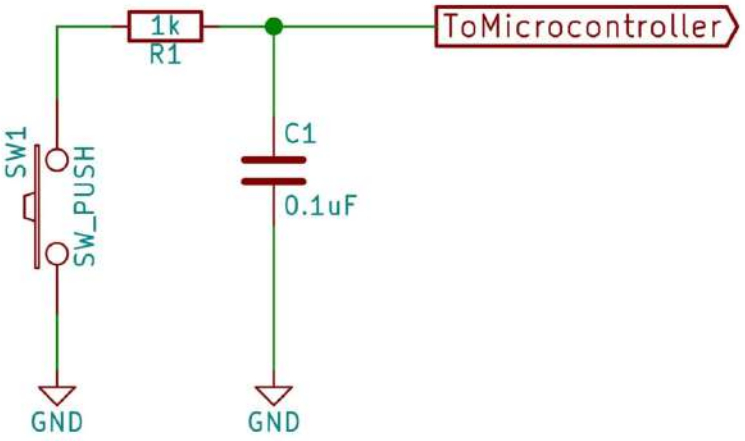
Дребезг кнопки

Кнопка не идеальна, и контакт замыкается не сразу, какое-то время он "дребезжит". Прогоняя данный алгоритм, система опрашивает кнопку и условия приблизительно за 6 мкс, то есть кнопка опрашивается 166'666 раз в секунду! Этого достаточно, чтобы получить несколько тысяч ложных срабатываний.



Избавиться от дребезга контактов можно как аппаратно, так и программно: аппаратно задача решается при помощи RC цепи, то есть резистора (~1-10k) и конденсатора (~100nF). Выглядит это следующим образом:





В прошлом разделе мы устранили эту проблему программно введя искусственную задержку:

```
time.sleep(0.2)
```

Величина необходимой задержки напрямую зависит от качества самой кнопки и обычно находится в пределах от 0.05 до 0.2 секунд и может быть подобрано экспериментально.

Тем не менее задержка выполняемая с "усыплением" микроконтроллера это очень плохая идея, так как в это время прекратятся и остальные выполняемые операции, поэтому лучше оптимизировать предыдущее решение:

```
from machine import Pin
import time

RED_LED_PIN = 15
BUTTON_PIN = 16

BOUNCE_TIME = 250

red_led = Pin(RED_LED_PIN, Pin.OUT)
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)

led_state = False
timer_0 = time.ticks_ms()
while True:
    red_led.value(led_state)
    button_value = button.value()
    delta_time = time.ticks_ms() - timer_0
    if button_value == 0 and delta_time >= BOUNCE_TIME:
        led_state = not led_state
        timer_0 = time.ticks_ms()
```

В этом решении мы инициировали переменную `bounce_time` со значением в 250. Под этим значением будем предполагать интервал в 250 миллисекунд (0.25 секунды) в течении которого может происходить "дребезг" кнопки.

Далее инициируем переменную `timer_0` текущим значением миллисекунд прошедших с момента запуска микроконтроллера. Получить необходимое значение можно с помощью функции `time.ticks_ms()` модуля `time`.

Далее на каждой итерации цикла будем вычислять переменную `delta_time`, определять временной промежуток, прошедший с времени сохраненного в переменной `timer_0`.

После этого проверяется условие нажатия кнопки с одновременной проверкой того, что прошедшее время `delta_time` превышает необходимую паузу в `bounce_time`.

В случае одновременного выполнения обоих условий изменяется состояние переменной `led_state`, а переменной `timer_0` присваивается текущее время.