Третье практическое занятие

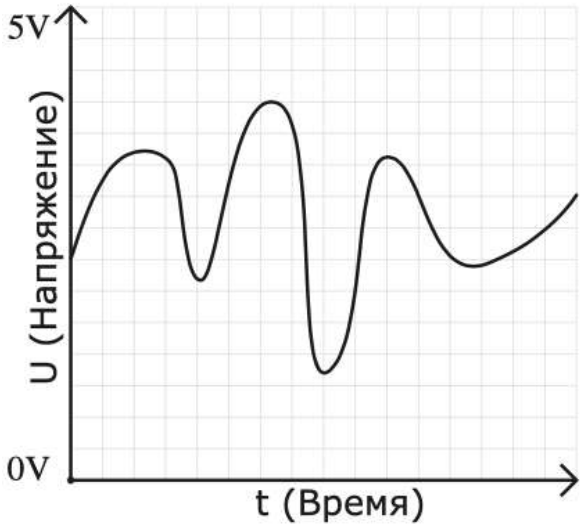
Работа с аналоговым сигналом

Ана́логовый сигна́л — сигнал, порождаемый физическим процессом, параметры которого можно измерить в любой момент времени.

Техническое определение *аналогового сигнала* дается в ГОСТе, посвящённом передаче данных:

> **Аналоговый сигнал** — сигнал данных, у которого каждый из представленных параметров описывается функцией времени и непрерывным множеством возможных значений.

Аналоговый сигнал несёт информацию, зашифрованную уровнем его напряжения, то есть количеством электронов, которые входят в микроконтроллер. Чем больше разница электронов на входе в пин, относительно количества их на контакте GND, тем больше напряжение мы получим. Аналоговый сигнал мы можем получить со множества устройств, и обусловлено это принципами их работы. Например, мы можем измерить напряжение на фоторезисторе, который меняет своё сопротивление, а соответственно и напряжение, в зависимости от того, сколько света на него падает. Или, например, микрофон - окружающие звуки создают в нём ток, а следовательно и некоторое напряжение, которое можно измерить и сохранить в память микроконтроллера.



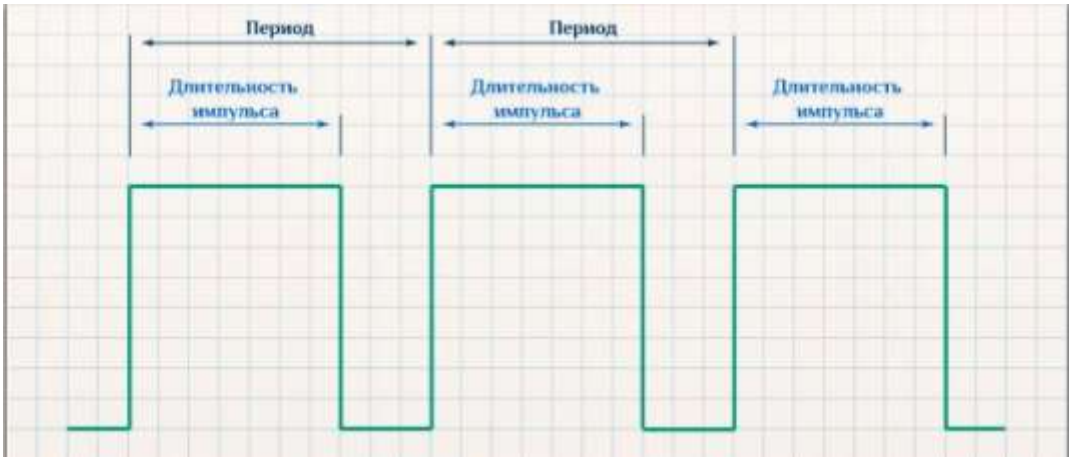
Микроконтроллер может работать только с цифровыми сигналами. Поэтому в нём для чтения аналогового сигнала используется специальное устройство, которое называется **Аналого-цифровой преобразователь (АЦП)**. А для записи - **Широтно-импульсная модуляция (ШИМ)**.

Широтно-импульсная модуляция сигнала

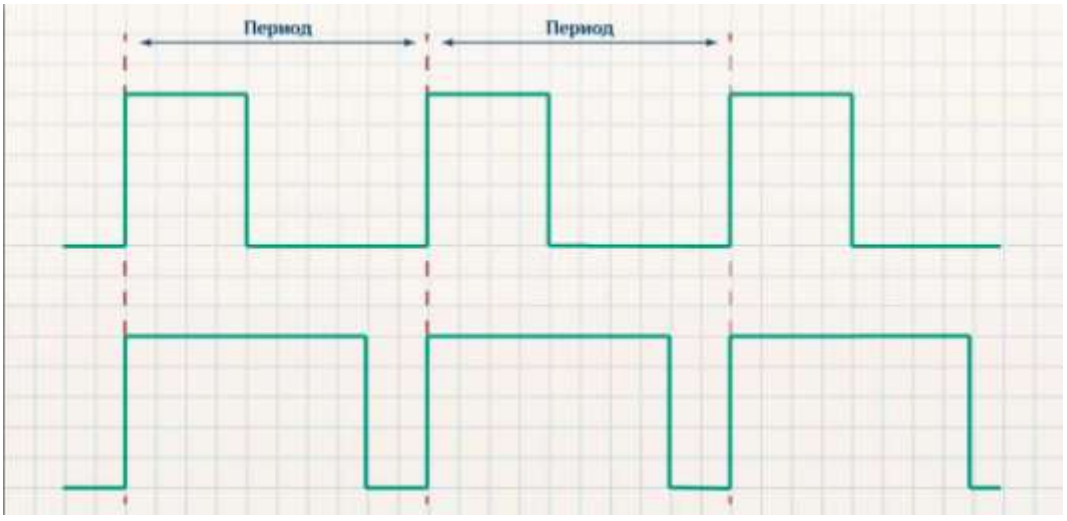
Широ́тно-и́мпульсная модуля́ция (ШИМ, pulse-width modulation (PWM)) — процесс управления мощностью методом пульсирующего включения и выключения потребителя энергии.

ШИМ-сигналы могут использоваться, к примеру, для управления сервоприводами или двигателями постоянного тока, либо для контроля яркости светодиодов. И это только несколько самых основных применений. Таким образом, необходимость в генерации ШИМ возникает довольно часто и в самых разных проектах.

Итак, ШИМ-сигнал представляет собой последовательность импульсов с постоянной частотой следования, но разной длительностью. Давайте рассмотрим наглядный пример:



Поскольку частота импульсов постоянна, то значит и период имеет фиксированную величину. А вот **длительность импульса** может меняться, собственно так и происходит модуляция. Рассмотрим два разных сигнала, использующихся для включения/отключения светодиодов:



Пусть светодиоды загораются при высоком уровне сигнала. **Период** в обоих случаях идентичен, а вот **длительность импульсов** второго сигнала больше в 2 раза. Соответственно, светодиод во втором случае будет гореть ярче, чем в первом.

Строго говоря, светодиод будет мигать, то есть часть периода гореть, а часть периода - нет. Но при высокой частоте следования импульсов глазу будут не заметны эти мигания, поэтому на практике получим 2 светодиода с разной яркостью. То есть, чем большую часть периода диод горит - тем ярче будет в результате светить.

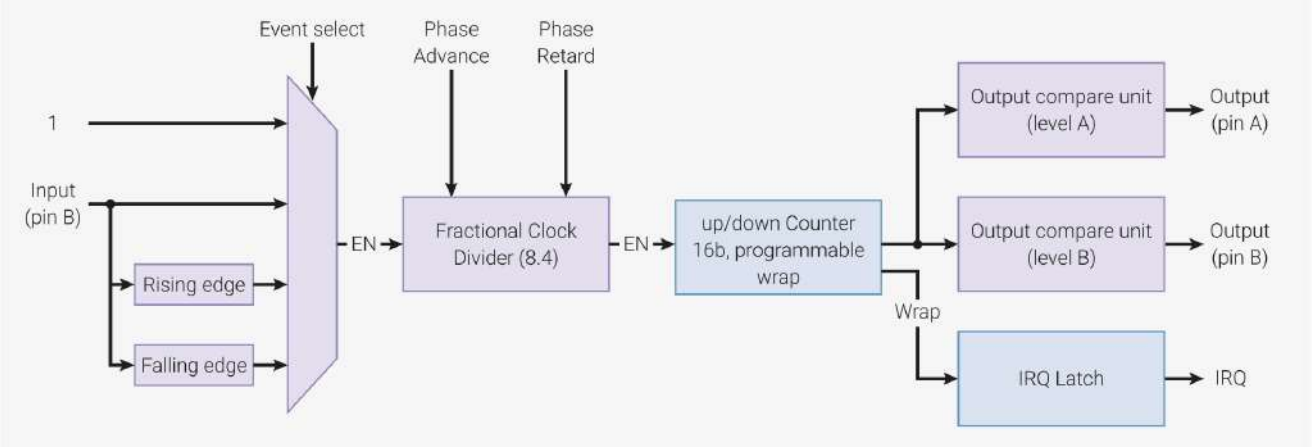
ШИМ-сигналы характеризуют параметром, который называется - **коэффициент заполнения** (**duty cycle**). Он вычисляется по формуле:

$$D = \frac{\tau}{T} \cdot 100\%$$

Здесь τ - длительность импульса, а T - период сигнала.

Вернемся к рассмотренным выше сигналам - для первого случая длительность импульса составляет $\frac{4}{10}$ периода сигнала, а значит $D = 40\%$. Аналогично, для второго случая $D = 80\%$.

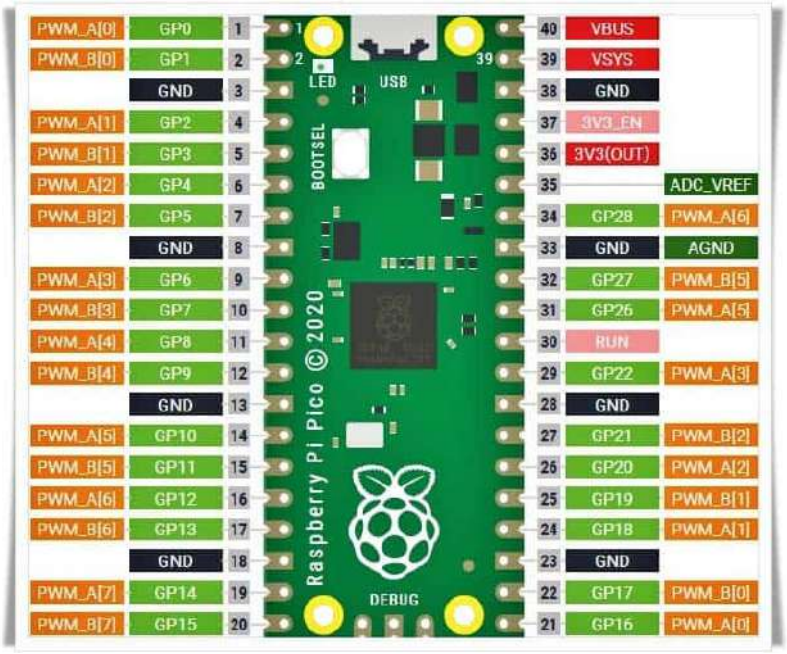
Микроконтроллер RP2040, лежащий в основе Raspberry Pi Pico, имеет 8 независимых между собой срезов ШИМ. При этом каждый срез может работать в двух выводах A и B. Это означает, что мы можем установить частоту одного из них, не влияя на остальные.



В документации к микроконтроллеру приведена следующая таблица, определяющая соответствия между конкретным пином и выводом ШИМ.:

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B		

Перенеся ее на схему GPIO получим:



Несмотря на то, что ШИМ сигнал может передаваться на любой из цифровых пинов, однако, одновременно и независимо смогут работать только 16 (A[0]-A[7] и B[0] - B[7]). Другими словами управлять ШИМ одновременно на 4 и 20 пинах не получится так они фактически запараллелены с выводом A[2].

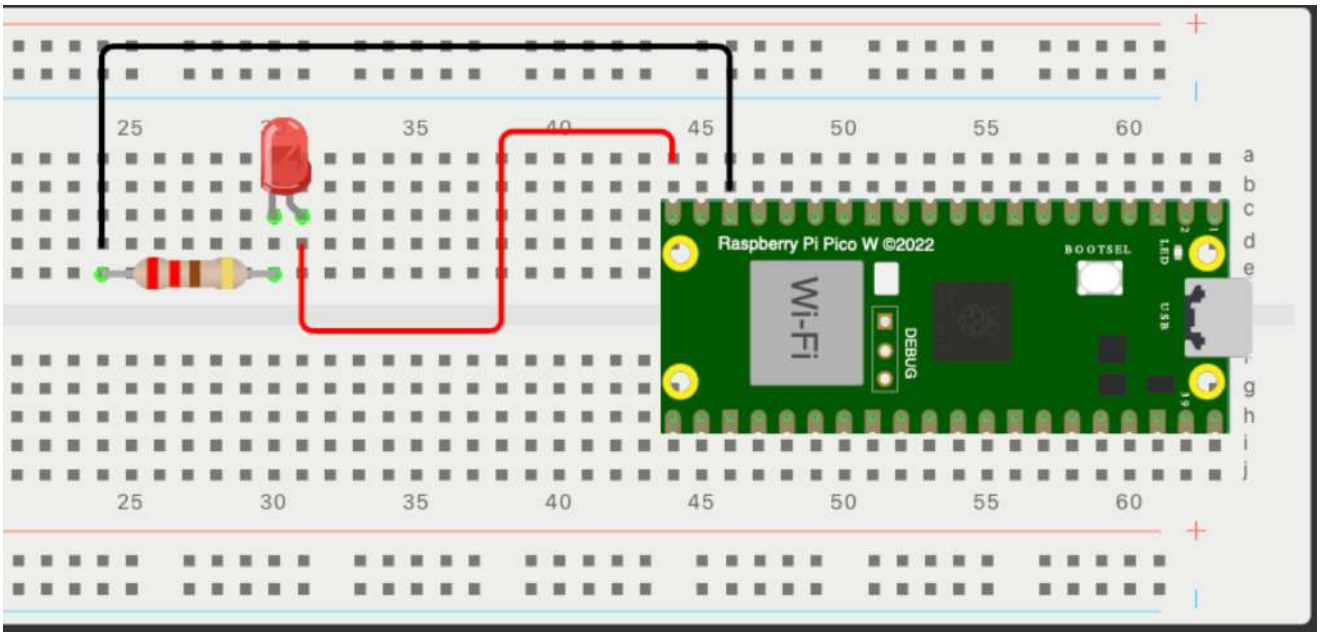
Управление яркостью светодиода через ШИМ

Так как в Raspberry Pi Pico присутствует возможность вывода ШИМ на все цифровые пины схема подключения будет выглядеть аналогично подключению с прошлого занятия.

При этом, естественно, сохраняются все требования к соблюдению полярности светодиода и подключению в цепь дополнительного резистора на 220Ω.

Схема подключения

Одна из возможных схем подключения представлена на рисунке:



Код программы

```
from machine import Pin, PWM
from time import sleep

red_led_pin = 15

red_led = Pin(red_led_pin, Pin.OUT)
red_led_pwm = PWM(red_led)
red_led_pwm.freq(1000)

while True:
    for duty in range(65535):
        red_led_pwm.duty_u16(duty)
        sleep(0.0001)
    for duty in range(65535, 0, -1):
        red_led_pwm.duty_u16(duty)
        sleep(0.0001)
```

После загрузки представленного скрипта на микроконтроллер светодиод должен последовательно менять свою яркость.

Разберем представленный код поэлементно:

Управление ШИМ требует использование специального класса PWM импортируемого из пакета machine:

```
from machine import Pin, PWM
```

Для создания и настройки объекта PWM необходимо указать конкретный пин на котором будет генерироваться ШИМ и настроить его частоту.

В разбираемом коде это сделано в следующем блоке:

```
red_led_pin = 15

red_led = Pin(red_led_pin, Pin.OUT)
red_led_pwm = PWM(red_led)
red_led_pwm.freq(1000)
```

В котором поэтапно создается объект пина передающего сигнал (аналогично работе с цифровым сигналом), после чего создается экземпляр класса PWM, объект - red_led_pwm.

Метод объекта:

```
red_led_pwm.freq(1000)
```

Определяет частоту ШИМ в 1000 Гц.

Так как ШИМ предполагает передачу сигнала только наружу, инициализацию объекта класса PWM можно заменить более компактной записью:

```
red_led_pin = 15

red_led_pwm = PWM(red_led_pin, freq=1000)
```

После инициализации объекта red_led_pwm в бесконечном цикле происходит управление им:

```
while True:
    for duty in range(65535):
        red_led_pwm.duty_u16(duty)
        sleep(0.0001)
    for duty in range(65535, 0, -1):
        red_led_pwm.duty_u16(duty)
        sleep(0.0001)
```

Непосредственное управление ШИМ осуществляется за счет указания коэффициента заполнения (duty cycle) с помощью метода duty_u16():

```
red_led_pwm.duty_u16(duty)
```

Параметр duty, переданный в метод, определяет длительность импульса высокого сигнала внутри цикла ШИМ.

> Передаваемое в метод значение **должно быть целым 16-битным числом**!! (числом от 0 до 65535 класса int)

Изменение значение параметра duty происходит в циклах:

- от 0 до 65534:

```
for duty in range(65535):
    red_led_pwm.duty_u16(duty)
    sleep(0.0001)
```

- от 65535 до 1:

```
for duty in range(65535, 0, -1):
    red_led_pwm.duty_u16(duty)
    sleep(0.0001)
```

Дополнительно

Поэкспериментируйте со следующим кодом:

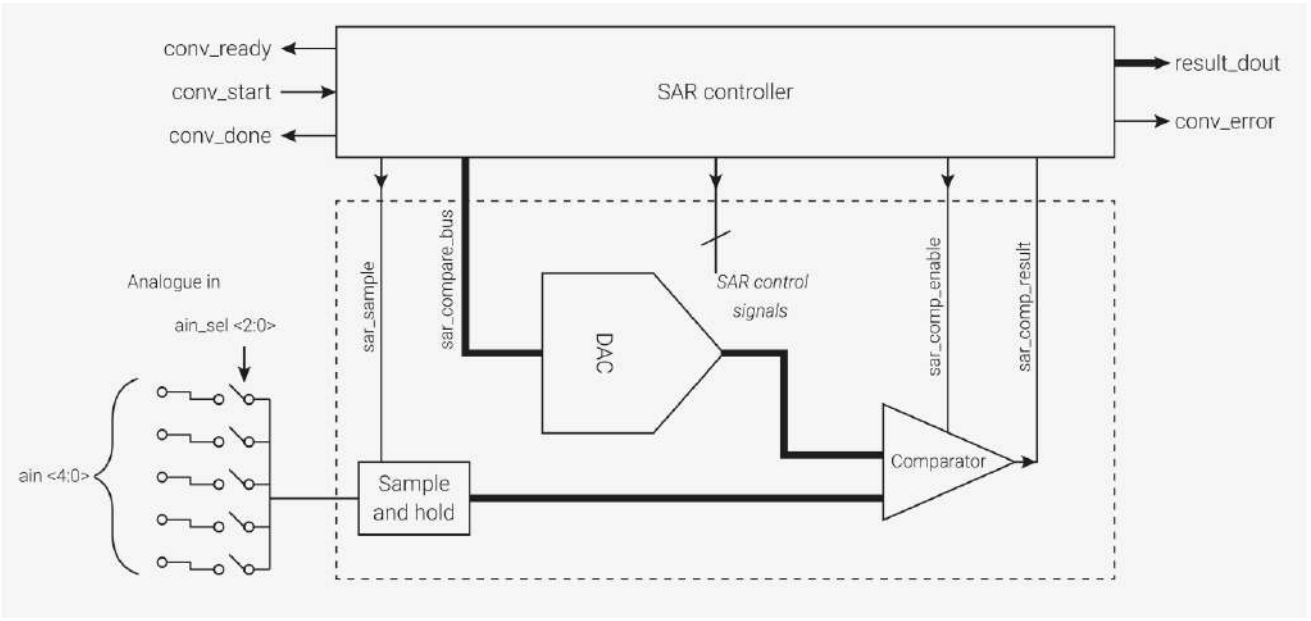
```
from machine import Pin, PWM

red_led_pin = 15

red_led = Pin(red_led_pin, Pin.OUT)
red_led_pwm = PWM(red_led)

duty = int(input("Duty: "))
freq = int(input("Freq: "))

red_led_pwm.freq(freq)

red_led_pwm.duty_u16(duty)
```

позволяющим вручную изменять с консоли значения коэффициента заполнения и частоты.

> Попробуйте найти границу частоты, при которой мерцание диода будет заметно глазу и камере телефона.

Аналого-цифровой преобразователь

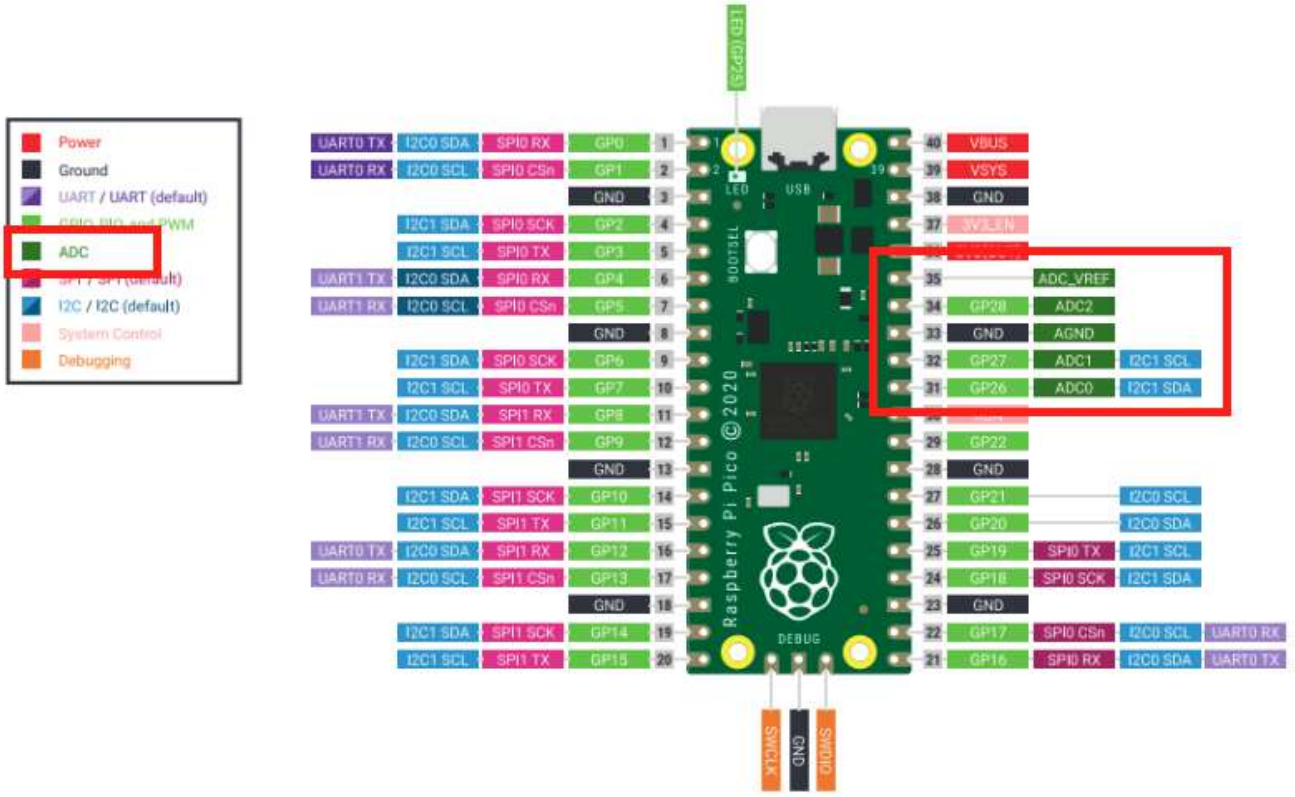
В RP Pico используется АЦП последовательного приближения:



Встроенная система обработки данных создает аналоговое опорное напряжение, равное выходному сигналу цифрового кода устройства выборки-хранения (3.3V). Оба сигнала передаются в компаратор, который отправляет результат сравнения в SAR контроллер. Этот процесс продолжается в течение n последовательных раз, причем n является битовым разрешением самого АЦП, пока не будет найдено значение, ближайшее к фактическому сигналу.

АЦП последовательного приближения не имеют внутреннего механизма фильтрации-сглаживания, поэтому при выборе слишком низкой частоты выборки ложные сигналы.

Плата Raspberry Pi Pico содержит в своем составе 4 АЦП канала с разрешением 12 бит, которые программно масштабируются до 16 бит (значений от 0 до 65535), но один из них подключен к внутреннему датчику температуры. Оставшиеся АЦП канала расположены на контактах GPIO26, GPIO27, GPIO28 и имеют обозначения ADC0, ADC1 и ADC2 соответственно.



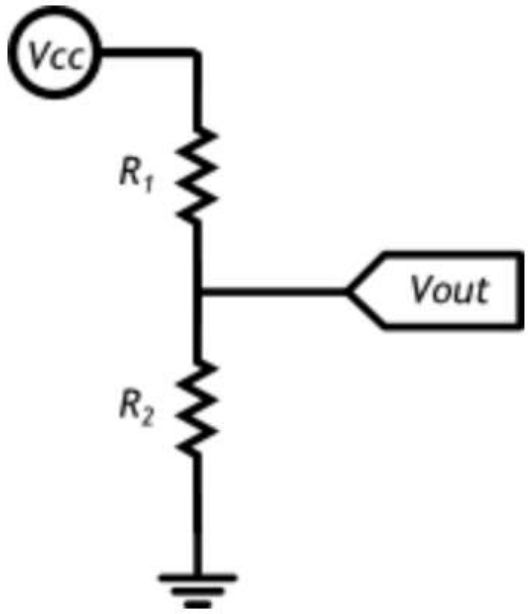
Итак, в плате Raspberry Pi Pico мы имеем три АЦП канала с разрешением 12 бит. Однако при использовании языка MicroPython мы можем увеличить разрешение этих АЦП каналов до 16 бит. С помощью класса ADC (АЦП) пакета machine языка MicroPython происходит масштабирование 12-битного разрешения в 16-битное, то есть мы получаем максимальное значение на выходе АЦП 65535 (т.е. 2^{16}) вместо 4096 (т.е. 2^{12}) при 12-битном разрешении.

Фактически когда мы отправляем команду измерить аналоговый сигнал на этих входах, он поступает в АЦП, и тот преобразует его в цифровой сигнал. С помощью этого количества бит можно сохранить число от 0 до 65535, где 0 соответствует нулю вольт, а 65535 соответствует максимальному напряжению (по умолчанию это напряжение питания контроллера, то есть 3.3 вольт).

Так как многие аналоговые датчики не имеют возможности к изменению напряжения от изменения внешних условий для работы с ними необходимо включать в схему **делитель напряжения**.

Делитель напряжения

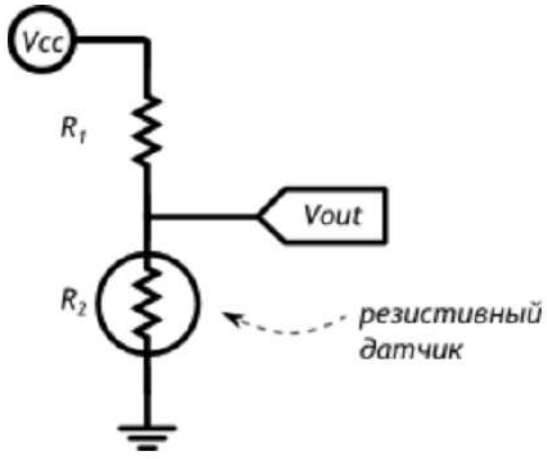
Классический способ понижения напряжения при постоянном токе предполагает использование "делителя напряжения". По своей сути это схема с последовательно подключенной парой резисторов:



При такой схеме выходное напряжение V_{out} будет определяться отношением сопротивлений по формуле:

$$V_{out} = \frac{R_2 \cdot V_{cc}}{R_1 + R_2}$$

Предположив возможность изменения сопротивлений резисторов становится возможным управление выходным напряжением V_{out} в пределах от 0 до V_{cc}.



Классическими примерами резистивных датчиков могут служить:

- потенциометры;
- фоторезисторы;

- термисторы.

Рассмотрим принцип их работы.

Подключение потенциометра

Потенцио́метр — жаргонное название переменного резистора, включенного как делитель электрического напряжения. Под потенциометрами, как правило, подразумевают резисторы с подвижным отводным контактом (движком).

Большинство разновидностей переменных резисторов могут использоваться как в качестве потенциометров, так и в качестве реостатов, разница в схемах подключения и в назначении (потенциометр — регулятор напряжения, реостат — силы тока).

Потенциометры используются в качестве регуляторов параметров (громкости звука, мощности, выходного напряжения и т. д.), для подстройки внутренних характеристик цепей аппаратуры (подстроечный резистор), на основе прецизионных потенциометров построены многие типы датчиков углового или линейного перемещения.

Схема работы потенциометра:

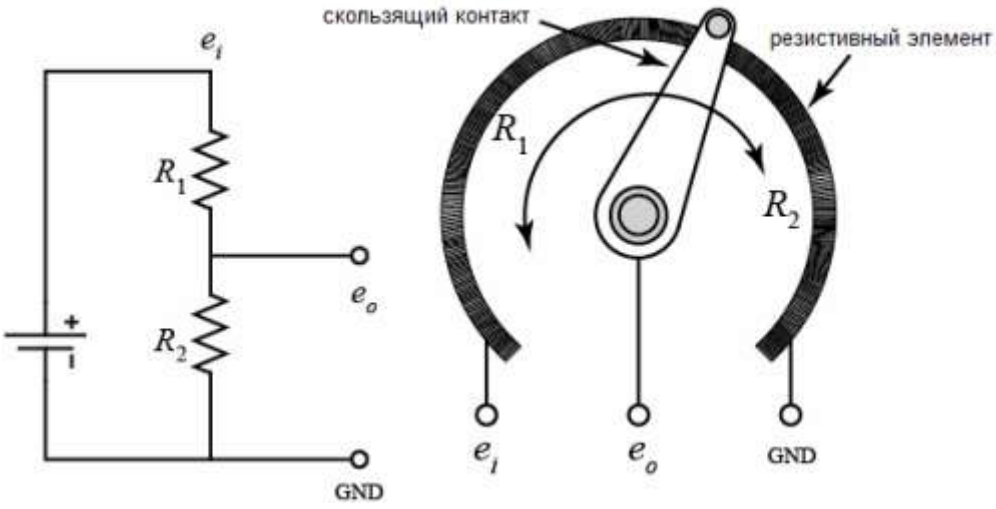
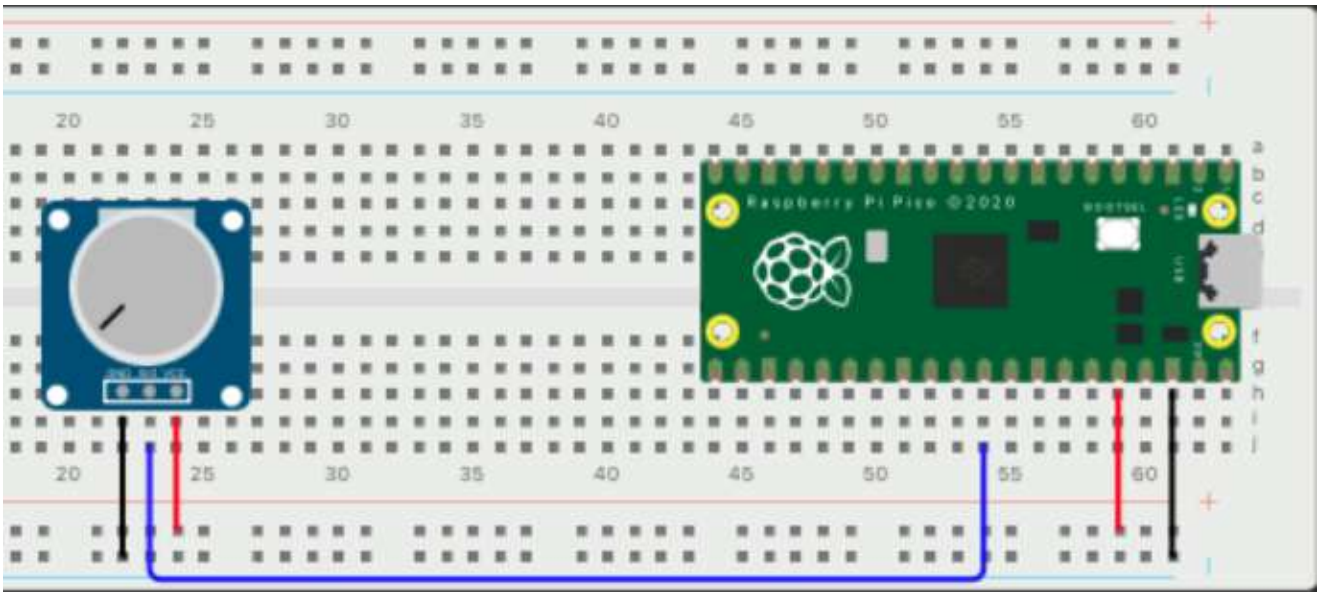
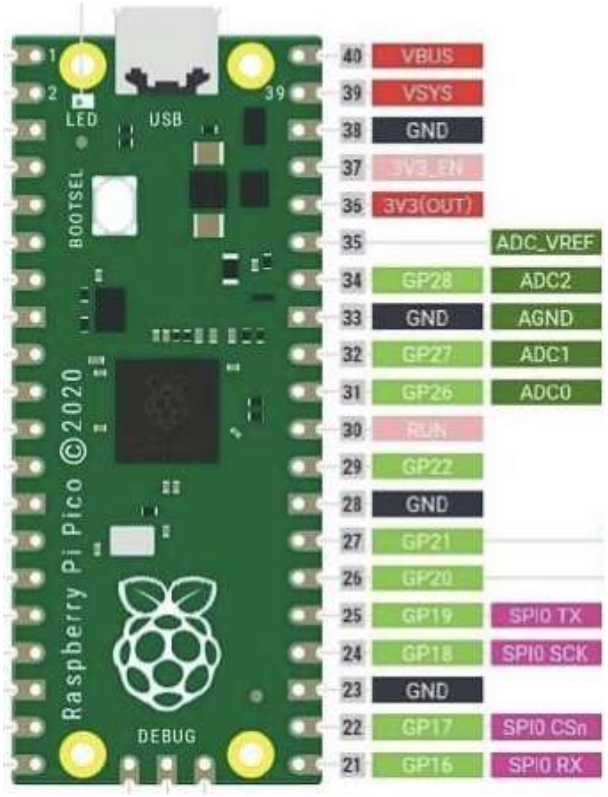


Схема подключения



> ВАЖНО! Центральный пин потенциометра должен быть подключен к пину с АЦП!



Полярность крайних пинов будет определять возрастание или убывание напряжения при повороте потенциометра.

Код программы

```
from machine import Pin, ADC
from time import sleep

pot_pin = 26
```

```
pot = ADC(Pin(pot_pin))

while True:
    pot_value = pot.read_u16()
    print(pot_value)
    sleep(0.1)
```

Разберем представленный код!

За работу с АЦП отвечает класс ADC пакета machine:

```
from machine import Pin, ADC
```

Класс принимает объект пина на котором будет выполняться "чтение" аналогового сигнала:

```
pot_pin = 26

pot = ADC(Pin(pot_pin))
```

Далее в бесконечном цикле:

```
while True:
    pot_value = pot.read_u16()
    print(pot_value)
    sleep(0.1)
```

С задержкой в 0.1 секунды будет считываться 16-битное значение (число от 0 до 65535):

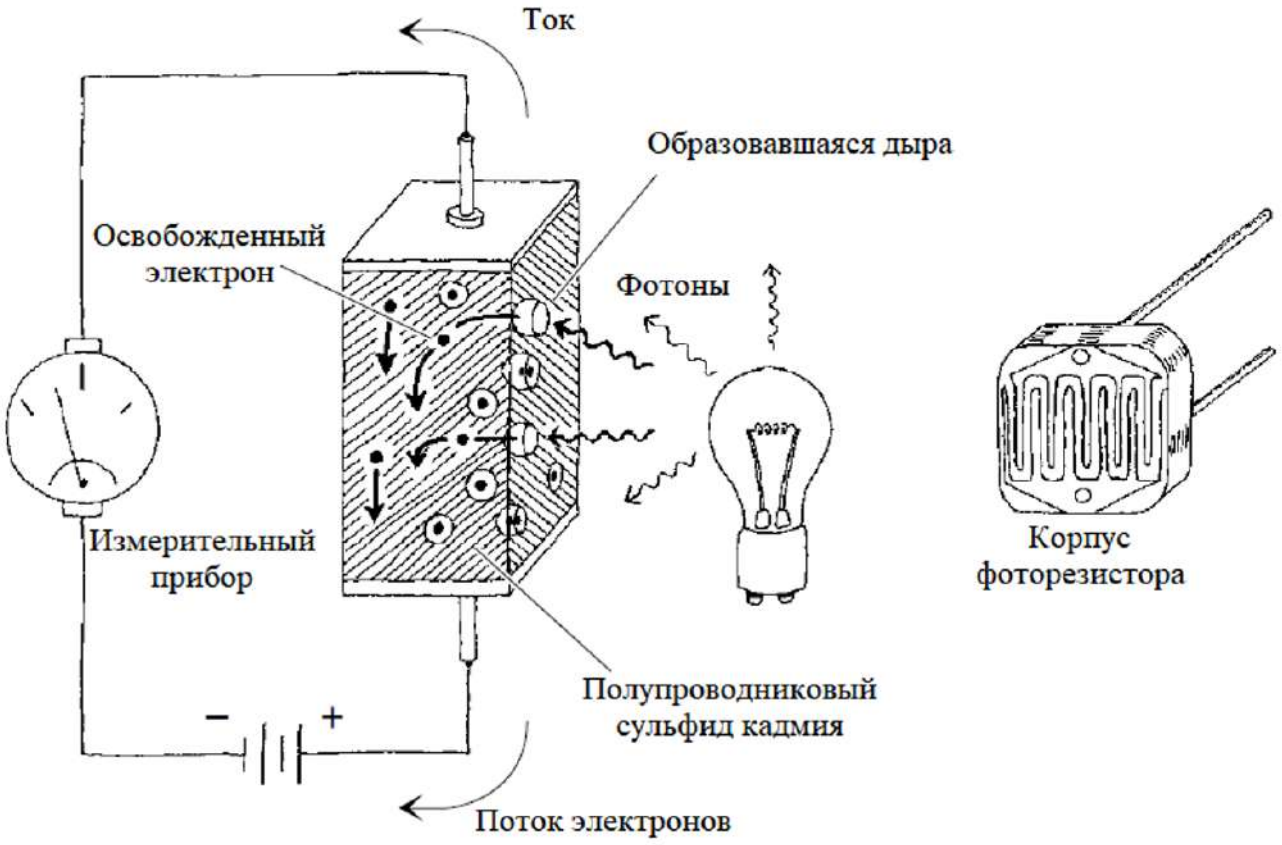
```
pot_value = pot.read_u16()
```

значение 0 эквивалентно 0V, а 65535 - опорному напряжению (по умолчанию 3.3V).

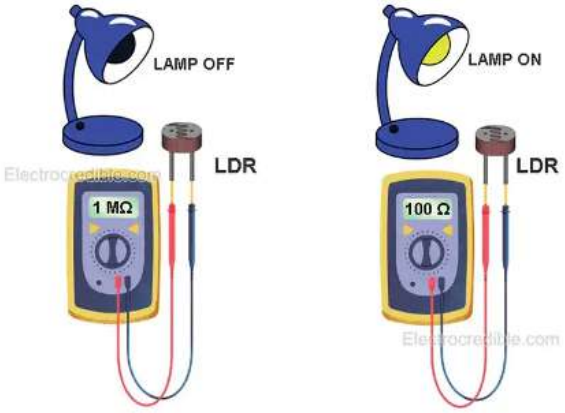
Полученное значение pot_value выводится в консоль.

Работа с фоторезистором

Фоторезистор (или LDR) — компонент, меняющий сопротивление в зависимости от количества света падающего на него.



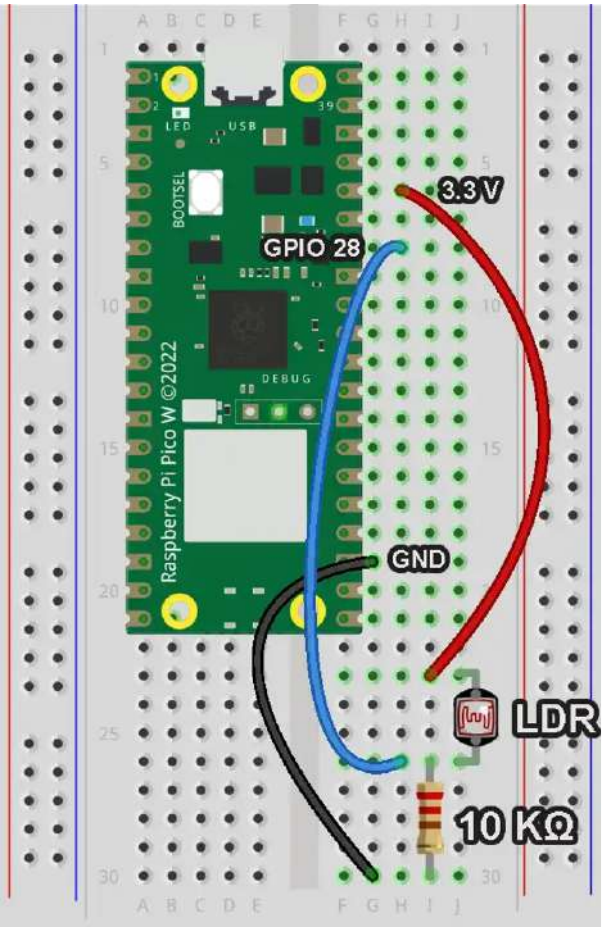
В полной темноте он имеет максимальное сопротивление в десятки килоом, а по мере роста освещённости сопротивление уменьшается до сотен ом.



На его основе очень просто создать схему, которая бы поставляла данные об уровне освещённости в виде аналогового сигнала на управляющую электронику.

Схема подключения

Фоторезистор подключается через делитель напряжения:



При этом дополнительный постоянный резистор должен быть соединен последовательно с фоторезистором.

Общий контакт резисторов подключается к аналоговому пину.

> Номинальное сопротивление постоянного резистора будет влиять на ширину считываемых аналоговым пином значений!

Код программы

```
from machine import Pin, ADC
from time import sleep

ldr_pin = 28

ldr = ADC(Pin(ldr_pin))

while True:
  ldr_value = ldr.read_u16()
  print(ldr_value)
  sleep(0.1)
```

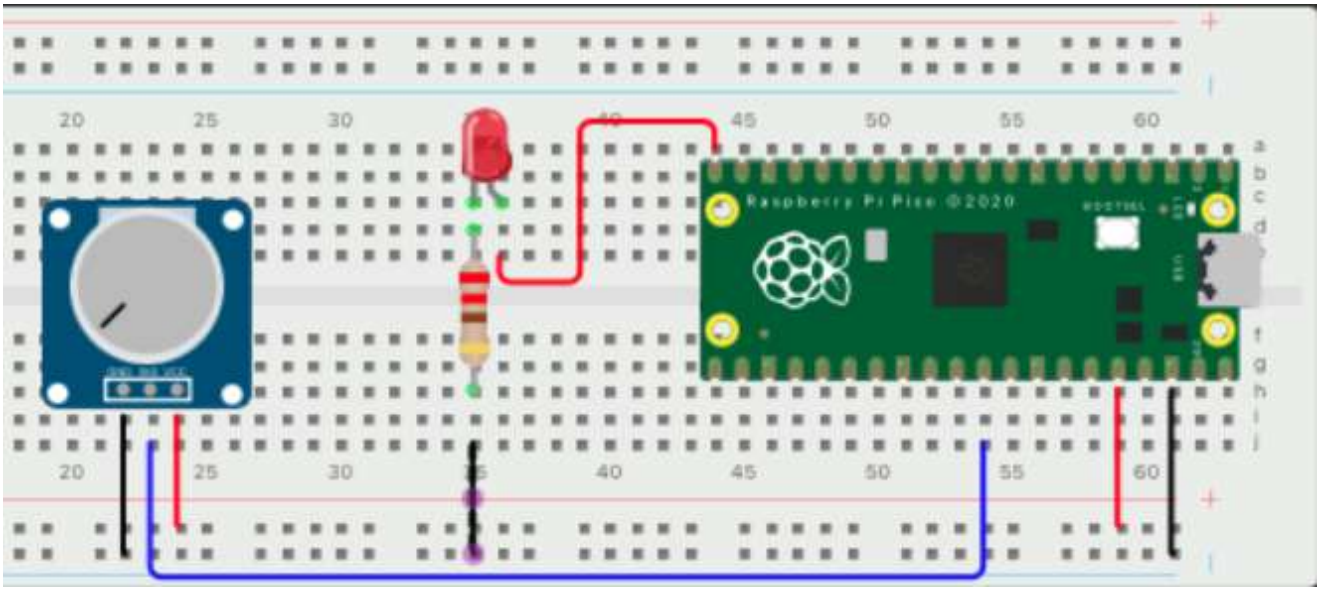
В целом, за исключением названий переменных, приведенная программа аналогична программе для получения значений с потенциометра.

Совместное использование ШИМ и АЦП

Управление яркостью светодиода потенциометром

Совместим вместе ранее представленные схемы и код

Схема подключения



Код программы

```
from machine import Pin, ADC, PWM
from time import sleep

led_pin = 15
pot_pin = 26

led = PWM(Pin(led_pin, Pin.OUT))
pot = ADC(Pin(pot_pin))

led.freq(1000)

while True:
  pot_value = pot.read_u16()
  led.duty_u16(pot_value)
  sleep(0.001)
```

В целом представленный код не должен формировать вопросов, при условии проработанности предыдущего материала.

Отметим единственный момент с добавлением задержки в тысячную долю секунды в конце цикла:

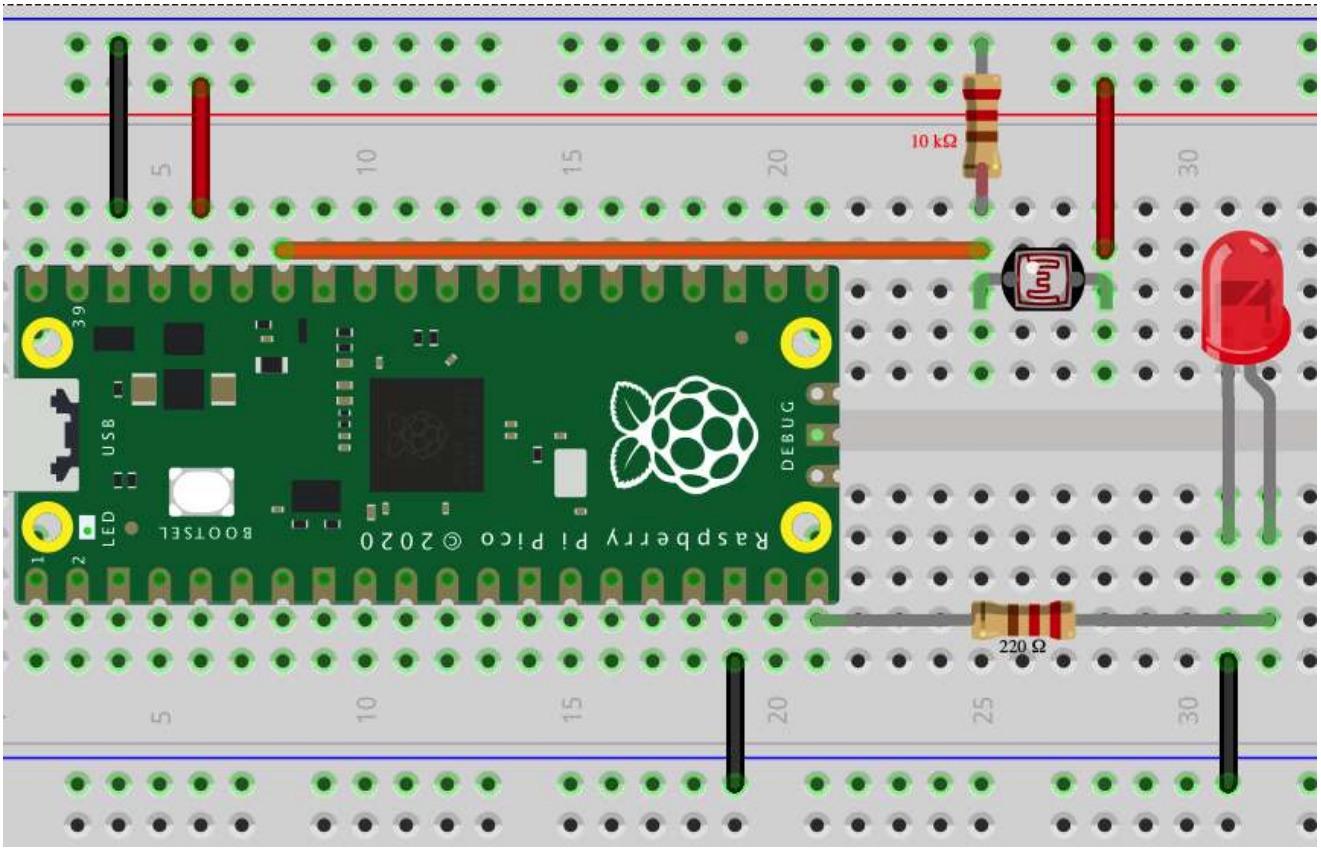
```
sleep(0.001)
```

Принципиально код сохранит свою работоспособность и без нее, однако она позволяет снизить нагрузку на процессор микроконтроллера, сократив дополнительные опросы АЦП и манипуляции с ШИМ.

Управление яркостью светодиода фоторезистором

Схема подключения

Принципиально схема подключения не отличается от схемы с диодом и потенциометром:



Код программы

Рассмотрим несколько вариантов программ:

Наивный вариант

```
from machine import Pin, ADC, PWM
from time import sleep

led_pin = 15
ldr_pin = 28

led = PWM(Pin(led_pin, Pin.OUT))
ldr = ADC(Pin(ldr_pin))

led.freq(1000)

while True:
    ldr_value = ldr.read_u16()
    led.duty_u16(ldr_value)
    print(ldr_value)
    sleep(0.001)
```

Наивность предлагаемого варианта заключается в попытке буквального повторения кода из примера с диодом и потенциометром.

При запуске приведенной программы видно, что формально она выполняет поставленную задачу, но результат весьма далек от ожидания.

Принципиально полученный результат включает в себя две проблемы:

- яркость диода изменяется только в малом промежутке;
- в темноте (при закрытом фоторезисторе) диод тухнет, хотя было бы логичнее чтобы наоборот горел ярче.

Решим программно эти проблемы!

Инвертирование сигнала фоторезистора

```
from machine import Pin, ADC, PWM
from time import sleep

led_pin = 15
ldr_pin = 28

led = PWM(Pin(led_pin, Pin.OUT))
ldr = ADC(Pin(ldr_pin))

led.freq(1000)

while True:
    ldr_value = ldr.read_u16()
    ldr_value = abs(ldr_value - 2 ** 16)
```

```
    print(ldr_value)

    led.duty_u16(ldr_value)
    sleep(0.001)
```

Для инвертирования значений фоторезистора пересчетам считываемое значение:

```
ldr_value = abs(ldr_value - 2 ** 16)
```

Для чего вычтем из считываемого пином значение максимальное значение и возьмем модуль от полученного результата.

После чего диод будет затухать на ярком свете и разгораться в темноте.

> Отметим нерациональность выражения $ldr\_value - 2^{16}$, так как оно требует повторяющегося вычисления значения 2^{16} на каждой итерации цикла.

Оставшуюся проблему ограниченного диапазона яркости решим в несколько этапов.

Первый этап

```
from machine import Pin, ADC, PWM
from time import sleep

led_pin = 15
ldr_pin = 28

UP_BORDER = 55_000
DOWN_BORDER = 28_000

led = PWM(Pin(led_pin, Pin.OUT))
ldr = ADC(Pin(ldr_pin))

led.freq(1000)

while True:
    ldr_value = ldr.read_u16()
    ldr_value = abs(ldr_value - 65535)
    print(ldr_value)
    if ldr_value > UP_BORDER:
        ldr_value = 65535
    elif ldr_value < DOWN_BORDER:
        ldr_value = 0

    led.duty_u16(ldr_value)
    sleep(0.001)
```

Добавленное в цикл ветвление:

```
    if ldr_value > UP_BORDER:
        ldr_value = 65535
    elif ldr_value < DOWN_BORDER:
        ldr_value = 0
```

позволяет полностью выключить или включить светодиод при превышении пороговых значений DOWN_BORDER и UP_BORDER.

В промежутке между ними на ШИМ диода будет передаваться промежуточное между константами значение.

Отметим, что значения констант:

```
UP_BORDER = 55_000
DOWN_BORDER = 28_000
```

подбираются экспериментально и задаются пользователем в теле программы.

> Отметим проблему такого решения, так как подстройка этих значений необходима в зависимости от изменения внешних условий, а непосредственный пользователь не имеет доступа и знаний к изменению исходного кода! Подумайте как можно решить эту проблему!

В любом случае исправление программы помогло лишь отчасти: хотя диод и начал включаться и выключаться управление яркостью диода не происходит по всему диапазону возможных значений.

Продолжим решать эту задачу!

Второй этап

```
from machine import Pin, ADC, PWM
from time import sleep

led_pin = 15
ldr_pin = 28

UP_BORDER = 55_000
DOWN_BORDER = 25_000

MAX_VALUE = 65535

led = PWM(Pin(led_pin, Pin.OUT))
```

```
ldr = ADC(Pin(ldr_pin))

led.freq(1000)

while True:
    ldr_value = ldr.read_u16()
    ldr_value = abs(ldr_value - MAX_VALUE)
    if ldr_value > UP_BORDER:
        ldr_value = MAX_VALUE
    elif ldr_value < DOWN_BORDER:
        ldr_value = 0
    else:
        ldr_value = int(MAX_VALUE * (ldr_value - DOWN_BORDER) / (UP_BORDER
- DOWN_BORDER))
    print(ldr_value)

    led.duty_u16(ldr_value)
    sleep(0.001)
```

Вместо того чтобы напрямую передавать значение между DOWN_BORDER и UP_BORDER на диод, пересчитаем его пропорционально этим значениям, добавив в ветвление выражение:

```
ldr_value = int(MAX_VALUE * (ldr_value - DOWN_BORDER) / (UP_BORDER -
DOWN_BORDER))
```

После чего на ШИМ будет передаваться значение в пределах не от DOWN_BORDER - UP_BORDER, а от нуля до MAX_VALUE.

> Отметим, что повторяющуюся в коде константу 65535 (2^{16}) в этом варианте мы присвоили отдельной переменной MAX_VALUE.

> **Всегда старайтесь избегать явного задания значений внутри Вашего кода!**