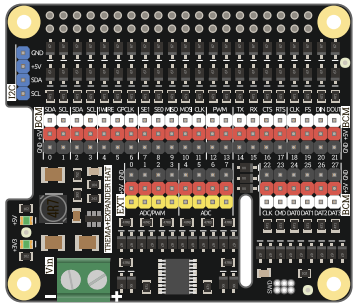
Trema+Expander Hat

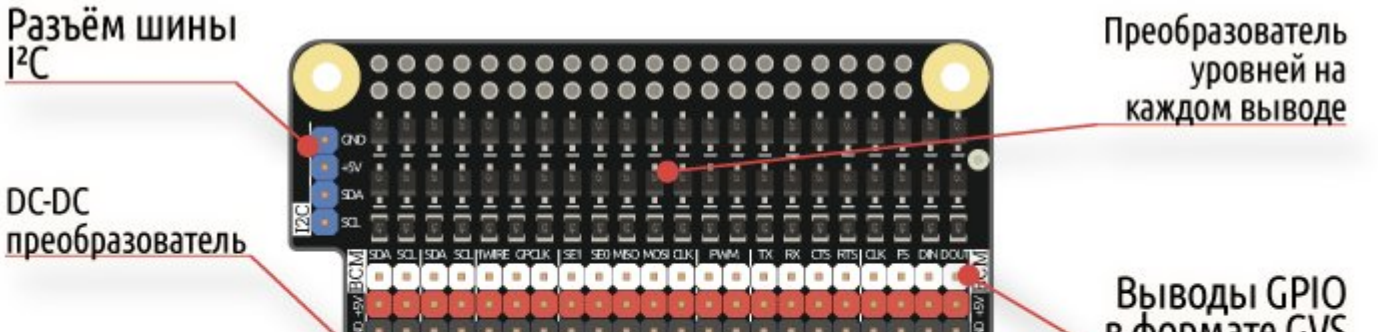


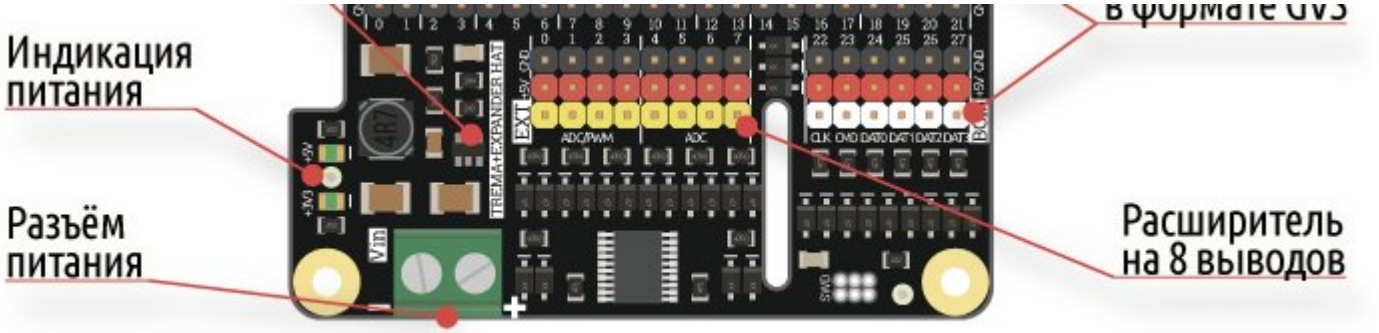
Общие сведения:

Trema+Expander Hat - это плата упрощающая процесс подключения устройств к Raspberry Pi, имеющая на борту цепь питания, все выводы GPIO Raspberry и добавляющая выводы с АЦП и аппаратной ШИМ. Все выводы выполнены в формате колодки GVS с питанием 5 вольт, добавлена совместимость с 5-ти вольтовой логикой, что упрощает процесс подключения устройств, таких как датчики, сервоприводы, потенциометры и т. д. Плата может питаться как от колодки GPIO Raspberry, так и от внешнего источника питания, при этом в последнем случае она сама будет питать Raspberry Pi и устройства, подключённые к колодкам.

Спецификация:

- Диапазон напряжений на разъёме Vin: 6 ... 12 В
- Выходное напряжение: 5 В
- Выходной ток: до 2,5 А при Vin 9 В
- Интерфейс: I2C
- Колодки: BCM, EXT, I2C
- 8 выводов расширения с АЦП через I2C (4 поддерживают ШИМ, колодка EXT)
- Диапазон аналоговых уровней входов: 0 ... 4 В;
- Адрес на шине I2C: изменяемый (по умолчанию 0x08)
- Габариты: Hat

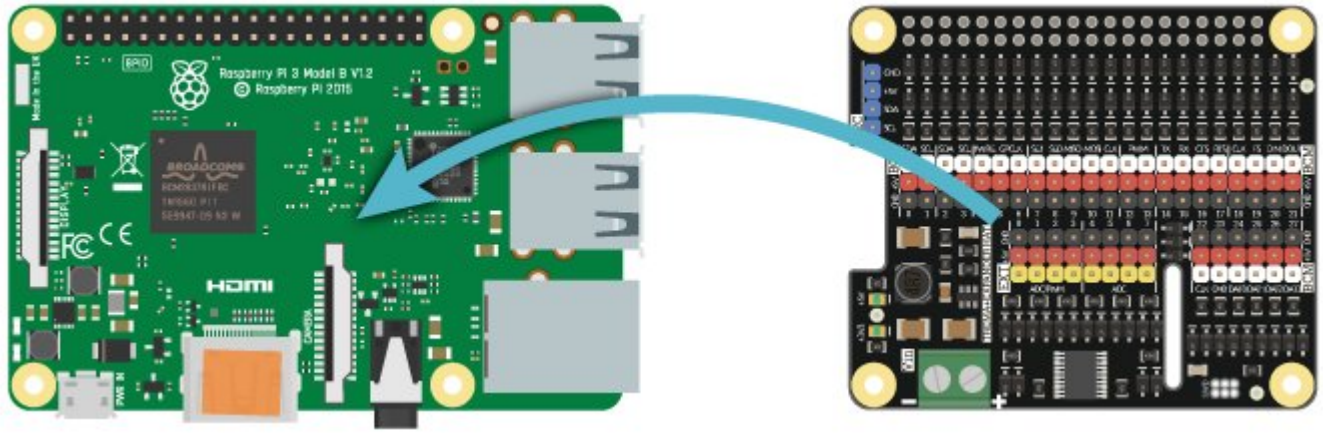




Подключение:

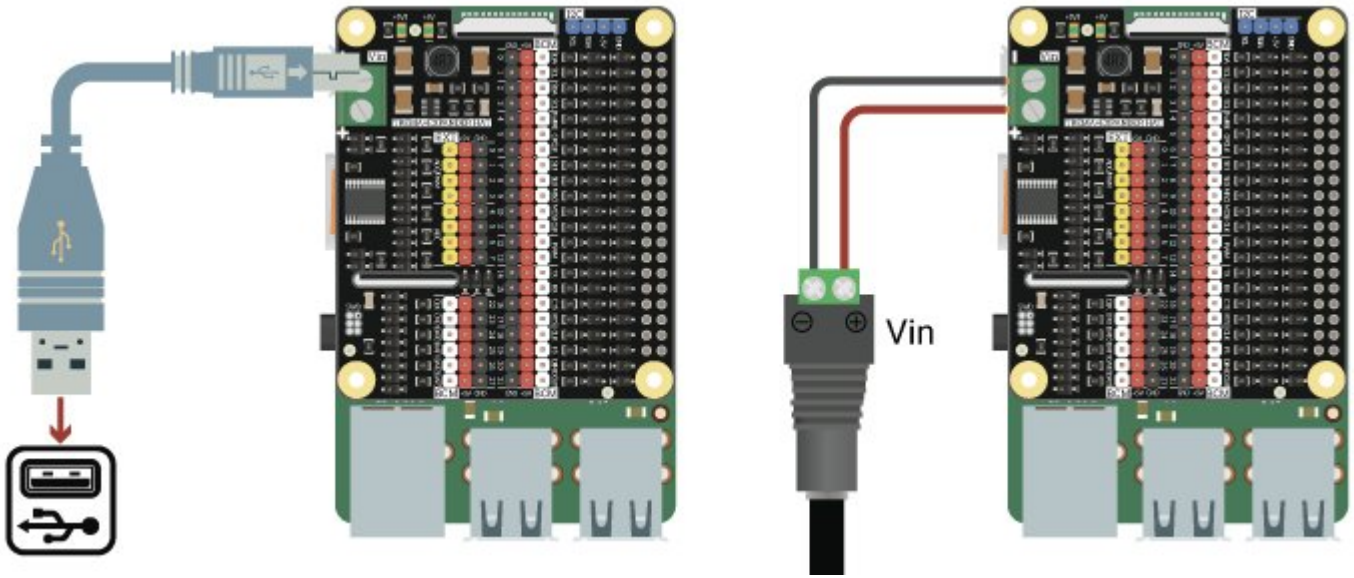
Для работы с модулем необходимо включить шину I2C.
Ссылка на подробное описание как это сделать.

Trema+Expander Hat разработан специально для одноплатных компьютеров семейства Raspberry Pi



Trema+Expander Hat устанавливается на Raspberry Pi, а модули подключаются проводами к колодкам Trema+Expander Hat.

Питание:



Плата может питаться как от Raspberry Pi, так и питать Raspberry Pi от внешнего источника. Для этого на плате имеется DC-DC преобразователь, работающий в диапазоне напряжений от 5,5 до 12 вольт, при этом напряжение питания на колодках всегда будет 5 вольт. Так же это даёт возможность подключения большего количества модулей.

Подробнее о Trema+Expander Hat:

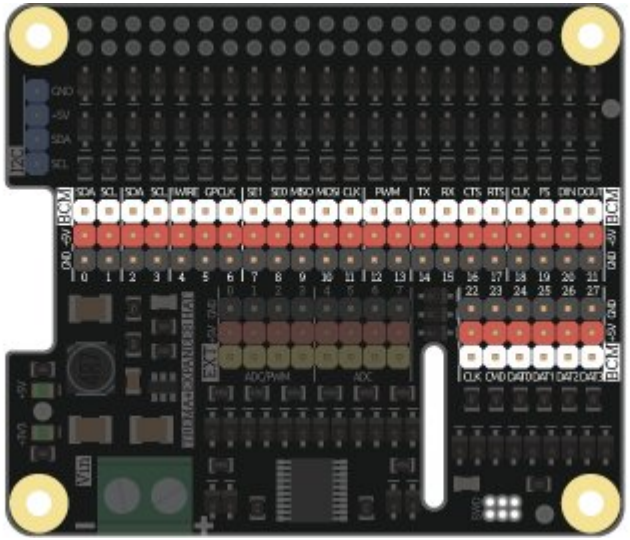
На плате Trema+Expander Hat имеются:

- **Колодка BCM** из 28 цифровых выводов GPIO с выводами шины питания (VCC и GND), для подключения цифровых модулей
- **Колодка EXT** из 8 выводов расширения:
  - Все 8 можно использовать их как аналоговые входы, что позволит подключить аналоговые датчики
  - Все 8 можно использовать как цифровые входы/выходы
  - Первые 4 можно использовать как выходы с широтно-импульсной модуляцией (ШИМ)
- **Колодка I2C** аппаратной шины I2C из 4 выводов (SDA, SCL, GND, Vcc) для подключения I2C модулей
- Два светодиода, 3V3 и 5V - информирует о наличии питания
- Преобразователи напряжений на каждом выводе GPIO, что делает возможным использовать модули, работающие от 5 вольт с Raspberry.
- **Цепь питания** 5 вольт для Raspberry и устройств, подключаемых к колодкам.

На плате Trema+Expander Hat возле каждого информационного вывода находятся два вывода питания (VCC и GND) значит количество подключённых модулей, без пайки и «скрутки» проводов, может совпадать с количеством GPIO выводов Raspberry.

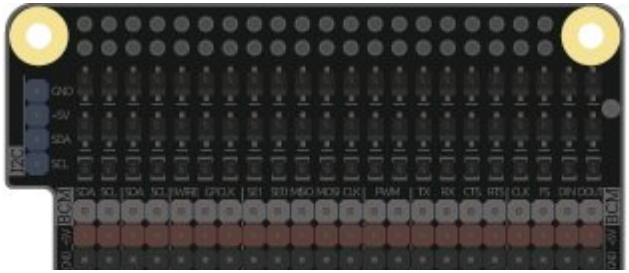
Помимо обычных выводов, на плате Trema+Expander Hat имеется колодка аппаратной шины I2C для подключения I2C модулей. Если Вам нужно подключить несколько модулей к шине I2C можно воспользоваться Trema I2C Hub.

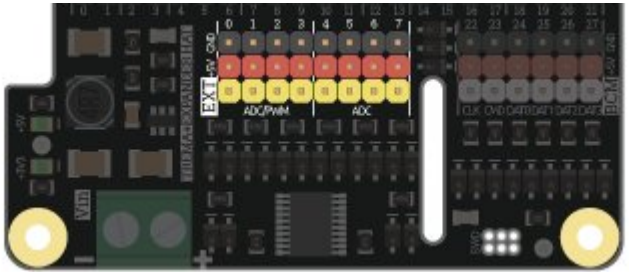
Колодка BCM:



Все 28 выводов GPIO Raspberry Pi выведены на удобную колодку в формате GVS. Каждый вывод согласован с 5-ти вольтовой логикой, поэтому к выводам можно подключать модули как с 3-х, так и с 5-ти вольтовой логикой. Напряжение питания колодки 5 В.

Колодка EXT:





Главной отличительной особенностью Trema+Expander Hat является наличие восьми выводов расширения, четыре из которых поддерживают ШИМ и все восемь могут быть дополнительными цифровыми входами/выходами или аналоговыми входами. Каждый вывод согласован с 5-ти вольтовой логикой, поэтому к выводам можно подключать модули как с 3-х, так и с 5-ти вольтовой логикой. Напряжение питания колодки 5 В.

Для использования выводов расширения Trema+Expander Hat необходимо установить модуль для Python под названием **pyiArduinoI2Cexpander**. Сделать это можно в редакторе Thonny Python IDE в меню `Tools -> Manage Packages...` или в эмуляторе терминала издав команду `sudo pip3 install pyiArduinoI2Cexpander` для установки для всех пользователей или `pip3 install pyiArduinoI2Cexpander` для установки для текущего пользователя или в виртуальную среду.

Подробнее про установку модулей Python

Аналоговый сигнал

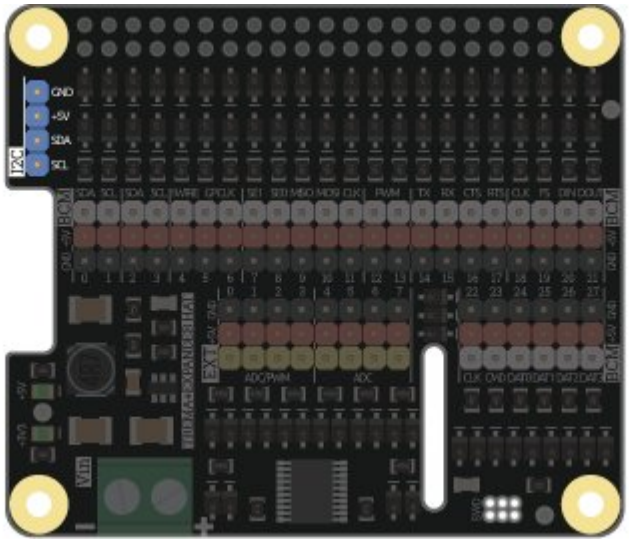
Аналоговые входы, пригодятся для подключения аналоговых датчиков, такие как Датчик Холла, Потенциометр, Датчик освещённости, Аналоговый термометр и т.д. Так же выводы 0, 1, 2, 3 поддерживают 12-ти битную ШИМ и сервоприводы

Как и выводы колодки BCM, выводы колодки EXT согласованы с пятивольтовыми уровнями. Диапазон напряжений аналоговых входов при котором значения АЦП имеют линейную зависимость — от 0 до 3 В. Если аналоговый датчик использует полный диапазон значений до 5 вольт, то верхняя часть от 3 до 4 вольт будет иметь логарифмическую зависимость, а после 4 вольт воспринята АЦП как максимальное значение.

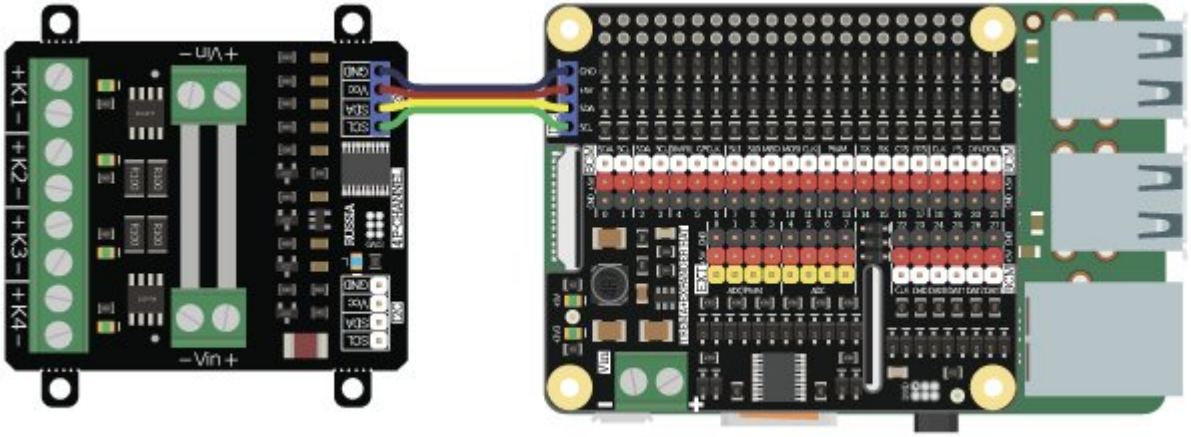
Цифровой сигнал

Цифровые входы/выходы так же могут быть использованы для подключения различных устройств, таких как Кнопка, Зуммер , Датчик наклона и т.д.

Колодка I2C:



Пример подключения внешних устройств:



Колода I2C подойдет для подключения внешних устройств, например очень удобно с помощью 4-проводного шлейфа «мама-мама» подключить модуль реле, и т.д. Напряжение питания колодки 5 В.

Примеры использования выводов расширения Trema+Expander Hat:

Для использования выводов расширения установите модуль Python **pyiArduinoI2Cexpander.**

Ссылка на подробную статью об установке модулей Python

Пример установки нового адреса:

```
from pyiArduinoI2Cexpander import *      # Подключаем модуль для работы с расширителем выводов.
import sys                               # Импортируем модуль системных команд
from time import sleep                   # Импортируем функцию ожидания из модуля времени
ext = pyiArduinoI2Cexpander()            # Создаём объект ext для работы с функциями модуля pyiArduinoI2Cexpander, указывая адр
                                         #
if len(sys.argv) < 2:                    # Если скрипту не были переданы аргументы
                                         #
    print("Недостаточно аргументов."     # Выводим текст в stdout
          "Для работы с программой"      #
          " введите: python3 "           #
          + sys.argv[0]                  #
          + " <новый_адрес_модуля>")     #
                                         #
elif ext.begin():                        # Иначе,
                                         # Проверяем наличие расширителя на шине I2C
    try:                                 # Входим в try блок
                                         #
        newAddress = int(sys.argv[1])    # Преобразуем значение введённого
                                         # аргумента в целое число
```

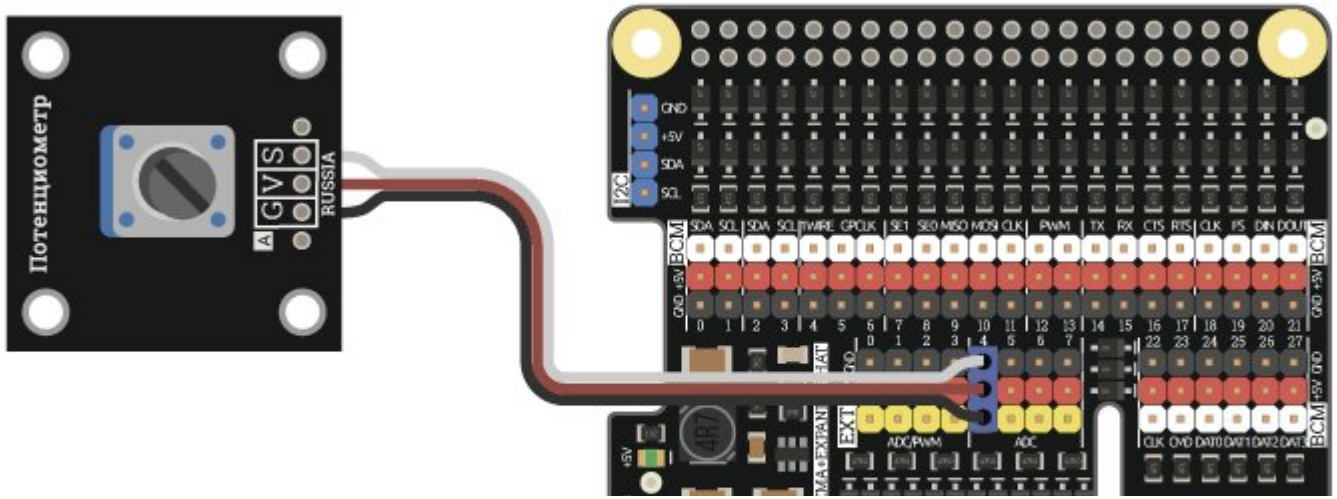
```
    except ValueError:                  # Если преобразование не удалось
                                        #
        print("адрес должен быть "     # информируем пользователя.
              "целым числом "           #
              "в десятичной системе")   #
    else:                               #
        print("На шине I2C найден "    # Иначе
              "модуль с адресом "       #
              "%#.2x который является"  # форматируем строку,
              " расширителем выводов"   #
              % ext.getAddress())       # выводим текущий адрес модуля.
                                        #
        if ext.changeAddress(           # Если удалось изменить
                    newAddress):        # адрес устройства на newAddress,
                                        #
              print("Адрес модуля"      # информируем пользователя,
                    " изменён на "      #
                    "%#.2x"             # форматируем строку,
                    % ext.getAddress()) # выводим текущий адрес модуля.
                                        #
        else:                           #
                                        #
          print("Адрес модуля "         #
                "изменить не удалось!") #
                                        #
else:                                   #
                                        #
    print("Расширитель "                #
          "выводов не найден!")         #
```

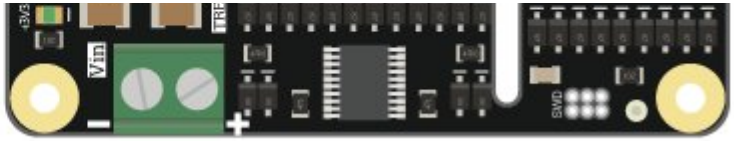
Данный скрипт меняет адрес расширителя выводов Trema+Expander Hat на шине I2C на указанный в стоке при запуске скрипта.

Например, `python3 NewAddress.py 10`

Пример считывания аналогового значения:

```
                                    # $ Строки со знаком $ являются необязательными.
                                    #
from pyiArduinoI2Cexpander import *     #   Подключаем библиотеку для работы с расширителем выводов.
from time import sleep                  #   Подключаем функцию sleep из модуля time
ext = pyiArduinoI2Cexpander(0x08)       #   Объявляем объект ext для работы с функциями и
                                        #    методами модуля pyiArduinoI2Cexpander.
sleep(.5)                               #
ext.pinMode(4, INPUT, ANALOG)           # $ Конфигурируем вывод 4 на работу в качестве аналогового входа.
                                        #
while True:                             #
  pin4 = ext.analogRead(4);             #   Читаем аналоговый уровень с вывода №4 в переменную «pin4».
  print("Pin_4="+str(pin4)              #
        +".\tДля выхода "               #
        "нажмите ctrl+c")               #   Выводим значение в stdout
  sleep(.1)                             # $ Ждём десятую долю секунды, чтоб не захламлять stdout.
```





Для проверки работы скрипта, подключите потенциометр к 4 выводу. Аналоговый уровень считывается из АЦП модуля и может принимать значение от 0 (0В) до 4095 (3,3В). По умолчанию, значения АЦП усредняются, это подавляет скачки показаний, но увеличивает их инерционность, для отключения или изменения усреднения обратитесь к функции analogAveraging()

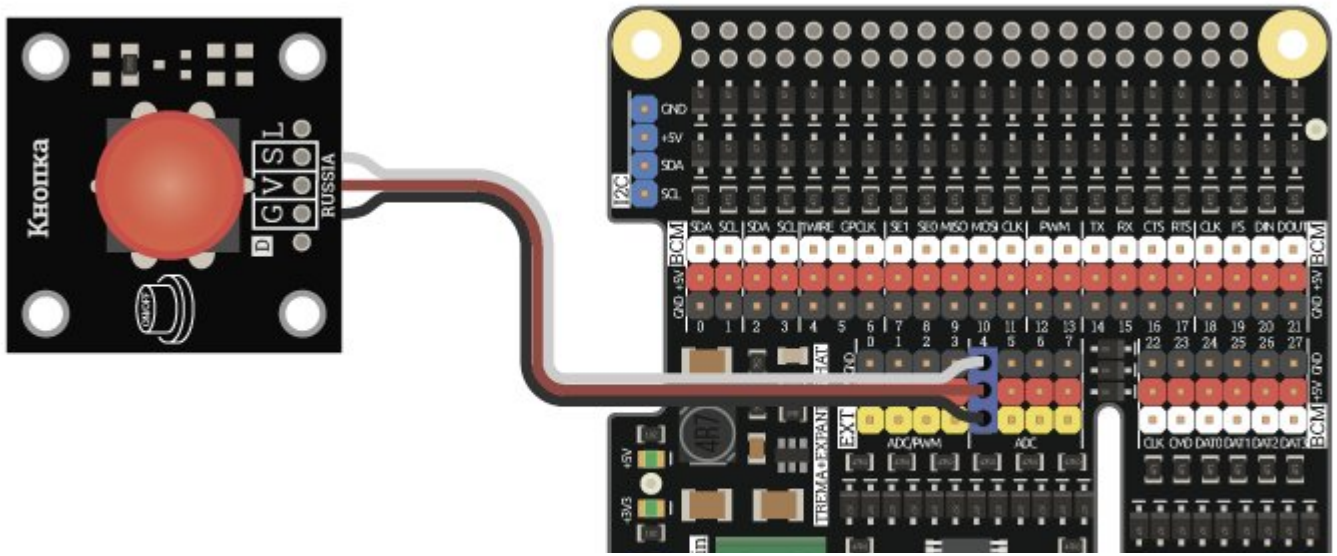
Пример усреднения АЦП:

```
                                    # $ Строки со знаком $ являются необязательными.
from pyiArduinoI2Cexpander import *  #   Подключаем библиотеку для работы с расширителем выводов.
from time import sleep
gpio = pyiArduinoI2Cexpander(0x08)  #   Объявляем объект gpio для работы с функциями модуля pyiArduinoI2Cexpander, указыва
                                    #   Если объявить объект без указания адреса (pyiArduinoI2Cexpander()), то адрес будет
gpio.analogAveraging(255)           #   Указываем максимальный коэффициент усреднения показаний АЦП.
gpio.pinMode(4, INPUT, ANALOG)      # $ Конфигурируем вывод 4 на работу в качестве аналогового входа.
                                    #
while True:                         #
  pin4 = gpio.analogRead(4)         #   Читаем аналоговый уровень с вывода №4 в переменную «pin4».
  print("Pin_4="+str(pin4)          #
        + ".\t Для выхода"          #
        " нажмите ctrl+c")          #
  sleep(.1)                         # $ Ждём десятую долю секунды, чтоб не захламлять stdout.
                                    #
```

Для проверки работы скрипта подключите потенциометр к 4 выводу. Коэффициент усреднения АЦП задаётся функцией analogAveraging() которая в качестве аргумента принимает значение от 0 до 255. По умолчанию 127. Усреднение применяется для всех выводов. Чем выше коэффициент усреднения, тем плавнее будут меняться показания считанных аналоговых уровней. Для отключения усреднения укажите значение 0 или 1.

Пример чтения логического уровня:

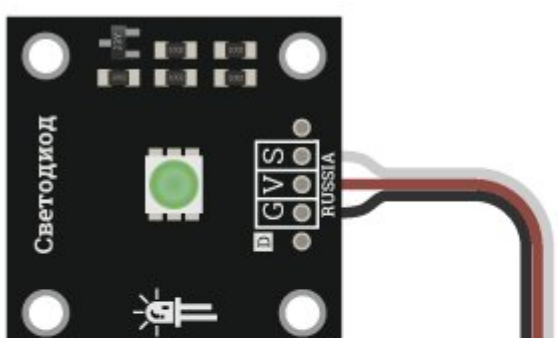
```
                                    # $ Строки со знаком $ являются необязательными.
from pyiArduinoI2Cexpander import *  #   Подключаем модуль для работы с расширителем выводов.
from time import sleep               #   Импортируем функцию ожидания из модуля времени
ext = pyiArduinoI2Cexpander(0x08)    #   Объявляем объект ext для работы с функциями модуля pyiArduinoI2Cexpander, указывая
                                    #   Если объявить объект без указания адреса (pyiArduinoI2Cexpander()), то адрес будет
ext.pinMode(4, INPUT, DIGITAL)       # $ Конфигурируем вывод 4 на работу в качестве цифрового входа.
ext.pinPull(4, PULL_DOWN)            # $ Прижимаем вывод 4 к уровню GND через внутренний резистор.
while True:                          #
  pin4 = ext.digitalRead(4)          #   Читаем логический уровень с вывода №4 в переменную «pin4».
  print("Pin_4=" + str(pin4)         #
        + "\t\tНажмите ctrl+c"       #
        " для выхода")               #
  sleep(.1)                          # $ Ждём десятую долю секунды, чтоб не захламлять stdout.
                                    #
```

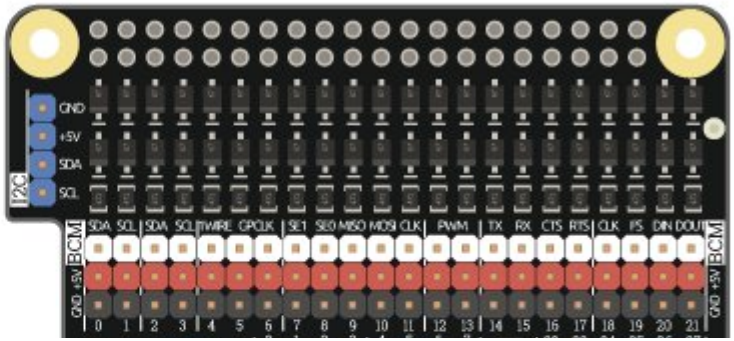


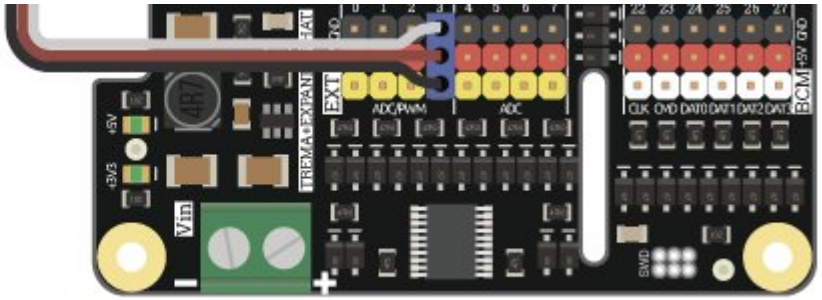
Для проверки работы скрипта подключите кнопку к 4 выводу и Vcc. Если Вы желаете подключить кнопку не к Vcc, а к GND, то уровень 4 вывода нужно не прижать к GND, а подтянуть до уровня Vcc. Для этого укажите функции pinPull() не PULL_DOWN, а PULL_UP.

Пример изменения логического уровня на выводе модуля:

```
                                    # $ Строки со знаком $ являются необязательными.
from pyiArduinoI2Cexpander import *  #   Подключаем библиотеку для работы с расширителем выводов.
from time import sleep               #
ext = pyiArduinoI2Cexpander(0x08)    #   Объявляем объект ext для работы с функциями модуля pyiArduinoI2Cexpander, указывая
                                     #   Если объявить объект без указания адреса (pyiArduinoI2Cexpander()), то адрес будет
ext.pinMode(4, OUTPUT, DIGITAL)      # $ Конфигурируем вывод 7 на работу в качестве цифрового выхода.
print("Моргаем светодиодом."         #
      " Нажмите ctrl+c для выхода")  #
while True:                          #
  ext.digitalWrite(4, HIGH)          #   Устанавливаем высокий логический уровень на выводе 7.
  sleep(.5)                          #   Ждём пол секунды.
  ext.digitalWrite(4, LOW)           #   Устанавливаем низкий логический уровень на выводе 7.
  sleep(.5)                          #   Ждём полсекунды.
```







Для проверки работы скрипта подключите светодиод к 4 выводу.

Пример установки ШИМ на выводе модуля:

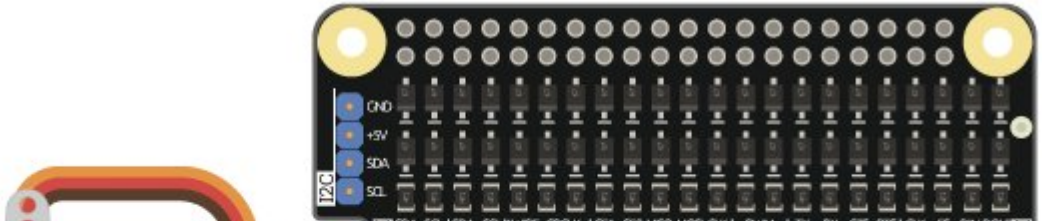
```
                                        # $ Строки со знаком $ являются необязательными.
from pyiArduinoI2Cexpander import *     #   Подключаем библиотеку для работы с расширителем выводов.
from time import sleep                  #   Импортируем функцию ожидания из модуля времени
ext = pyiArduinoI2Cexpander(0x08)       #   Объявляем объект ext для работы с функциями модуля pyiArduinoI2Cexpander, указывая
                                        #   Если объявить объект без указания адреса (pyiArduinoI2Cexpander()), то адрес будет
val = 2                                 #   Определяем начальное аналоговое значение.
flg = 1                                 #   Определяем флаг приращения аналогового значения (0-убывает, 1-растёт).
                                        #
ext.pinMode(3, OUTPUT, ANALOG)          # $ Конфигурируем вывод 3 на работу в качестве аналогового выхода.
print("Меняем аналоговое значение."    #
      " Нажмите ctrl+c для выхода")     #
while True:                             #   Входим в бесконечный цикл
  sleep(.01)                            #   Чем выше задержка, тем плавнее меняется аналоговый уровень.
  if val <= 2 or flag:                  #   Меняем спад аналогового уровня на рост.
    val *= 1.05                         #   Увеличиваем аналоговое значение
    flag = True                         #   Устанавливаем флаг
  if val >= 4095 or not flag:           #   Меняем рост аналогового уровня на спад.
    val /= 1.05                         #   Уменьшаем аналоговое значение
    flag = False                        #   Сбрасываем флаг
                                        #
```

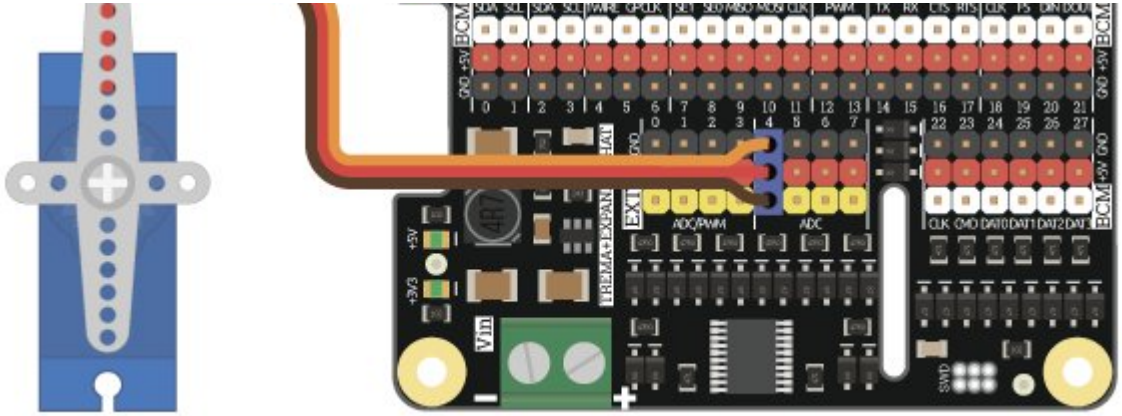
```
  ext.analogWrite(3, val);              #   Устанавливаем на 3 выводе уровень равный значению «val».
```

Для проверки работы скетча подключите светодиод к 3 выводу. Расширитель выводов имитирует аналоговый сигнал используя ШИМ. Сигнал ШИМ поддерживают только выводы 0,1,2,3. Для выводов 4,5,6,7, установка уровня от 0 до 2047 приведёт к появлению логического 0, а установка уровня от 2048 до 4095 к появлению логической 1.

Пример управления сервоприводом:

```
                                    # $ Строки со знаком $ являются необязательными.
from pyiArduinoI2Cexpander import *     #   Подключаем библиотеку для работы с расширителем выводов.
from time import sleep                  #
ext = pyiArduinoI2Cexpander(0x08)       #   Объявляем объект ext для работы с функциями модуля pyiArduinoI2Cexpander, указывая
                                        #
ext.pinMode(3, OUTPUT, SERVO)           # $ Конфигурируем вывод 3 на работу в качестве выхода для сервопривода.
print("Управляем сервоприводом. "      #
      "Нажмите ctrl+c для выхода.")     #
while True:                             #
  ext.servoWrite(3, 0)                  #   Поворачиваем сервопривод в угол 0°.
  sleep(.5)                             #   Ждём полсекунды.
  ext.servoWrite(3, 90)                 #   Поворачиваем сервопривод в угол 90°.
  sleep(.5)                             #   Ждём полсекунды.
  ext.servoWrite(3, 180)                #   Поворачиваем сервопривод в угол 180°.
  sleep(.5)                             #   Ждём полсекунды.
```





Для проверки работы скрипта подключите сервопривод к 3 выводу. По умолчанию модуль работает с сервоприводами у которых угол поворота равен 180°. Для работы с другими сервоприводами задайте их параметры используя функцию servoAttach(). Управление сервоприводами осуществляется с помощью сигнала ШИМ. Обращение к функции servoWrite() приводит к смене частоты ШИМ в значение 50 Гц на всех выводах поддерживающих ШИМ: 0,1,2,3. Значит если сконфигурировать любой другой вывод, поддерживающий ШИМ, на работу в качестве аналогового выхода, его сигнал так же снизится до 50 Гц.

Пример установки параметров сервопривода:

```
                                    # $ Строки со знаком $ являются необязательными.
from pyiArduinoI2Cexpander import *  #   Подключаем библиотеку для работы с расширителем выводов.
from time import sleep               #   Импортируем функцию ожидания из модуля времени
ext = pyiArduinoI2Cexpander(0x08)    #   Объявляем объект ext для работы с функциями модуля pyiArduinoI2Cexpander, указывая
                                    #
ext.pinMode(0, OUTPUT, SERVO)        # $ Конфигурируем вывод 0 на работу в качестве выхода для сервопривода.
ext.pinMode(1, OUTPUT, SERVO)        # $ Конфигурируем вывод 1 на работу в качестве выхода для сервопривода.
ext.pinMode(2, OUTPUT, SERVO)        # $ Конфигурируем вывод 2 на работу в качестве выхода для сервопривода.
ext.pinMode(3, OUTPUT, SERVO)        # $ Конфигурируем вывод 3 на работу в качестве выхода для сервопривода.
ext.servoAttach(0,  500, 2500,       #   Определяем параметры сервопривода,
                0,  180)             #   с углом поворота 180°, подключённого к выводу 0.
ext.servoAttach(1,  900, 2100,       #   Определяем параметры сервопривода,
                0,   90)             #   с углом поворота 90°,  подключённого к выводу 1.
```

```
ext.servoAttach(2, 1320, 1600,          #   Определяем параметры сервопривода,
                -100, +100)             #   постоянного вращения,  подключённого к выводу 2.
ext.servoAttach(3, 1100, 1700,          #   Определяем параметры сервопривода,
                -100, +100)             #   постоянного вращения,  подключённого к выводу 3.
print("Управляем сервоприводами. "      #
      "Нажмите ctrl+c для выхода.")     #   Назначение аргументов функции servoAttach() см. внизу скетча.
                                        #
while True:                             #
  ext.servoWrite(0,   0)                #   Поворачиваем сервопривод в минимальный угол.
  ext.servoWrite(1,   0)                #   Поворачиваем сервопривод в минимальный угол.
  ext.servoWrite(2, -60)                #   Заставляем медленно вращаться сервопривод против часовой стрелки.
  ext.servoWrite(3, -60)                #   Заставляем медленно вращаться сервопривод против часовой стрелки.
  sleep(.5)                             #   Ждём пол секунды.
  ext.servoWrite(0,  90)                #   Поворачиваем сервопривод в среднее положение.
  ext.servoWrite(1,  45)                #   Поворачиваем сервопривод в среднее положение.
  ext.servoWrite(2,   0)                #   Останавливаем сервопривод постоянного вращения.
  ext.servoWrite(3,   0)                #   Останавливаем сервопривод постоянного вращения.
  sleep(.5)                             #   Ждём пол секунды.
  ext.servoWrite(0, 180)                #   Поворачиваем сервопривод в максимальный угол.
  ext.servoWrite(1,  90)                #   Поворачиваем сервопривод в максимальный угол.
  ext.servoWrite(2, +60)                #   Заставляем медленно вращаться сервопривод по часовой стрелке.
  ext.servoWrite(3, +60)                #   Заставляем медленно вращаться сервопривод по часовой стрелке.
  sleep(.5)                             #   Ждём пол секунды.
```

Для проверки работы скетча подключите сервоприводы к выводам. Параметры сервоприводов задаются из их datasheet. Функция servoAttach() принимает до 5 аргументов:

1 - номер вывода к которому подключён сервопривод.

2 - минимальная ширина импульсов ШИМ в микросекундах.

3 - максимальная ширина импульсов ШИМ в микросекундах.

4 - угол или значение соответствующее минимальной ширине импульса.

5 - угол или значение соответствующее максимальной ширине импульса.

ПРИМЕР:

Сервопривод с углом поворота в 45° подключён к 3 выводу модуля. При 0° ширина ШИМ = 600мкс, при 45° ширина ШИМ = 2300мкс.

ext.servoAttach(3, 600, 2300, 0, 45)

ext.servoWrite(3, угол от 0 до 45°)

Описание функций модуля pyiArduinoI2Cexpander:

Подключение модуля:

```
from pyiArduinoI2Cexpander import *
```

Создание объекта:

```
ext = pyiArduinoI2Cexpander(АДРЕС)
```

Функция begin()

- Назначение: инициализация расширителя выводов
- Синтаксис: `ext.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 ext.begin():
    print("устройство найдено и инициализировано")
else:
    print("устройство не найдено, проверьте включена ли шина I2C")
```

Функция changeAddress()

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

```
ext.changeAddress(0x0A)
```

Функция reset()

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

```
ext.reset()
```

Функция getAddress()

- Назначение: узнать текущий адрес устройства на шине I2C

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

```
addr = ext.getAddress()
```

Функция getVersion()

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

```
ver = ext.getVersion()
```

Функция pinMode()

- Назначение: конфигурирование выводов
- Синтаксис: `ext.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
ext.pinMode(pot, INPUT, ANALOG)
ext.pinMode(led, OUTPUT, DIGITAL)
ext.pinMode(servo, OUTPUT, SERVO)
```

Функция pinPull()

- Назначение: подключение к выводу прижимающего или подтягивающего резистора
- Синтаксис: `ext.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
ext.pinPull(button_pin, PULL_UP)
```

Функция pinOutScheme()

- Назначение: выбор схемы выхода
- Синтаксис: `ext.pinOutScheme(pin, mode)`
- Параметры:
  - **pin** - номер вывода (0-7)

    - **mode** - схема (**OUT_PUSH_PULL**, **OUT_OPEN_DRAIN**)
- Возвращаемые значения:нет
- Примечание: нет
- Пример:

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

Функция digitalWrite()

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

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

Функция digitalRead()

- Назначение: чтение логического уровня
- Синтаксис: `ext.digitalRead(pin)`
- Параметры: **pin** - номер вывода (0-7)
- Возвращаемые значения: логический уровень вывода
- Примечание: нет

- Пример:

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

Функция analogWrite()

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

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

Функция analogRead()

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

```
pot_pin = 3
```

```
val = ext.analogRead(pot_pin)
```

Функция analogAveraging()

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

```
ext.analogAveraging(255)
```

Функция levelWrite()

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

```
ext.levelWrite(512)
```

Функция levelRead()

- Назначение: чтение логического уровня с аналогового вывода
- Синтаксис: `ext.levelRead()`
- Параметры: **pin** - номер вывода (0-7)

- Возвращаемые значения: логический уровень (0, 1)
- Примечание: нет
- Пример:

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

Функция levelHyst()

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

```
ext.levelHyst(12)
```

Функция freqPWM()

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

```
ext.freqPWM(440)
```

Функция servoAttach()

- Назначение: конфигурирование вывода для сервопривода
- Синтаксис: `ext.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
ext.pinMode(servo_pin, OUTPUT, SERVO)
ext.servoAttach(servo_pin, 500, 2500, 0, 180)
```

Функция servoWrite()

- Назначение: установка угла поворота сервопривода
- Синтаксис: `ext.servoWrite(pin, angle)`
- Параметры:
  - **pin** - номер вывода (0-3)
  - **angle** - угол поворота, градусы(0-360°)
- Возвращаемые значения: нет
- Примечание: нет
- Пример:

```
```

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

Функция servoWriteMicroseconds()

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

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

Применение:

- Подключение большого количества модулей к Raspberry Pi