Восьмое практическое занятие

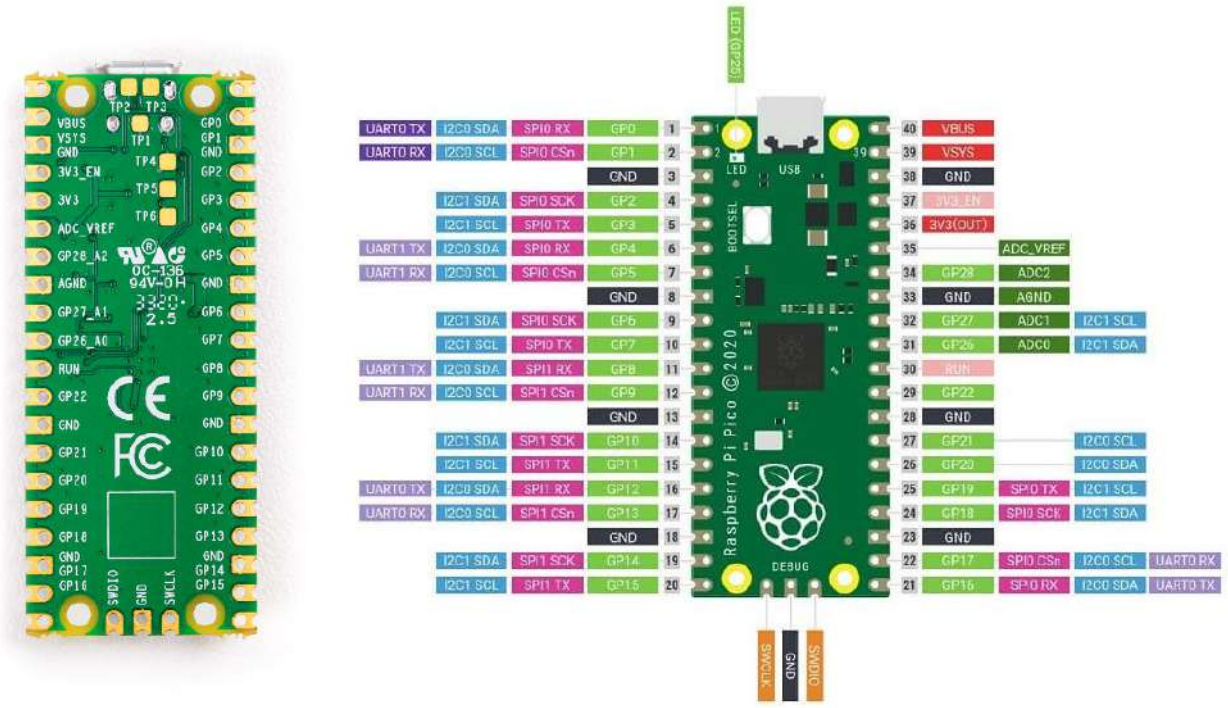
Новый семестр... Новые вызовы!

Чтобы не "оторваться" от земли и плавно войти в учебный процесс - сегодня немного вспомним, чуть-чуть обобщим и капельку продвинем вперед наши знания в области программирования микроконтроллеров!

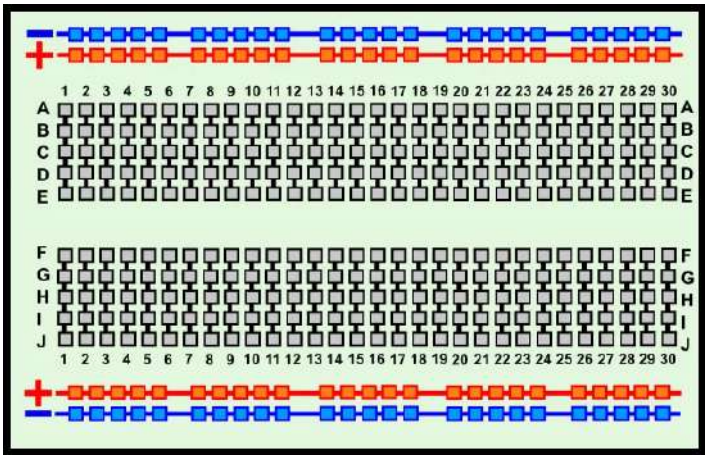
Озвучим базу:

База

- Мы работаем с микроконтроллером Rasspberry Pi Pico:



- У этой платы много выводов к которым мы подключаемся через "Брэдборд":



- Далее с помощью `micropython` (диалекта языка `python`) в IDE `PyCharm` мы можем по-разному подавать и измерять напряжение на различных "пинах" платы, управляя тем самым работой периферийных устройств!

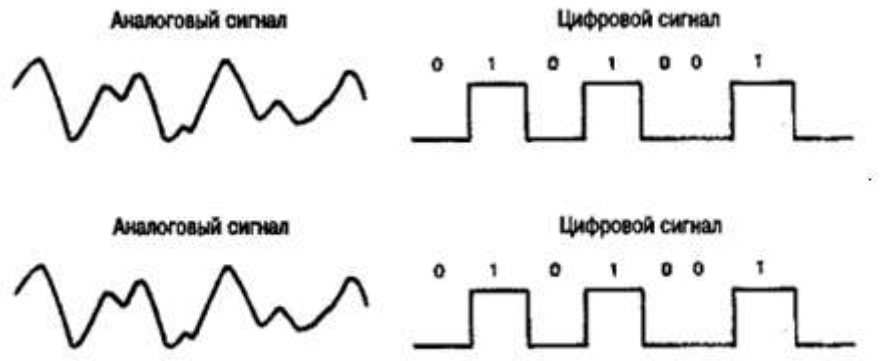
Все просто!

Вспомним типы сигналов

Все элементарно:

Сигналы бывают двух типов:

- Цифровой;
- Аналоговый.



Цифровой сигнал - может принимать только два значения:

- 1 (`HIGH`) - высокий сигнал, когда на пин е присутствует опорное напряжение (Для RP Pico это 3.3V)
- 0 (`LOW`) - низкий сигнал, логический ноль - когда напряжение на пине отсутствует.

Что происходит когда на пине напряжение между 0 и 3.3V? Сигнал "подтягивается" до ближайшего значения (как бы округляется в нужную сторону). За счет этого на цифровой сигнал в меньшей степени влияют помехи и "наводки" тем самым достигается более стабильная работа устройств.

Аналоговый сигнал - может принимать промежуточные значения между 0 и опорным напряжением.

С ним все несколько сложнее.

Так как мы все не любим когда "сложнее", то вспомним эту часть материала в конце занятия, а пока вернемся к цифровому сигналу!

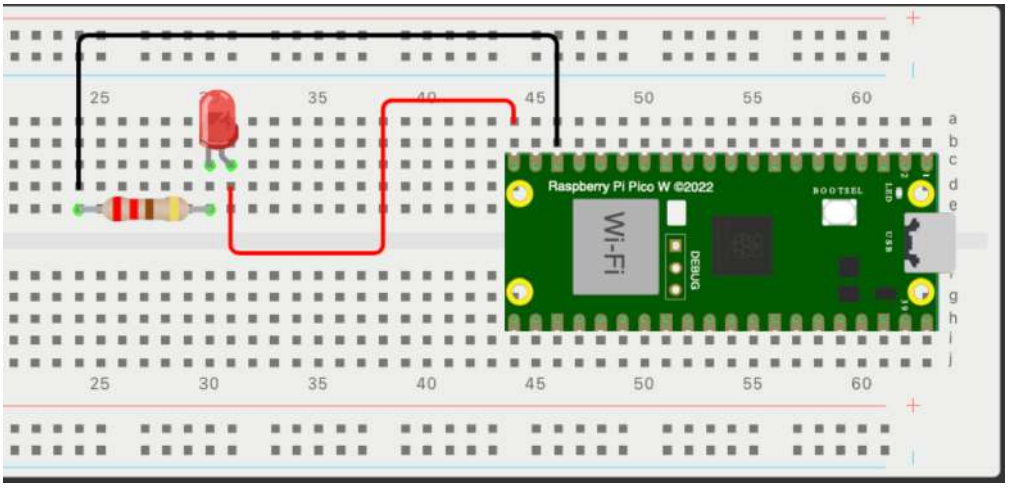
Цифровой сигнал

Любой сигнал мы можем передавать наружу (включать или выключать напряжение на пине) или измерять поступающее на пин напряжение.

Включение и выключение в нашей жизни встречается чаще и воспринимается проще - поэтому с него и начнем!

DigitalWrite

Классическим (и уже порядком приевшимся) примером этого класса сигнала является включение и выключение светодиода:



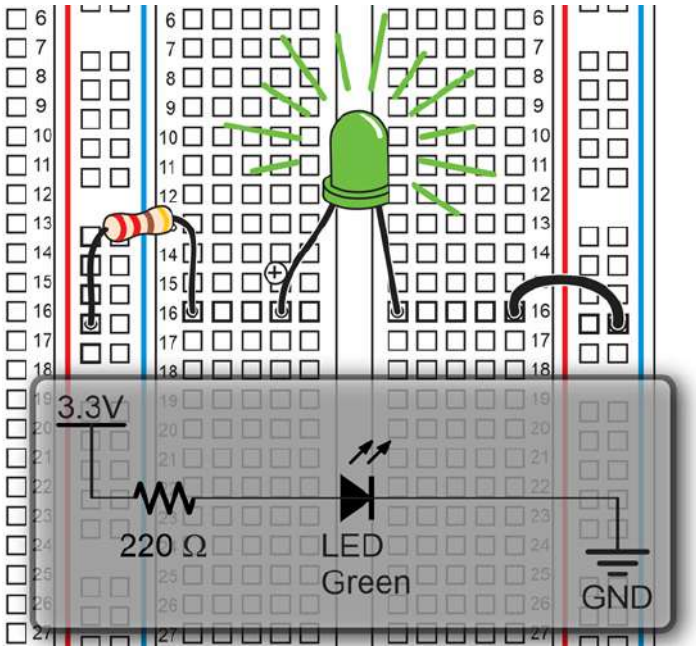
Напомним только, что так как светодиод не создает сопротивления в сети (да! Такой он ведомый и управляемый), нам необходимо **всегда** подключать последовательно с ним дополнительный резистор (сопротивление) на 220Ω.

При этом зададим себе риторический вопрос! Есть ли разница куда подключать провода к микроконтроллеру?

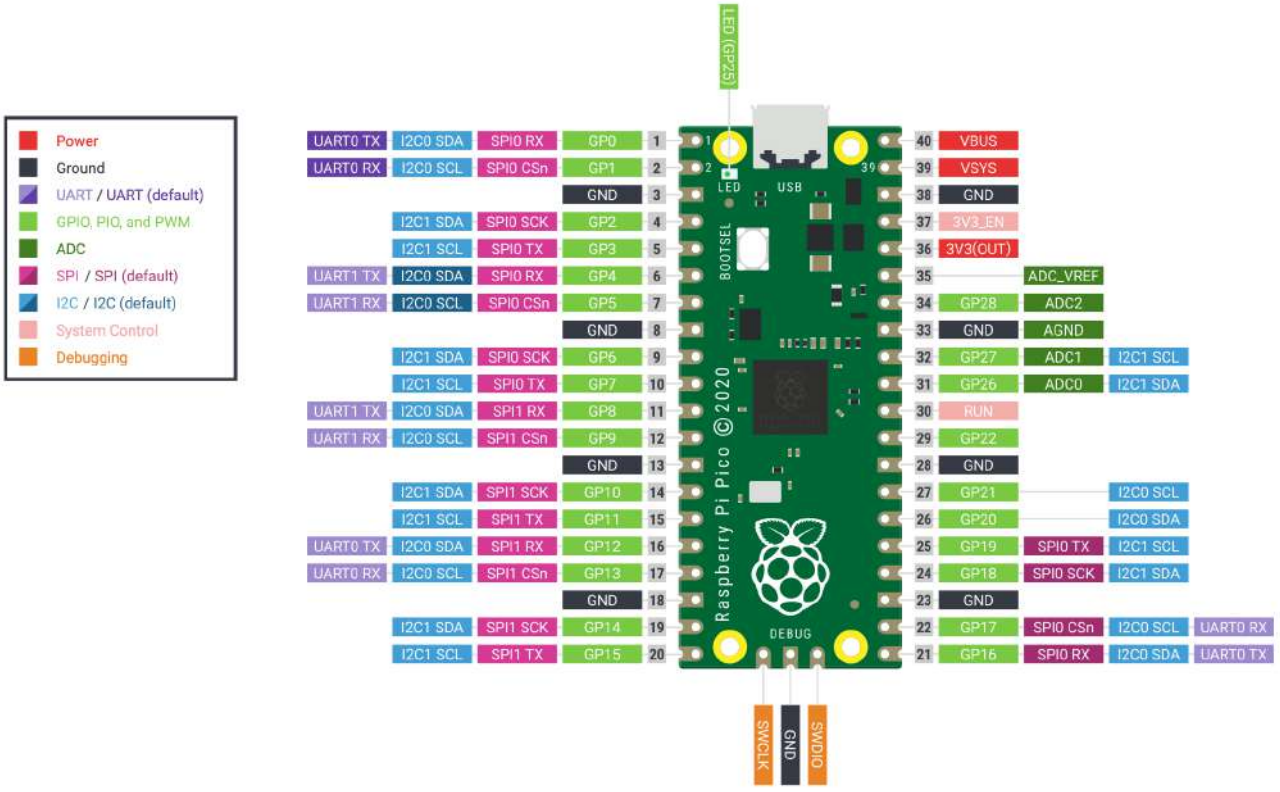
Надеюсь, что Ваш ответ уверенное: "**ДА**!".

Почему?

- Во-первых, светодиод будет гореть только если соблюсти верную полярность:



- Во-вторых, не все пины на плате микроконтроллера одинаковы:



Цифровой сигнал мы можем получить только со светло-зеленых "цифровых" пинов, а "заземлить" сигнал проще всего на черных пинах `GND`

Отметим так же, что у каждого пина есть свой номер и он подписан в индексе пина: GP0, GP1, ..., GP15, ..., Gp28 - это 0, 1, ..., 15, ... и 28 пин соответственно. Внимательный читатель может заметить, что некоторые из номеров пинов пропущены (23-25) и он окажется абсолютно прав! На самом деле эти пины на самом "камне" есть, просто они не выведены на периферийные пины и используются самой платой.

Ну а как же наконец зажечь на светодиод?

Можно примитивно:

```
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)
```

Но это уже как-то совсем туповато...

Так что сделаем умновато:

```
from time import sleep
from machine import Pin

class Led:
    def __init__(self, led_pin):
        self.led = Pin(led_pin, Pin.OUT)

    def on(self):
        self.led.on()

    def off(self):
        self.led.off()


LED_PIN = 15
led = Led(led_pin=LED_PIN)

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

```
from time import sleep
from machine import Pin

class Led:
    def __init__(self, led_pin):
        self.led = Pin(led_pin, Pin.OUT)
        self.led_state = None
```

```
    def on(self):
        self.led_state = "ON"
        self.led.on()

    def off(self):
        self.led_state = "OFF"
        self.led.off()

    def switch(self):
        if self.led_state == "ON":
            self.off()
        else:
            self.on()


LED_PIN = 15
led = Led(led_pin=LED_PIN)

while True:
    led.switch()
    sleep(2)
```

Что мы сделали?

Использовали пугающее и непонятное ООП...

Зачем? Чтобы вызвать у всех окружающих чувство зависти к уровню нашей крутости и знаний? Нееееет... (Ну или совсем чуть-чуть...) Затем, чтобы разделить логику управления устройством (пускай и таким простым как светодиод) от интерфейса управления им.

Не понятно? Не постесняйся задать вопрос на занятии! Стесняешься? Не страшно! Перечитай материалы к предыдущим занятиям, а потом задавай вопрос! Всегда задавай вопросы, если тебе не понятно!

Продолжим двигаться вперед!

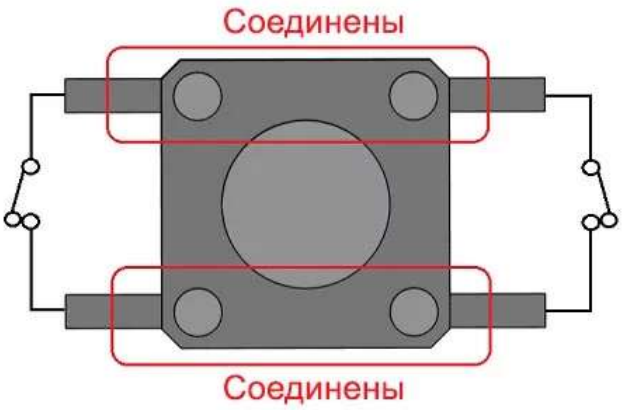
Мы много раз уже "записывали" сигнал, но, вероятно, могли подзабыть как его "читать"...

Исправим это!

DigitalRead

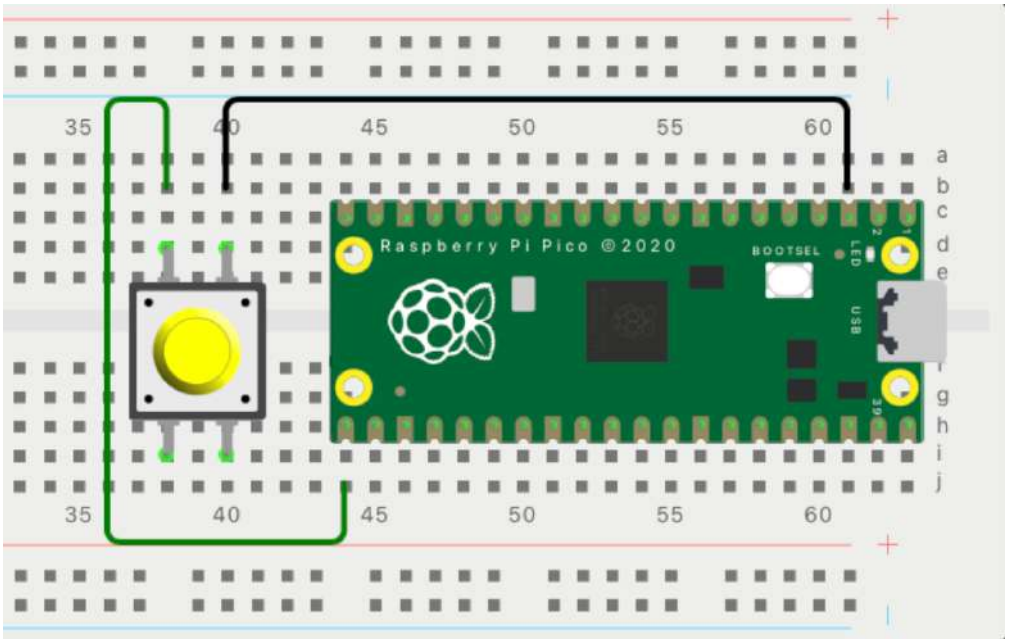
Цифровой сигнал мы читали лишь однажды - когда работали с тактовой кнопкой:

Устройство не сказать чтобы очень сложное:



Нажимаем кнопку, и пока держим ее нажатой, замыкаются контакты на ее торцах.

Стандартная схема подключения кнопки:



При этом вспомним, что подключать ее нужно в режиме "подтяжки к внутреннему резистору" `Pin.PULL_UP`:

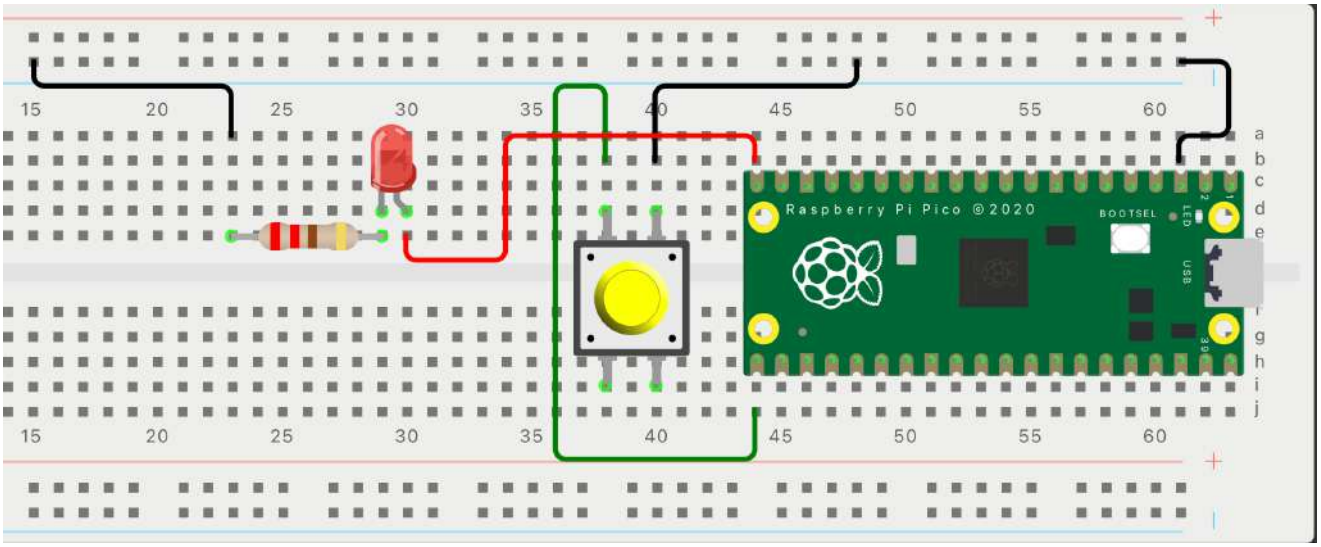
```
from machine import Pin
import time
```

```
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
```

Запустив этот код, мы могли видеть в консоли следующее:

> Испытайте свою память и вспомните, как получить в консоли сообщения с микроконтроллера!

Попробуем собрать все это вместе:



Вот с чего мы начинали когда-то давно....

```
from machine import Pin
import time

LED_PIN = 15
BUTTON_PIN = 16

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

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

Но этот вариант решения нам не нравился даже тогда! Почему? Потому что расточительно тратить свой потенциал, удерживая кнопку постоянно нажатой для того, чтобы диод постоянно горел или потухал.

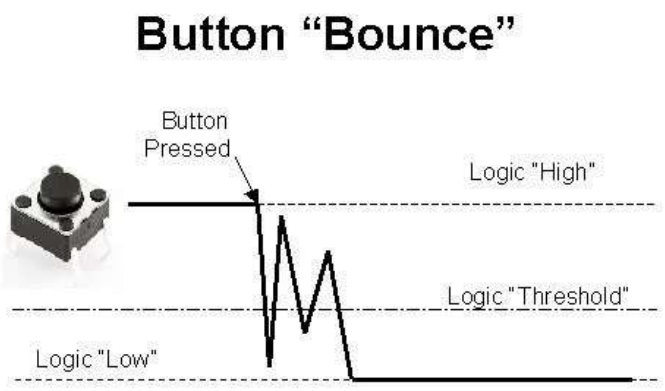
Как мы от этого избавились? Использовали голову... Точнее логику... Если быть еще точнее, логику посредством головы...

Короче, в итоге мы остановились на этом:

```
from machine import Pin
import time

LED_PIN = 15
BUTTON_PIN = 16

bounce_time = 250

red_led = Pin(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()
```

> Попробуйте самостоятельно разобраться в том, как работает этот код! Отдельно вспомните зачем мы использовали таймеры `time.ticks_ms()` и время "дребезга" `bounce_time`!

Возможно эта картинка поможет:



В любом случае приведенный ранее код мы переросли - ведь он неразрывно связан с логикой решения конкретной задачи - управления одним светодиодом одной кнопкой.

В нем перемешаны и прямое управление пинами микроконтроллера и логика решаемой нами задачи, это нужно исправить!

Напишем отдельный класс для кнопки `Button`!

Класс для кнопки `Button`

```
from machine import Pin
from time import ticks_ms, sleep_ms

class Button:
    def __init__(self, button_pin_number, debounce_time=50):
        self.pin = Pin(button_pin_number, Pin.IN, Pin.PULL_UP)
        self.debounce_time = debounce_time
        self.last_state = self.pin.value()
        self.last_change_time = ticks_ms()

    def check_button_click(self):
        current_time = ticks_ms()
        current_state = self.pin.value()
        if current_state != self.last_state:
            if current_time - self.last_change_time > self.debounce_time:
                self.last_state = current_state
                self.last_change_time = current_time
                if not current_state:  # Кнопка нажата (состояние LOW)
                    return "Click!"


BUTTON_PIN = 16
button = Button(button_pin_number=BUTTON_PIN)

while True:
    action = button.check_button_click()
```

```
    if action:
        print(ticks_ms(), action)
    sleep_ms(10)
```

Разберем получившийся код!

Начнем с простого - с того как работать с нашей кнопкой:

```
BUTTON_PIN = 16
button = Button(button_pin_number=BUTTON_PIN)

while True:
    action = button.check_button_click()
    if action:
        print(ticks_ms(), action)
    sleep_ms(10)
```

- Наша кнопка подключена к 16 пину, поэтому в первой строчке интерфейсной части мы присвоили это значение константе `BUTTON_PIN` (так как это константа название переменной пишется заглавными буквами).
- Далее мы создаем экземпляр класса `Button`, передавая в него в качестве необходимого атрибута `button_pin_number` ранее созданную константу `BUTTON_PIN`. Сам созданный объект кнопки присваивается переменной `button`.
- Далее в бесконечном цикле:
  - опрашиваем экземпляр класса `Button` из переменной `button` через его метод `check_button_click` (проверка клика кнопки);
  - этот метод возвращает строку `Click!` в момент нажатия кнопки (если в момент опроса кнопки она не была нажата метод вернет `None`);
  - результат опроса кнопки присваивается переменной `action`;
  - далее проверяется результат опроса кнопки: если результат `None` - условие `if action` не выполняется, в случае не пустой строки (напр. `Click!`) она будет приводиться к истине (`True`) и условие выполнится;
  - В случае выполнения условия в консоль будет выведен текст состоящий из текущего таймера `ticks_ms()` и непосредственного действия `action`;
  - далее идет задержка в 10 миллисекунд (`sleep_ms(10)`) для снижения нагрузки на микроконтроллер.

Разберем код самого класса `Button`:

- Рассмотрим конструктор класса:

```
    def __init__(self, button_pin_number, debounce_time=50):
        self.pin = Pin(button_pin_number, Pin.IN, Pin.PULL_UP)
        self.debounce_time = debounce_time
```

```
        self.last_state = self.pin.value()
        self.last_change_time = ticks_ms()
```

В качестве необходимого атрибута, при создании экземпляра класса, необходимо передать номер пина `button_pin_number`, второй параметр (`debounce_time=50`) имеет значение по умолчанию и определяет интервал возможного "дребезга" кнопки в течении которого изменение состояния кнопки не должно учитываться.

Сам объект содержит в себе 4 атрибута:

- `pin` - экземпляр класса `Pin` из модуля `machine`, настроенный в режим подтяжки к внутреннему сопротивлению;
- `debounce_time` - время задержки для устранения "дребезга";
- `last_state` - сохраняет последнее зафиксированное состояние кнопки;
- `last_change_time` - время последнего изменения состояния.

Разберем работу метода `check_button_click`

```
    def check_button_click(self):
        current_time = ticks_ms()
        current_state = self.pin.value()
        if current_state != self.last_state:
            if current_time - self.last_change_time > self.debounce_time:
                self.last_state = current_state
                self.last_change_time = current_time
                if not current_state:  # Кнопка нажата (состояние LOW)
                    return "Click!"
```

- при вызове метода определяется текущее время и состояние пина:

```
current_time = ticks_ms()
current_state = self.pin.value()
```

- далее, если текущее состояние кнопки изменилось (оно не равно предыдущему состоянию):

```
 if current_state != self.last_state:
        ...
```

- и время с прошлого изменения кнопки превышает возможный интервал "дребезга":

```
if current_time - self.last_change_time > self.debounce_time:
        ...
```

- обновляются состояния кнопки и времени в атрибутах объекта:

```
self.last_state = current_state
self.last_change_time = current_time
```

- далее, если текущее состояние пина `LOW` (кнопка нажата) метод возвращает строку `"Click!"`. (Если не делать эту проверку строка `"Click!"` возвращалась бы в момент нажатия и отпускания кнопки).

Используем написанный класс кнопки для управления светодиодом:

```
from machine import Pin

from time import ticks_ms, sleep_ms


class Led:
    def __init__(self, led_pin):
        self.led = Pin(led_pin, Pin.OUT)
        self.led_state = None

    def on(self):
        self.led_state = "ON"
        self.led.on()

    def off(self):
        self.led_state = "OFF"
        self.led.off()

    def switch(self):
        if self.led_state == "ON":
            self.off()
        else:
            self.on()


class Button:
    def __init__(self, button_pin_number, debounce_time=50):
        self.pin = Pin(button_pin_number, Pin.IN, Pin.PULL_UP)
        self.debounce_time = debounce_time
        self.last_state = self.pin.value()
        self.last_change_time = ticks_ms()

    def check_button_click(self):
        current_time = ticks_ms()
        current_state = self.pin.value()
        if current_state != self.last_state:
            if current_time - self.last_change_time > self.debounce_time:
```

```
                self.last_state = current_state
                self.last_change_time = current_time
                if not current_state:  # Кнопка нажата (состояние LOW)
                    return "Click!"


LED_PIN = 15
BUTTON_PIN = 16

led = Led(led_pin=LED_PIN)
button = Button(button_pin_number=BUTTON_PIN)

while True:
    action = button.check_clicks()
    if action:
        led.switch()
    sleep_ms(10)
```

Класс кнопки `Button` для обработки множественных кликов

Опираясь на ранее написанный код, внесем в него несколько изменений:

- добавим в метод `__init__` новый атрибут `self.click_count = 0` для сохранения количества "кликов";
- заменим строку `return "Click!"` в методе `check_button_state` на инкримент счетчика `click_count`:

```
if not current_state:  # Кнопка нажата (состояние LOW)
        self.click_count += 1
```

- добавим в класс новый метод `check_clicks`:

```
    def check_clicks(self):
        self.check_button_state()
        if self.pin.value() and self.click_count > 0:
            return self.click_count
```

в котором мы:

- проверяем состояние кнопки через метод `check_button_state` (в случае отработки нажатия метод изменит значение счетчика `click_count`);
- если кнопка отпущена и значение счетчика больше нуля метод вернет текущее количество нажатий на кнопку.

В результате получим следующий код:

```
from machine import Pin

from time import ticks_ms, sleep_ms


class Button:
    def __init__(self, button_pin_number, debounce_time=50):
        self.pin = Pin(button_pin_number, Pin.IN, Pin.PULL_UP)
        self.debounce_time = debounce_time
        self.last_state = self.pin.value()
        self.last_change_time = ticks_ms()
        self.click_count = 0

    def check_button_state(self):
        current_time = ticks_ms()
        current_state = self.pin.value()
        if current_state != self.last_state:
            if current_time - self.last_change_time > self.debounce_time:
                self.last_state = current_state
                self.last_change_time = current_time
                if not current_state:  # Кнопка нажата (состояние LOW)
                    self.click_count += 1

    def check_clicks(self):
        self.check_button_state()
        if self.pin.value() and self.click_count > 0:
            return self.click_count


BUTTON_PIN = 16
button = Button(button_pin_number=BUTTON_PIN)

while True:
    action = button.check_clicks()
    if action:
        print(ticks_ms(), action)
    sleep_ms(10)
```

В результате его запуска в консоль после первого нажатия на кнопку будет постоянно выводиться текущее количество нажатий на кнопку.

Пока результат не полностью отвечает нашим требованиям, но главное, что корректно обрабатываются нажатия и ведется их счет.

Отметим, что до момента первого нажатия в консоль ничего не выводится (что хорошо), а значит для остановки вывода сигнала с кнопки нам было бы достаточно "обнулить" текущее состояние счетчика `click_count`.

Проблема в том, что если его постоянно "обнулять" он теряет весь свой смысл, так что для корректной обработки нажатий необходимо добавить дополнительный таймер, для того чтобы успеть выполнить несколько последовательных нажатий кнопки.

Для этого добавим в наш объект новый атрибут `last_click_time`, сохраняющий в себе время последнего отработанного клика.

Здравый смысл подсказывает, что этот таймер должен обновляться синхронно со счетчиком `click_count` в момент его изменения. (Это может произойти только при выполнении метода `check_button_state`)

Добавим также новый атрибут `total_command_time` , определяющий допустимый интервал времени между кликами при их наборе.

Далее модифицируем метод `check_button_state`, добавив в него дополнительно блок:

```
if self.pin.value() and self.click_count > 0:
            if current_time - self.last_click_time >
self.total_command_time:
                if self.click_count == 1:
                    self.click_count = 0
                    return "single"
                elif self.click_count == 2:
                    self.click_count = 0
                    return "double"
                elif self.click_count == 3:
                    self.click_count = 0
                    return "triple"
                else:
                    result = f"multiple - {self.click_count}"
                    self.click_count = 0
                    return result
```

в котором вывод значения может наступить после выполнения трех условий:

- отпущенной кнопки (`self.pin.value()`);
- количества зафиксированный "кликов" больше 0 (`self.click_count > 0`);
- если время с момента последнего "клика" превышает максимальное время команды `total_command_time` (`current_time - self.last_click_time > self.total_command_time`).

В случае выполнения всех трех условий выполняется вывод возможных значений:

- `"single"` для одиночного нажатия;
- `"double"` - для двойного;
- `"triple"` - для тройного.

В случае большего числа набранных "кликов" - метод вернет форматированную `f""` строку - `f"multiple - {self.click_count}"`.

Собрав все это вместе, получим:

```
from machine import Pin
from time import ticks_ms, sleep_ms


class Button:
    def __init__(self, button_pin_number, debounce_time=50,
total_command_time=500):
        self.pin = Pin(button_pin_number, Pin.IN, Pin.PULL_UP)
        self.debounce_time = debounce_time
        self.total_command_time = total_command_time
        self.last_state = self.pin.value()
        self.last_change_time = ticks_ms()
        self.click_count = 0
        self.last_click_time = ticks_ms()

    def check_button_state(self):
        current_time = ticks_ms()
        current_state = self.pin.value()
        if current_state != self.last_state:
            if current_time - self.last_change_time > self.debounce_time:
                self.last_state = current_state
                self.last_change_time = current_time
                if not current_state:  # Кнопка нажата (состояние LOW)
                    self.click_count += 1
                    self.last_click_time = current_time

    def check_clicks(self):
        self.check_button_state()
        current_time = ticks_ms()
        if self.pin.value() and self.click_count > 0:
            if current_time - self.last_click_time >
self.total_command_time:
                if self.click_count == 1:
                    self.click_count = 0
                    return "single"
                elif self.click_count == 2:
                    self.click_count = 0
                    return "double"
                elif self.click_count == 3:
                    self.click_count = 0
                    return "triple"
                else:
                    result = f"multiple - {self.click_count}"
                    self.click_count = 0
```

```
                    return result


BUTTON_PIN = 16
button = Button(button_pin_number=BUTTON_PIN)

while True:
    action = button.check_clicks()
    if action:
        print(action)
    sleep_ms(10)
```

Используем полученный класс для хитрого управления светодиодом:

```
from machine import Pin

from time import ticks_ms, sleep_ms


class Led:
    def __init__(self, led_pin):
        self.led = Pin(led_pin, Pin.OUT)
        self.led_state = None

    def on(self):
        self.led_state = "ON"
        self.led.on()

    def off(self):
        self.led_state = "OFF"
        self.led.off()

    def switch(self):
        if self.led_state == "ON":
            self.off()
        else:
            self.on()


class Button:
    def __init__(self, button_pin_number, debounce_time=50,
total_command_time=500):
        self.pin = Pin(button_pin_number, Pin.IN, Pin.PULL_UP)
        self.debounce_time = debounce_time
        self.total_command_time = total_command_time
        self.last_state = self.pin.value()
        self.last_change_time = ticks_ms()
        self.click_count = 0
        self.last_click_time = 0
```

```
    def check_button_state(self):
        current_time = ticks_ms()
        current_state = self.pin.value()
        if current_state != self.last_state:
            if current_time - self.last_change_time > self.debounce_time:
                self.last_state = current_state
                self.last_change_time = current_time
                if not current_state:  # Кнопка нажата (состояние LOW)
                    self.click_count += 1
                    self.last_click_time = current_time

    def check_clicks(self):
        self.check_button_state()
        current_time = ticks_ms()
        if self.pin.value() and self.click_count > 0:
            if current_time - self.last_click_time >
self.total_command_time:
                if self.click_count == 1:
                    self.click_count = 0
                    return "single"
                elif self.click_count == 2:
                    self.click_count = 0
                    return "double"
                elif self.click_count == 3:
                    self.click_count = 0
                    return "triple"
                else:
                    result = f"multiple - {self.click_count}"
                    self.click_count = 0
                    return result
        return None


LED_PIN = 15
BUTTON_PIN = 16

led = Led(led_pin=LED_PIN)
button = Button(button_pin_number=BUTTON_PIN)

while True:
    action = button.check_clicks()
    if action == "single":
        led.on()
    if action == "double":
        led.off()
    if action == "triple":
        led.switch()
    sleep_ms(10)
```

В этом примере:

- **однократное нажатие кнопки** всегда будет пытаться включить светодиод (если он уже включен ничего не будет происходить);
- **двухкратное нажатие** - будет пытаться его выключить;
- а **трехкратное** - переключать светодиод в обратное состояние.