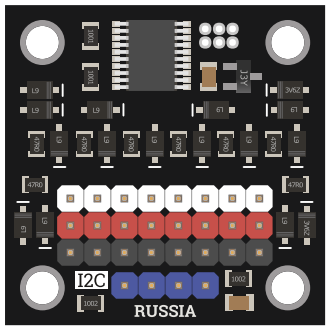
Расширитель выводов, FLASH-I2C (Trema-модуль), подключаем к Raspberry



Общие сведения:

Trema модуль - Расширитель выводов, I2C-flash - является устройством ввода/вывода с подключением по шине I2C. У модуля имеются 8 выводов, каждый из которых может работать в качестве: цифрового входа, цифрового выхода, или аналогового входа. Первые 4 вывода (с номерами 0 - 3) могут работать в качестве выходов с ШИМ, они же позволяют управлять сервоприводами.

Модуль относится к серии «Flash», а значит к одной шине I2C можно подключить более 100 модулей, так как их адрес на шине I2C (по

умолчанию 0x09), хранящийся в энергонезависимой памяти, можно менять программно.

Модуль можно использовать в любых проектах где требуется большое число выводов, как цифровых, так и аналоговых.

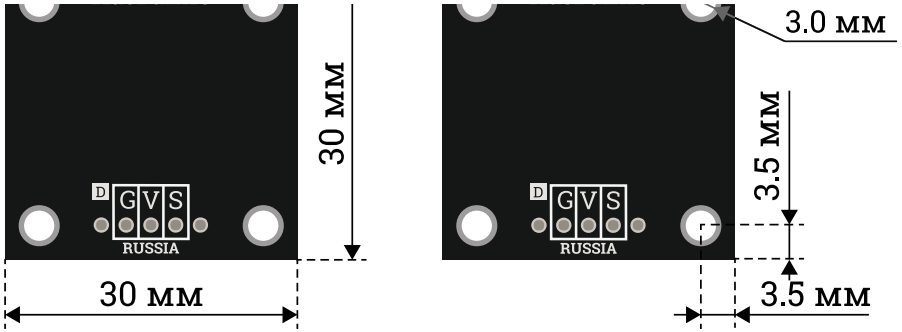
Видео:

Редактируется ...

Спецификация:

- Напряжение питания: 3,3 или 5 В (постоянного тока)
- Потребляемый ток: до 6 мА.
- Напряжение логических уровней: 3,3 В (все выводы толерантны к 5 В).
- Напряжение аналоговых уровней: до 3,3 В (все выводы толерантны к 5 В).
- Разрешение АЦП: 12 бит (значение от 0 до 4095).
- Разрешение ШИМ: 12 бит (значение от 0 до 4095).
- Частота ШИМ: 1 - 12'000 Гц (по умолчанию 490 Гц).
- Количество цифровых выводов: 8 (работают как на вход, так и на выход).
- Количество аналоговых входов: 8 (АЦП).
- Количество выходов с поддержкой ШИМ: 4 (выводы № 0 - 3).
- Интерфейс: I2C.
- Скорость шины I2C: 100 кбит/с.
- Адрес на шине I2C: устанавливается программно (по умолчанию 0x09).
- Уровень логической 1 на линиях шины I2C: 3,3 В (толерантны к 5 В).
- Рабочая температура: от -40 до +65 °C.
- Габариты: 30 х 30 мм.
- Вес: 7 г.

Все модули линейки "Trema" выполнены в одном формате



Подключение:

По умолчанию все модули FLASH-I2C имеют установленный адрес 0x09.
— Перед подключением 1 модуля к шине I2C **настоятельно рекомендуется** изменить адрес модуля.
— При подключении 2 и более FLASH-I2C модулей к шине необходимо **в обязательном порядке предварительно изменить адрес каждого модуля**, после чего уже подключать их к шине.
Более подробно о том, как это сделать, а так же о многом другом, что касается работы FLASH-I2C модулей, вы можете прочесть в **этой статье**.

Модуль подключается по шине **I2C**, все выводы которой (GND, Vcc, SDA, SCL) размещены на одной колодке модуля.

- **SCL** - вход/выход линии тактирования шины I2C.
- **SDA** - вход/выход линии данных шины I2C.
- **Vcc** - вход питания 3,3 или 5 В.
- **GND** - общий вывод питания.

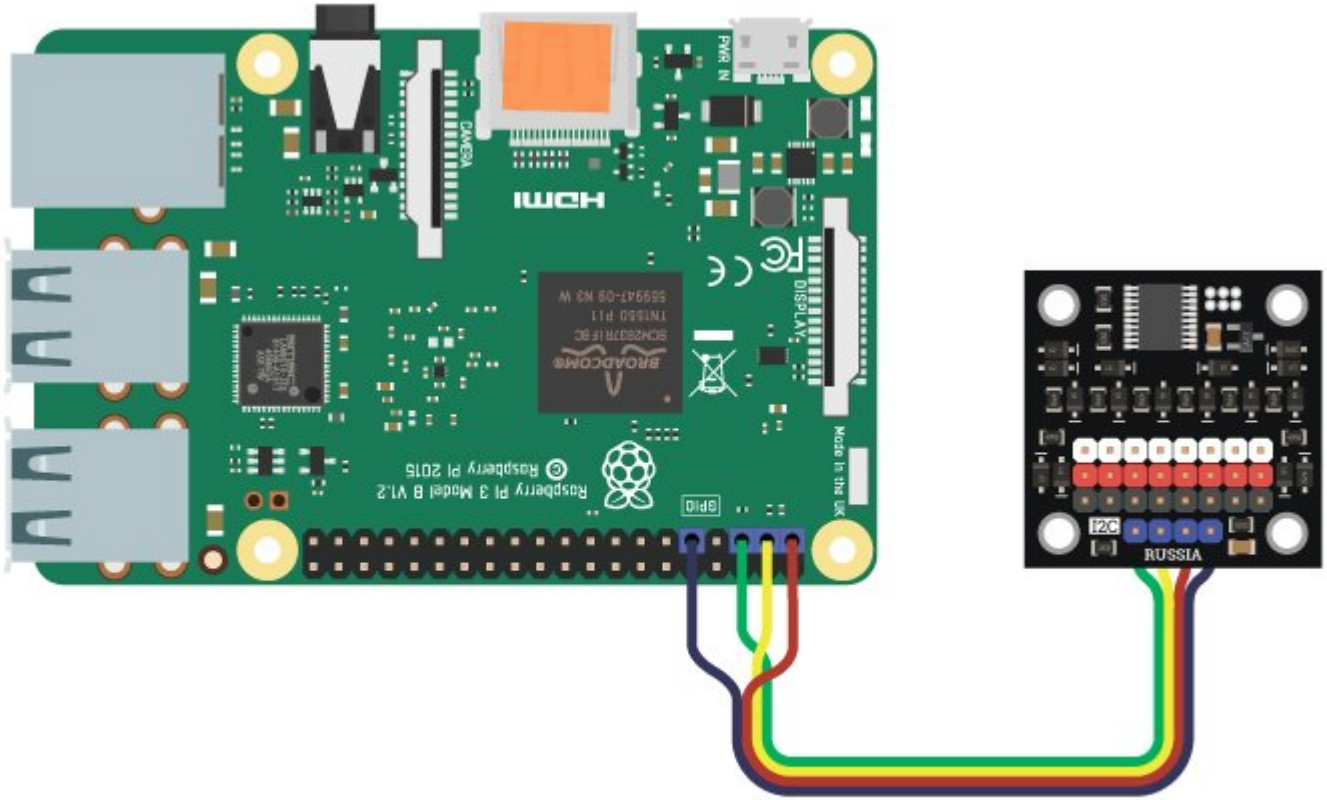
Для удобства подключения, предлагаем воспользоваться Trema+Expander Hat.

Модуль удобно подключать 2 способами, в зависимости от ситуации:

Способ - 1: Используя провода и Raspberry Pi

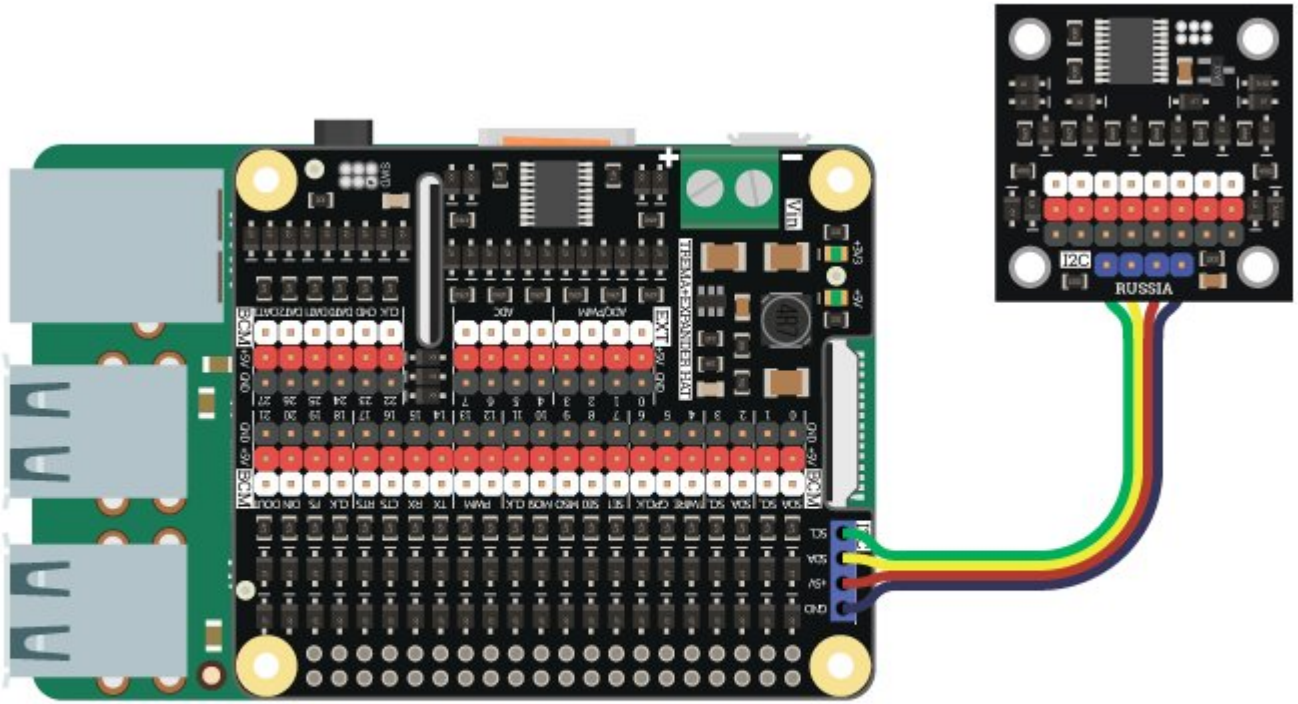
Используя провода «Мама — Мама», подключаем напрямую к Raspberry Pi

В этом случае необходимо питать логическую часть модуля от 3,3 В Raspberry



Способ - 2: Используя Trema+Expander Hat

Используя 4-х проводной шлейф, подключаем к Trema+Expander Hat.



Питание:

Входное напряжение питания модуля 3,3В или 5В постоянного тока (поддерживаются оба напряжения питания), подаётся на выводы Vcc и GND.

Питание:

Входное напряжение питания модуля 3,3В или 5В постоянного тока (поддерживаются оба напряжения питания), подаётся на выводы Vcc и GND.

Подробнее о модуле:

Модуль построен на базе микроконтроллера STM32F030F4 и снабжен собственным стабилизатором напряжения. У модуля имеются 8 выводов GPIO размещённых на белой колодке. Каждый вывод пронумерован с обеих сторон платы. Рядом с выводами GPIO есть два вывода питания (Vcc - красная колодка и GND - чёрная колодка). Все выводы GPIO могут работать как цифровые входы/выходы, так и в качестве аналоговых входов, а выводы 0-3 поддерживают вывод сигналов ШИМ на аппаратном уровне. На каждом выводе GPIO установлена схема защиты микроконтроллера от напряжений выше 3,3 В.

Модуль позволяет:

- Менять свой адрес на шине I2C.
- Считывать или задавать логические уровни на любом выводе GPIO.
- Считывать аналоговый уровень (12 бит АЦП) с любого вывода GPIO.
- Задавать сигнал ШИМ с указанным коэффициентом заполнения (12 бит) на первых 4 выводах.
- Задавать частоту ШИМ от 1 до 12'000 Гц (по умолчанию 490 Гц).
- Внутрисхемно подключать подтягивающие или прижимающие резисторы для каждого вывода.
- Выбирать внутреннюю схему работы выхода (двухтактная / с общим стоком).
- Управлять сервоприводами подключёнными к выводам GPIO 0 - 3.

Примеры:

Специально для работы с Trema модулем - Расширитель выводов, I2C-flash , нами разработана библиотека pyiArduinoI2Cexpander которая позволяет реализовать все функции модуля.

Для работы с модулем необходимо включить шину I2C.
Ссылка на подробное описание как это сделать.

Внимание! Для корректной работы модулей FLASH-I2C на Raspberry Pi под управлением Raspberry OS "Buster" необходимо выключить динамическое тактирование ядра (опция **core_freq_min** должна быть равна **core_freq** в `/boot/config.txt`) Ссылка на подробное

описание.

Для подключения библиотеки необходимо сначала её установить. Сделать это можно в менеджере модулей в Thonny Python IDE (тогда библиотека установится локально для этого IDE), в виртуальной среде командой `pip3 install pyiArduinoI2Csht` или в терминале Raspberry (тогда библиотека будет системной) командой:

```
sudo pip3 install pyiArduinoI2Cexpander
```

Подробнее об установке библиотек можно узнать в этой статье.

Смена адреса модуля на шине I2C:

```
# Подключаем библиотеку для работы с расширителем выводов
from pyiArduinoI2Cexpander import *
import sys

# Создаём объект module для работы с функциями и методами библиотеки pyiArduinoI2Cexpander.
# Если при объявлении объекта указать адрес, например, module(0x0B),
# то пример будет работать с тем модулем, адрес которого был указан.
module = pyiArduinoI2Cexpander(None, NO_BEGIN)

# Если сценарию не были переданы аргументы
if len(sys.argv) < 2:
    # Назначаем модулю адрес (0x07 < адрес < 0x7F).
    newAddress = 0x09

# Иначе
else:
    # Новый адрес - первый аргумент
    newAddress = int(sys.argv[1])
```

```
# Если модуль найден
if module.begin():
    print("Найден модуль %#.2x" % module.getAddress())

    # Если адрес удалось изменить
    if module.changeAddress(newAddress):
            print("Адрес изменён на %#.2x" % module.getAddress())

    else:
            print("Адрес не изменён!")

else:
    print("Датчик не найден!")
```

Для работы данного примера, на шине I2C должен быть только один расширитель выводов.

Данный скрипт демонстрирует не только возможность смены адреса на указанный в переменной `newAddress`, но и обнаружение, и вывод текущего адреса модуля на шине I2C.

Работа с логическими уровнями:

```
from pyiArduinoI2Cexpander import *      # Подключаем модуль для работы с расширителем выводов.
from time import sleep                   # Импортируем фунуцию ожидания из модуля времени
exp = pyiArduinoI2Cexpander(0x09)        # Создаём объект exp для работы с функциями модуля pyiArduinoI2Cexpander, указывая адр
                                         # Если создать объект без указания адреса ( exp = pyiArduinoI2Cexpander() ), то адрес
exp.pinMode(0, INPUT,  DIGITAL)          # Конфигурируем вывод 0 на работу в качестве цифрового входа.
# exp.pinPull(0, PULL_UP)                # Подтягиваем вход 0 до уровня VCC через внутренний резистор (для работы с кнопкой).
# exp.pinPull(0, PULL_DOWN)              # Прижимаем   вход 0 к  уровню GND через внутренний резистор (для работы с кнопкой).
exp.pinMode(1, OUTPUT, DIGITAL)          # Конфигурируем вывод 1 на работу в качестве цифрового выхода.
                                         #
whiel True:                              #
    level = exp.digitalRead(0)           #   Читаем логический уровень с вывода №0 в переменную «level».
```

```
    exp.digitalWrite(1, level)          #   Устанавливаем на выводе №1 уровень равный значению «level».
    sleep(.1)                           #
```

Данный пример устанавливает на 1 выводе логический уровень равный логическому уровню поступившему на вывод 0. Если логический уровень на выводе 0 создаётся кнопкой, то этот вывод можно подтянуть до уровня Vcc или прижать к уровню GND, в зависимости от схемы подключения кнопки.

Работа с аналоговыми уровнями:

```
from pyiArduinoI2Cexpander import *     # Подключаем модуль для работы с расширителем выводов.
from time import sleep                  # Импортируем фунуцию ожидания из модуля времени
exp = pyiArduinoI2Cexpander(0x09)       # Создаём объект exp для работы с функциями модуля pyiArduinoI2Cexpander, указывая адр
                                        # Если создать объект без указания адреса ( exp = pyiArduinoI2Cexpander() ), то адрес
exp.pinMode(0, INPUT,  ANALOG)          #   Конфигурируем вывод 0 на работу в качестве аналогового входа.
exp.pinMode(1, OUTPUT, ANALOG)          #   Конфигурируем вывод 1 на работу в качестве аналогового выхода.
                                        #
while True:                             #
    level = exp.analogRead(0)           #   Читаем аналоговый уровень с вывода №0 в переменную «level».
    exp.analogWrite(1, level)           #   Устанавливаем на выводе №1 уровень равный значению «level».
    sleep(.1)                           #
```

Данный пример устанавливает на 1 выводе сигнал ШИМ, уровень которого прямо пропорционален напряжению подведённому к выводу 0. Удобство данного примера заключается в том, что диапазон считанных аналоговых уровней (0-4095) совпадает с диапазоном устанавливаемых значений ШИМ (0-4095).

Чтение логического уровня с аналогового входа:

```
from pyiArduinoI2Cexpander import *     # Подключаем модуль для работы с расширителем выводов.
from time import sleep                  # Импортируем фунуцию ожидания из модуля времени
exp = pyiArduinoI2Cexpander(0x09)       # Создаём объект exp для работы с функциями модуля pyiArduinoI2Cexpander, указывая адр
```

```
                                    # Если создать объект без указания адреса ( exp = pyiArduinoI2Cexpander() ), то адрес
                                    #
exp.pinMode(0, INPUT,  ANALOG)      # Конфигурируем вывод 0 на работу в качестве аналогового входа.
exp.pinMode(1, OUTPUT, DIGITAL)     # Конфигурируем вывод 1 на работу в качестве цифрового выхода.
exp.levelWrite(512)                 # Указываем аналоговый уровень разделяющий уровень логического 0 от 1.
exp.levelHyst(12)                   # Указываем гистерезис аналогового уровня в ±12.
                                    #
while True:                         #
    level = exp.levelRead(0)        # Читаем логический уровень с аналогового вывода №0 в переменную «level».
    exp.digitalWrite(1, level)      # Устанавливаем на выводе №1 уровень равный значению «level».
    sleep(.1)                       #
```

Данный пример преобразует аналоговый уровень со входа 0 в логический уровень на выходе 1. Главной особенностью данного примера является то, что exp.levelWrite(512) задаёт границу по которой модуль определяет разницу между уровнями логической 1 и логического 0 на аналоговом входе (все аналоговые значения выше 512+12 будут расценены как логическая 1, все значения ниже 512-12 будут расценены как 0, а значения 512-12...512+12 вернут предыдущий логический уровень). Таким образом модуль способен считывать данные со схем, логические уровни которых ниже 3,3 В.

Управление сервоприводами:

```
from pyiArduinoI2Cexpander import *     # Подключаем модуль для работы с расширителем выводов.
from time import sleep                  # Импортируем фунуцию ожидания из модуля времени
exp = pyiArduinoI2Cexpander(0x09)       # Создаём объект exp для работы с функциями модуля pyiArduinoI2Cexpander, указывая адр
                                        # Если создать объект без указания адреса ( exp = pyiArduinoI2Cexpander() ), то адрес
                                        #
exp.pinMode(0, OUTPUT, SERVO)           # Конфигурируем вывод 0 на работу в качестве выхода для сервопривода.
exp.pinMode(1, OUTPUT, SERVO)           # Конфигурируем вывод 1 на работу в качестве выхода для сервопривода.
# exp.servoAttach(0, 500, 2500, 0, 180) # Определяем параметры сервопривода: вывод 0, длительность импульсов от 500 до 2500 мс
# exp.servoAttach(1, 513, 2430, 0, 180) # Определяем параметры сервопривода: вывод 1, длительность импульсов от 513 до 2430 мс
                                        #
whiel True:                             #
```

```
    exp.servoWrite(0, 170)              # Поворачиваем сервопривод подключённый к выводу 0 в угол 170°,
    exp.servoWrite(1, 100)              # а подключённый к выводу 1 в угол 100° и
    sleep(.5)                           # ждём пол секунды.
                                        #
    exp.servoWrite(0, 10)               # Поворачиваем сервопривода (на выводе 0) в угол 10°,
    exp.servoWrite(1, 10)               # поворачиваем сервопривода (на выводе 1) в угол 10°,
    sleep(.5);                          # ждём пол секунды.
```

Данный пример поворачивает один сервопривод на угол 170°, а второй на угол 100°, ждёт пол секунды и возвращает оба сервопривода в угол 0°, через пол секунды всё повторяется. Если Ваш сервопривод имеет угол поворота не 180°, а 60°, 90°, 270°, 360° и т.д., или он не точно поворачивается, или Вы желаете ограничить угол его поворота, то Вы можете настроить его параметры, указав крайние углы поворота и длительность импульсов ШИМ для них при помощи функции servoAttach().

Описание функций библиотеки:

Подключение модуля:

```
from pyiArduinoI2Cexpander import *
```

Создание объекта:

```
exp = pyiArduinoI2Cexpander(АДРЕС)
```

Функция begin()

- Назначение: инициализация расширителя выводов
- Синтаксис: `exp.begin()`
- Параметры: нет
- Возвращаемые значения: флаг инициализации
- Примечание: функция не обязательная, выполняется автоматически при создании объекта. Можно использовать для проверки наличия устройства на шине.

- Пример:

```
if exp.begin():
    print("устройство найдено и инициализировано")
else:
    print("устройство не найдено, проверьте включена ли шина I2C")
```

Функция changeAddress()

- Назначение: смена адреса устройства
- Синтаксис: `exp.changeAddress(newAddr)`
- Параметры: **newAddr** - новый адрес для устройства
- Возвращаемые значения: флаг выполнения. 1 - успешно, 0 - неуспешно.
- Примечание: нет
- Пример:

```
exp.changeAddress(0x0A)
```

Функция reset()

- Назначение: перезагрузка устройства
- Синтаксис: `exp.reset()`
- Параметры: нет
- Возвращаемые значения: флаг выполнения
- Примечание: нет
- Пример:

```
exp.reset()
```

Функция getAddress()

- Назначение: узнать текущий адрес устройства на шине I2C
- Синтаксис: `exp.getAddress()`
- Параметры: нет
- Возвращаемые значения: текущий адрес модуля на шине I2C
- Примечание: нет
- Пример:

```
addr = exp.getAddress()
```

Функция getVersion()

- Назначение: узнать текущую версию прошивки модуля
- Синтаксис: `exp.getVersion()`
- Параметры: нет
- Возвращаемые значения: текущая версия прошивки модуля
- Примечание: нет
- Пример:

```
ver = exp.getVersion()
```

Функция pinMode()

- Назначение: конфигурирование выводов
- Синтаксис: `exp.pinMode(pin, dir, type)`
- Параметры:
  - **pin** - номер вывода (0-7)
  - **dir** - направление работы вывода (**INPUT**, **OUTPUT**)

    - **type** - тип сигнала (**ANALOG**, **DIGITAL**, **SERVO**)
- Возвращаемые значения:нет
- Примечание: функция не обязательная, выполняется автоматически при вызове функций, связанных с чтением или установкой уровней на выводах. Можно использовать, если необходимо чтоб вышеупомянутые функции выполнялись быстрее в первый раз в коде.
- Пример:

```
pot = 0
led = 1
servo = 3
exp.pinMode(pot, INPUT, ANALOG)
exp.pinMode(led, OUTPUT, DIGITAL)
exp.pinMode(servo, OUTPUT, SERVO)
```

Функция pinPull()

- Назначение: подключение к выводу прижимающего или подтягивающего резистора
- Синтаксис: `exp.pinMode(pin, pull)`
- Параметры:
    - **pin** - номер вывода (0-7)
    - **pull** - резистор (**PULL_UP**, **PULL_DOWN**, **PULL_NO**)
- Возвращаемые значения:нет
- Примечание: нет
- Пример:

```
button_pin = 0
exp.pinPull(button_pin, PULL_UP)
```

Функция pinOutScheme()

- Назначение: выбор схемы выхода

- Синтаксис: `exp.pinOutScheme(pin, mode)`
- Параметры:
  - **pin** - номер вывода (0-7)
  - **mode** - схема (**OUT_PUSH_PULL**, **OUT_OPEN_DRAIN**)
- Возвращаемые значения:нет
- Примечание: нет
- Пример:

```
exp.pinOutScheme(0, OUT_PUSH_PULL)
exp.pinOutScheme(1, OUT_OPEN_DRAIN)
```

Функция digitalWrite()

- Назначение: установка логического уровня
- Синтаксис: `exp.digtalWrite(pin, level)`
- Параметры:
  - **pin** - номер вывода (0-7)
  - **level** - логический уровень (0, 1)
- Возвращаемые значения:нет
- Примечание: нет
- Пример:

```
led_pin = 1
exp.digitalWrite(led_pin, HIGH)
```

Функция digitalRead()

- Назначение: чтение логического уровня
- Синтаксис: `exp.digitalRead(pin)`

- Параметры: **pin** - номер вывода (0-7)
- Возвращаемые значения: логический уровень вывода
- Примечание: нет
- Пример:

```
button_pin = 0
exp.digitalRead(button_pin)
```

Функция analogWrite()

- Назначение: установка аналогового уровня
- Синтаксис: `exp.analogWrite(pin, level)`
- Параметры:
  - **pin** - номер вывода (0-7)** **
  - **level** - аналоговый уровень (0-4095)
- Возвращаемые значения:нет
- Примечание: нет
- Пример:

```
led_pin = 1
exp.analogWrita(led_pin, 2048)
```

Функция analogRead()

- Назначение: чтение аналогового уровня
- Синтаксис: `exp.analogRead(pin)`
- Параметры: **pin** - номер вывода (0-7)
- Возвращаемые значения: аналоговый уровень (0-4095)
- Примечание: нет

- Пример:

```
pot_pin = 3
val = exp.analogRead(pot_pin)
```

Функция analogAveraging()

- Назначение: установка коэффициента усреднения показаний АЦП
- Синтаксис: `exp.analogAveraging(coef)`
- Параметры: **coef** - коэффициент усреднения (0-255)
- Возвращаемые значения: нет
- Примечание: нет
- Пример:

```
exp.analogAveraging(255)
```

Функция levelWrite()

- Назначение: установка аналогового уровня для функции levelRead()
- Синтаксис: `exp.levelWrite(level)`
- Параметры: **level** - аналоговый уровень, разделяющий логический 0 и 1 (0-4095)
- Возвращаемые значения:нет
- Примечание: нет
- Пример:

```
exp.levelWrite(512)
```

Функция levelRead()

- Назначение: чтение логического уровня с аналогового вывода
- Синтаксис: `exp.levelRead()`
- Параметры: **pin** - номер вывода (0-7)
- Возвращаемые значения: логический уровень (0, 1)
- Примечание: нет
- Пример:

```
pot_pin = 3
exp.levelWrite(512)
val = exp.levelRead(pot_pin)
```

Функция levelHyst()

- Назначение: установка гистерезиса для функции levelRead()
- Синтаксис: `exp.levelHyst(hyst)`
- Параметры: **hyst** - гистерезис (0-4094)
- Возвращаемые значения: нет
- Примечание: нет
- Пример:

```
exp.levelHyst(12)
```

Функция freqPWM()

- Назначение: установка частоты ШИМ
- Синтаксис: `exp.freqPWM(freq)`
- Параметры: **freq** - частота ШИМ в кГц (0-12000)
- Возвращаемые значения:нет
- Примечание: нет

- Пример:

```
exp.freqPWM(440)
```

Функция servoAttach()

- Назначение: конфигурирование вывода для сервопривода
- Синтаксис: `exp.servoAttach(pin, width_min, width_max, angle_min, angle_max)`
- Параметры:
  - **pin** - номер вывода (0-7)
  - **width_min** - минимальная ширина импульса, мкс (0-20000)
  - **width_max** - максимальная ширина импульса, мкс (0-20000)
  - **angle_min** - угол при минимальной ширине, градусы (0-360°)
  - **angle_max** - угол при максимальной ширине, градусы (0-360°)
- Возвращаемые значения:нет
- Примечание: нет
- Пример:

```
servo_pin = 0
exp.pinMode(servo_pin, OUTPUT, SERVO)
exp.servoAttach(servo_pin, 500, 2500, 0, 180)
```

Функция servoWrite()

- Назначение: установка угла поворота сервопривода
- Синтаксис: `exp.servoWrite(pin, angle)`
- Параметры:
  - **pin** - номер вывода (0-3)
  - **angle** - угол поворота, градусы(0-360°)

- Возвращаемые значения: нет
- Примечание: нет
- Пример:

```
servo_pin = 0
exp.servoWrite(servo_pin, 90)
```

Функция servoWriteMicroseconds()

- Назначение: установка ширины импульсов для сервопривода
- Синтаксис: `exp.servoWriteMicroseconds(pin, width)`
- Параметры:
  - **pin** - номер вывода (0-3)
  - **width** - ширина импульсов, мкс (0-20000)
- Возвращаемые значения: нет
- Примечание: нет
- Пример:

```
servo_pin = 0
exp.servoWriteMicroseconds(servo_pin, 1500)
```

Ссылки:

-
- Библиотека pyiArduinoI2Cexpander.
- Wiki - Установка библиотек Python