Модуль силовых ключей, 4P канала с измерением тока, FLASH-I2C, подключаем к Raspberry

Общие сведения:

Модуль силовых ключей на 4 P-канала с измерением тока, I2C, Flash - является устройством коммутации, которое позволяет подключать и отключать питание от входа «Vin» (от 4 до 24В) к любому из 4 выходов модуля «K1», «K2», «K3», «K4». При этом устройства подключённые к выходам модуля, не должны потреблять более 2А постоянного тока (на каждый канал).

Модуль позволяет не только подавать питание «Vin» на свои выходы, но и устанавливать на них сигнал ШИМ с амплитудой питания «Vin». Частота и коэффициент заполнения задаются программно, значит модуль может не только включать и выключать устройства, но и управлять скоростью вращения моторов, яркостью свечения ламп, светодиодов и т.д. Так же модуль позволяет узнать ток потребляемый устройствами на каждом выходе модуля.

Модуль способен автоматически отключать нагрузку при превышении заданного тока, или ограничивать потребление тока на указанном уровне.

Управление модулем осуществляется по шине I2C. Модуль относится к линейке «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 управлять устройствами с напряжением питания до 24В и потреблением постоянного тока до 2А.

Видео:

Редактируется ...

Спецификация:

- Напряжение питания: 5 В (постоянного тока)
- Потребляемый ток: до 25 мА.
- Коммутируемое напряжение: от 4 до 24 В.
- Коммутируемые токи: до 2А (на каждый выход модуля).
- 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).
- Шаг измерения токов: 10 мА.
- Ограничение тока на заданном уровне: есть.
- Отключение нагрузки при превышении указанного тока: есть.
- Количество выходов: 4 (все выходы поддерживают ШИМ).
- 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 с креплением: 55 x 55 мм.
- Габариты без креплений: 55 x 45 мм.

- Вес: 20 г.

Внимание! Убедитесь что устройства подключённые к модулю не потребляют токи выше указанных в характеристиках модуля. Превышение нагрузкой тока выше коммутируемого (даже кратковременно) приведёт к безвозвратному повреждёнию канала!

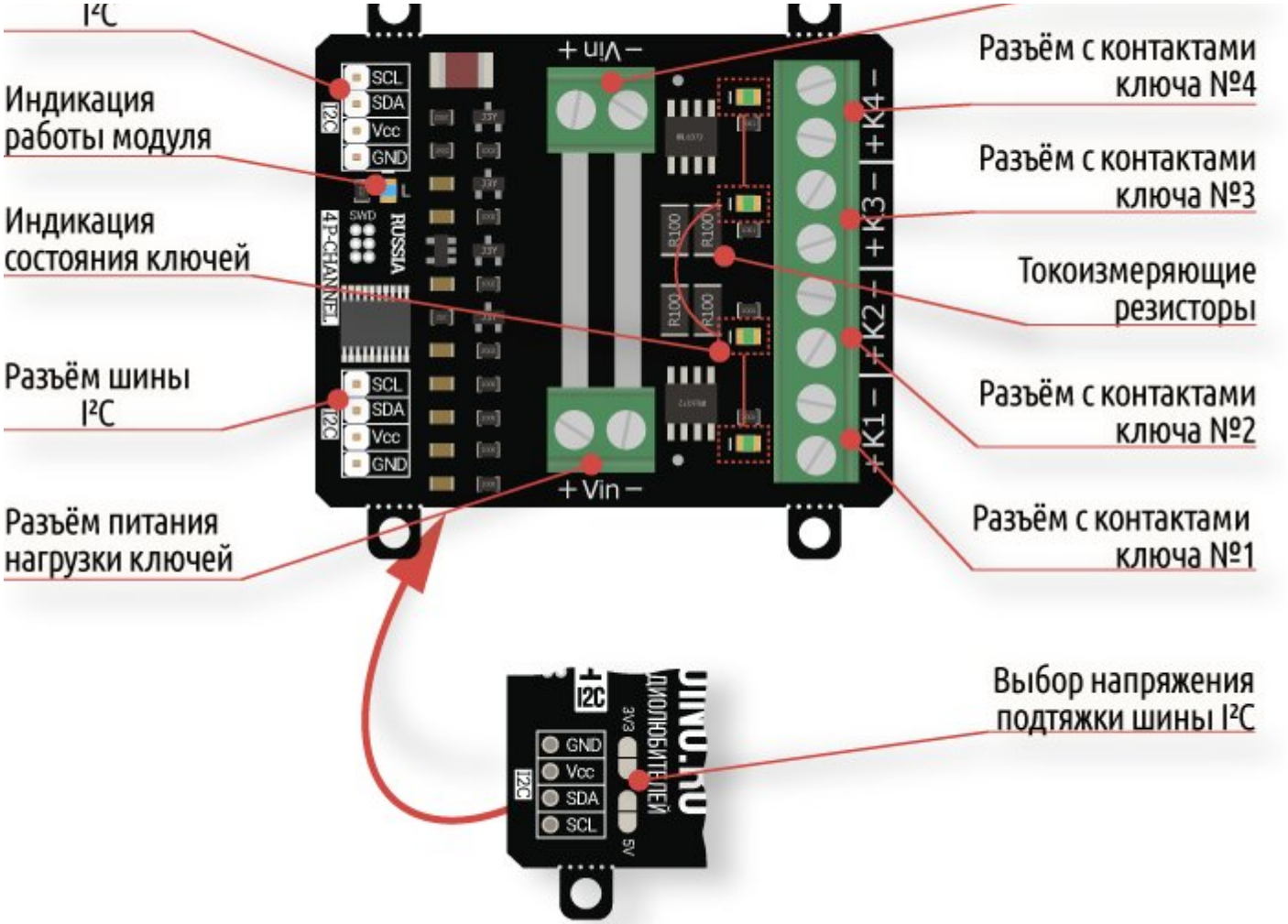
На заметку: Пусковой ток коллекторного двигателя многократно превышает рабочий ток!

Ток потребляемый коллекторным двигателем (двигателем постоянного тока) зависит не только от приложенного к нему напряжения (U), но и от оборотов двигателя (n):
I = (U - (Ce * Ф * n)) / R

I - ток потребляемый коллекторным двигателем.
U - напряжение приложенное к двигателю.
R - сопротивление обмоток ротора (можно измерить омметром).
Ce - конструктивная константа (зависит от двигателя: количества полюсов, витков, зазоров и т.д.).
Ф - сила магнитного поля статора (для постоянных магнитов, так же константа).
n - обороты ротора двигателя.

Из формулы видно, что при запуске двигателя (пока n=0), потребляемый им ток (пусковой ток) может в разы превышать рабочий ток (при n>0), который и указывается на двигателе.

На заметку: При высоких нагрузках возможен нагрев шунтирующих резисторов и отклонение показаний силы тока от реальных



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**.

В левой части платы расположены два разъема для подключения модуля к шине **I2C**. Шина подключается к любому разъему **I2C**, а второй разъем можно использовать для подключения следующего модуля силовых ключей, или других устройств.

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

По центру платы, сверху и снизу, расположены разъемы **Vin**, для подключения питания коммутируемого на выходы модуля. Напряжение питания до 24 В постоянного тока, подаётся на любой разъём **Vin**, а второй разъем можно использовать для подачи питания на следующий модуль силовых ключей. Дорожки печатной платы, соединяющие разъемы **Vin**, не изолированы паяльной маской, что позволяет нанести на них слой припоя, это уменьшит потери при работе с несколькими модулями силовых ключей.

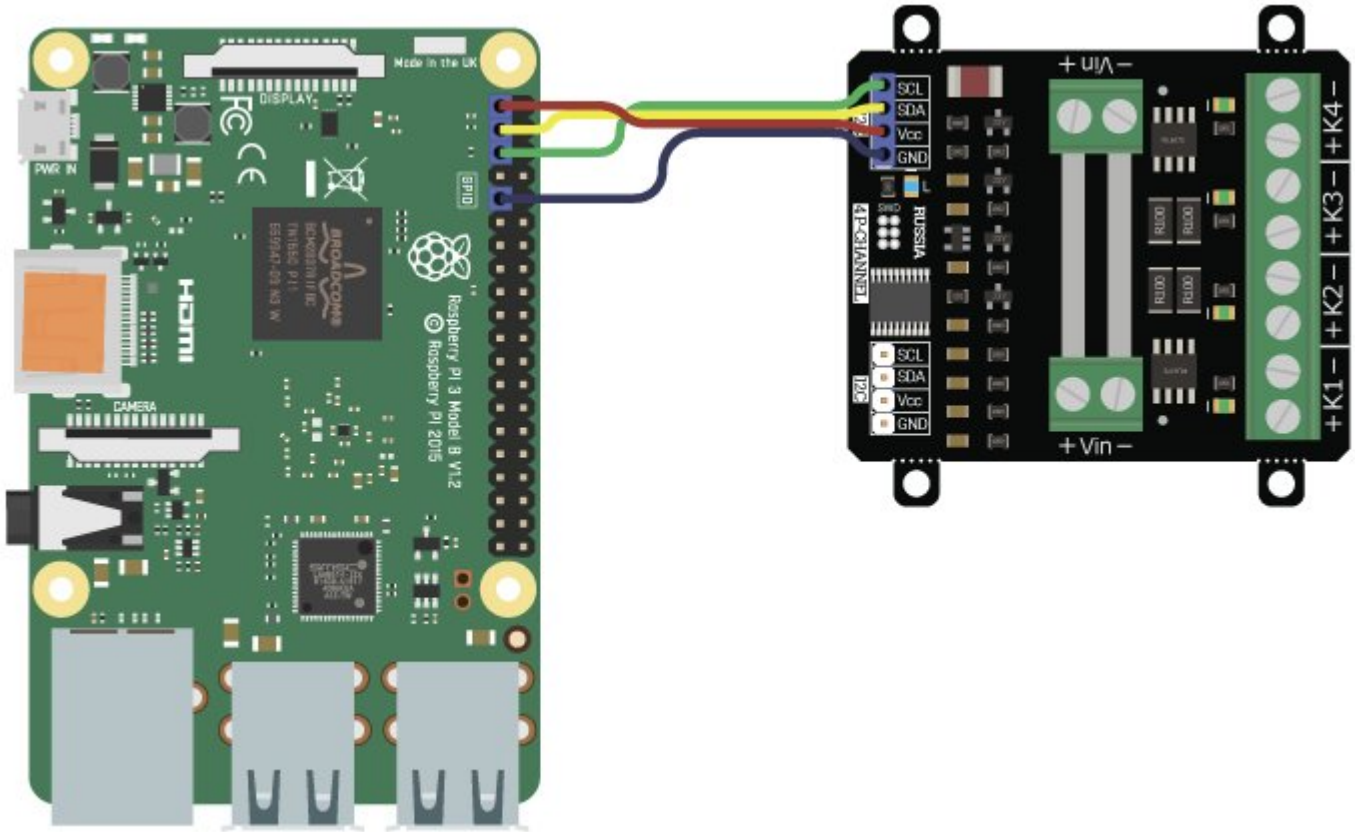
- **Vin** - вход питания (до 24В) коммутируемого на выходы модуля.
- Вывод «**-Vin**» разъема Vin и вывод **GND** разъема I2C электрически соединены на плате модуля.

В правой части платы расположены четыре разъема: **K1**, **K2**, **K3**, **K4**, это выходы на которые коммутируется напряжение со входа Vin. К выходам модуля подключаются устройства которыми Вы желаете управлять. Устройства не должны потреблять более 2А (на каждый выход).

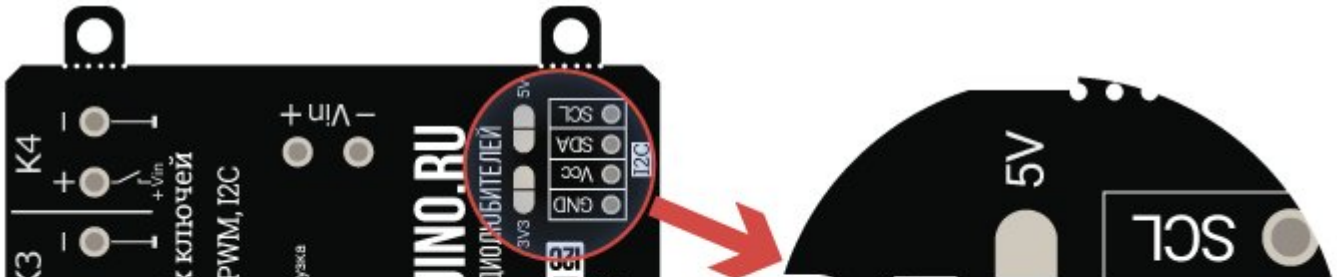
- **K1** - выход №1 для подключения устройства с током потребления до 2А.
- **K2** - выход №2 для подключения устройства с током потребления до 2А.
- **K3** - выход №3 для подключения устройства с током потребления до 2А.
- **K4** - выход №4 для подключения устройства с током потребления до 2А.

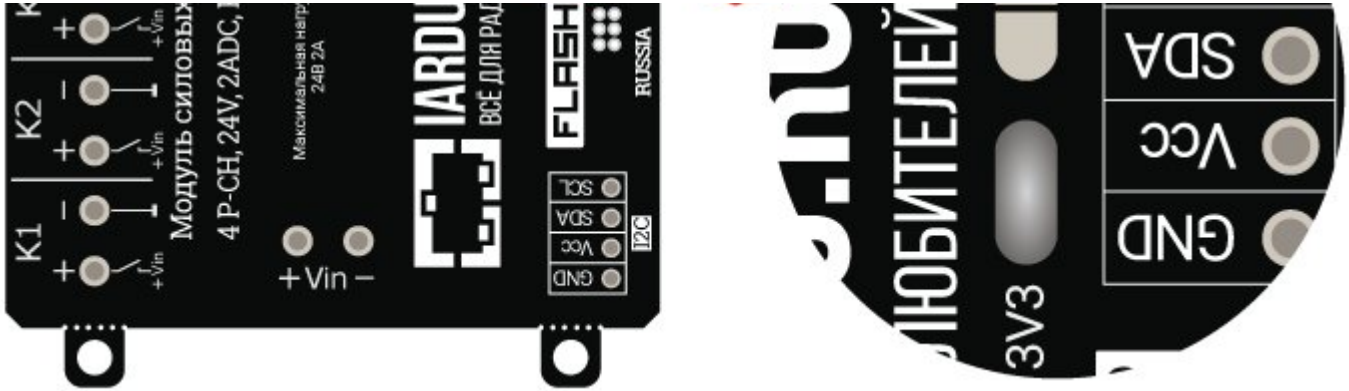
Способ - 1: Используя провода и Piranha UNO

Используя провода «Папа — Мама», подключаем напрямую к контроллеру Raspberry



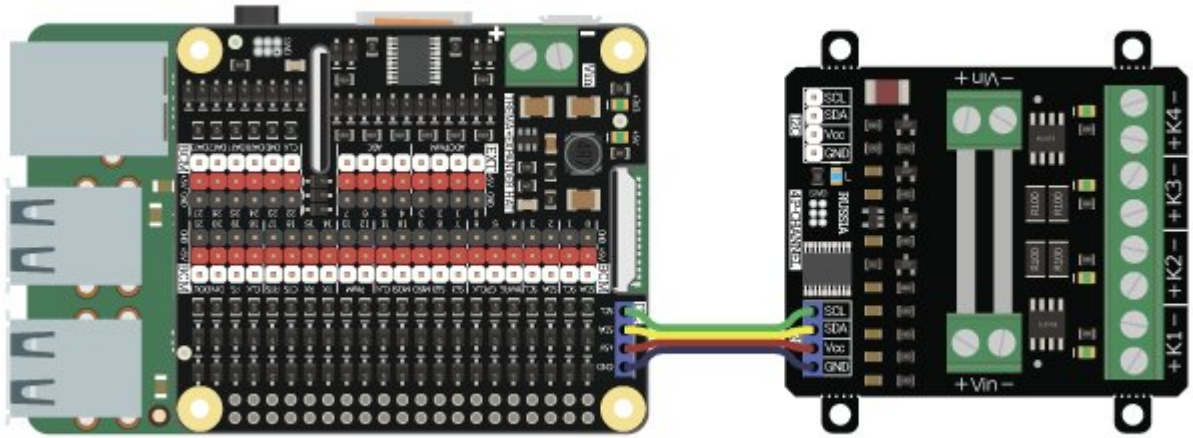
При таком подключении необходимо перепаять джампер выбора подтяжки линий SDA и SCL сзади модуля на 3,3 вольта. Так же, другие устройства на шине I2C должны быть совместимы с этим напряжением и их линии SDA и SCL должны быть либо подтянуты к этому напряжению, либо резисторы на этих линиях должны отсутствовать.





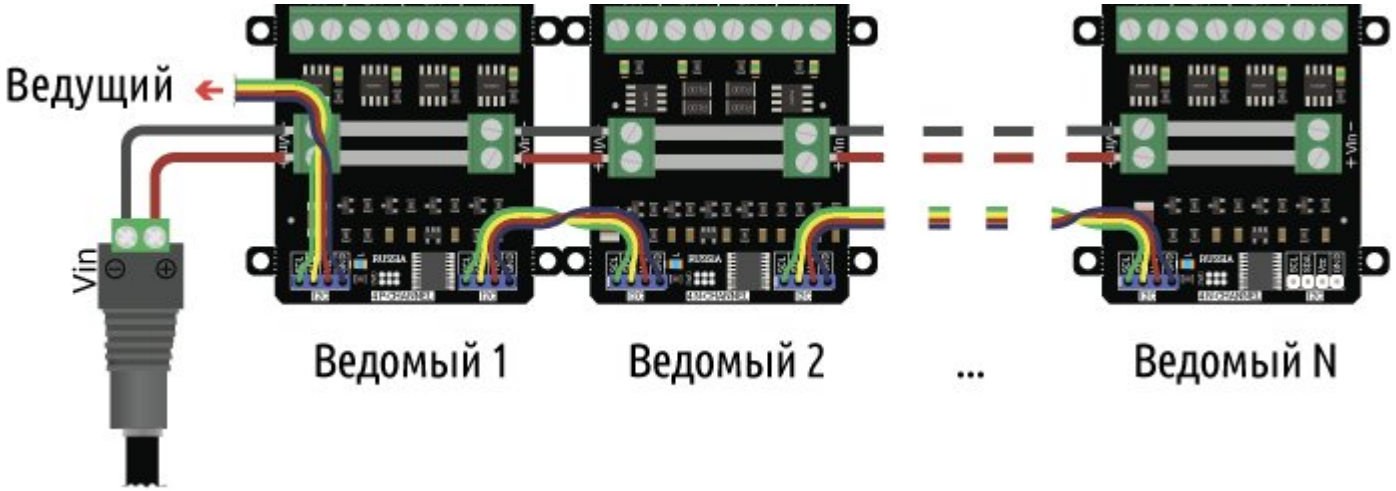
Способ - 2: Используя проводной шлейф и Trema+Expander Hat

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



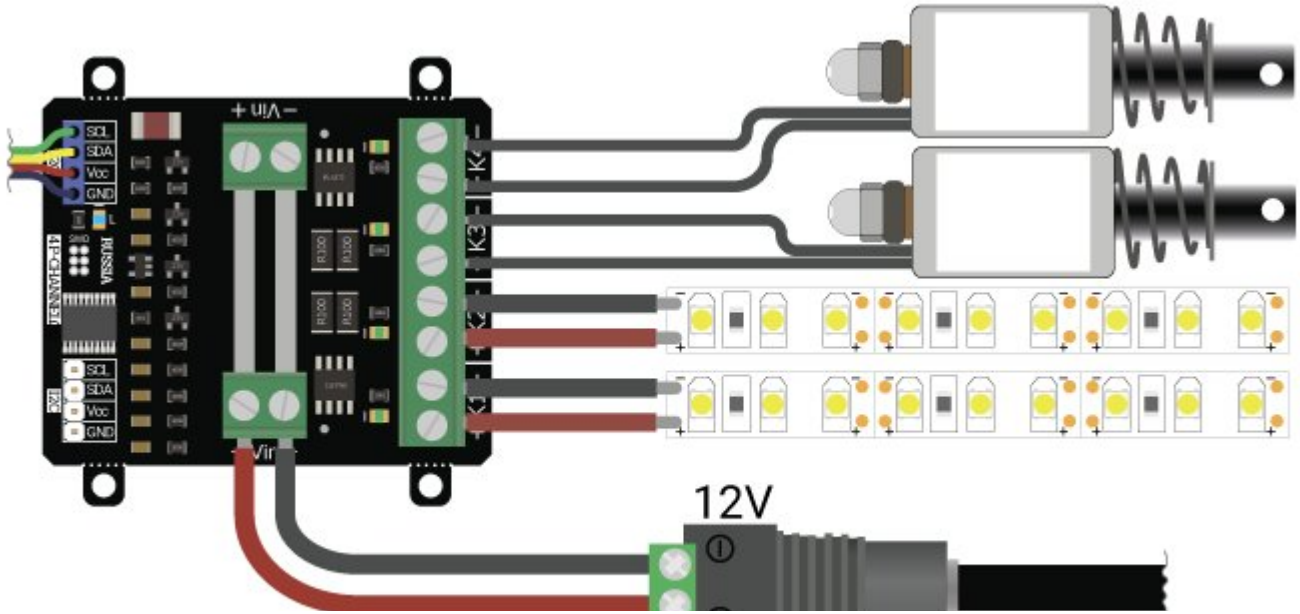
Подключение нескольких модулей:

Благодаря разъёмам питания с двух сторон и двум разъёмам I2C модули можно соединять в одну цепь, предварительно назначив разные адреса:

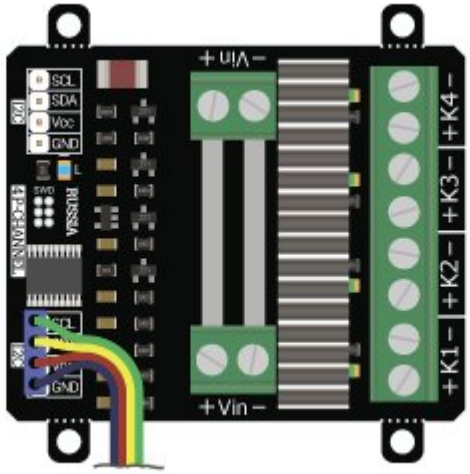


Подключение нагрузки:

Питание потребителей подключается к разъёму Vin модуля, а сами потребители к разъёмам K1, K2, K3 и K4. Все подключённые потребители должны быть с одинаковыми требованиями к напряжению питания. На рисунке ниже каждый потребитель может быть запитан от 12 вольт:



При нагрузках близких к максимальным рекомендуется установка радиаторов и(или) использование активного охлаждения. Например, можно установить один радиатор 43x8x10.



Питание:

Входное напряжение питания логической части модуля, 5В постоянного тока, подаётся на выводы Vcc и GND любого разъёма шины I2C.

Входное напряжение коммутируемое на выходы модуля, до 24 В постоянного тока, подаётся на любой разъём Vin.

Отрицательный вывод разъема Vin и вывод GND разъема I2C электрически соединены на плате модуля.

Подробнее о модуле:

Модуль силовых ключей на 4 P-канала с измерением тока, I2C, Flash построен на базе микроконтроллера STM32F030F4 и снабжен собственным стабилизатором напряжения. У модуля имеются 4 выхода «K1», «K2», «K3», «K4» которые подключены к входу питания «Vin» через полевые P-канальные MOSFET транзисторы. Закрытие транзисторов приводит к разрыву цепи «+Vin» между входом «Vin» и выходом

модуля.

О состоянии транзисторов можно судить по светодиодам расположенным рядом с разъёмами выходов модуля. Если светодиод светится, значит соответствующий транзистор открыт и питание «Vin» поступает на соответствующий выход модуля.

Последовательно с транзисторами, установлены шунтирующие резисторы на которых падает напряжение пропорционально силе тока потребляемого нагрузкой на выходе. Напряжение с резисторов поступает на входы АЦП микроконтроллера, который рассчитывает силу тока проходящего через каждый выход модуля.

Модуль позволяет:

- Установить / отключить питание «Vin» на любом из выходов.
- Установить сигнал ШИМ на любом из выходов (12 бит).
- Задать частоту ШИМ для всех выходов.
- Прочитать силу тока (в мА) на любом выходе.
- Разрешить / запретить автоотключение любого выхода при превышении заданной силы тока.
- Разрешить / запретить ограничение силы тока на любом выходе до заданного значения.

Специально для работы с модулями силовых ключей и реле, нами разработан модуль Python pyiarduinoI2Crelay который позволяет реализовать все функции модуля.

Подробнее про установку модулей Python читайте в нашей инструкции.

Примеры:

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

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

```
sudo pip3 install pyiArduinoI2Crelay
```

Подробнее об установке библиотек можно узнать в [этой статье.]

Все примеры рассчитаны на версию Python > 3.

Поочерёдное включение и выключение каналов модуля:

```
from pyiArduinoI2Crelay import *          # Подключаем библиотеку для работы с реле и силовыми ключами.
from time import sleep                    # Подключаем метод sleep библиотеки time
pwrfet = pyiArduinoI2Crelay(0x09);        # Создаём объект pwrfet для работы с функциями и методами библиотеки Py_iarduino_I2C_R
                                          # Если объявить объект без указания адреса (pwrfet = pyiArduinoI2Crelay()), то адрес 6
pwrfet.digitalWrite(ALL_CHANNEL,LOW);     # Выключаем все каналы модуля.
while True:                               # Входим в бесконечный цикл
#Включаем и выключаем каналы модуля:      #
  pwrfet.digitalWrite(1,HIGH)             # Включаем 1 канал
  pwrfet.digitalWrite(4,LOW)              # и выключаем 4.
  sleep(.5)                               # Ждём   500 мс.
  pwrfet.digitalWrite(2,HIGH)             # Включаем 2 канал
  pwrfet.digitalWrite(1,LOW)              # и выключаем 1.
  sleep(.5)                               # Ждём   500 мс.
  pwrfet.digitalWrite(3,HIGH)             # Включаем 3 канал
  pwrfet.digitalWrite(2,LOW)              # и выключаем 2.
  sleep(.5)                               # Ждём   500 мс.
  pwrfet.digitalWrite(4,HIGH)             # Включаем 4 канал
  pwrfet.digitalWrite(3,LOW)              # и выключаем 3.
  sleep(.5)                               # Ждём   500 мс.
```

Данный пример будет поочерёдно включать и выключать каждый канал модуля.

Чтение логических состояний каналов модуля:

```
from pyiArduinoI2Crelay import *       # Подключаем библиотеку (модуль) для работы с ключом
pwrfet = pyiArduinoI2Crelay()          # Объявляем объект pwrfet
#   Включаем и выключаем каналы модуля:
pwrfet.digitalWrite(1, LOW);           # Отключаем 1 канал.
pwrfet.digitalWrite(2, HIGH);          # Включаем  2 канал.
pwrfet.digitalWrite(3, LOW);           # Отключаем 3 канал.
pwrfet.digitalWrite(4, HIGH);          # Включаем  4 канал.

#  Проверяем состояние каналов модуля в цикле:
for i in range(4):                     # Проходим по всем каналам модуля.
    print("Канал № %s" % (i), end='')  # Выводим номер очередного канала.
    if pwrfet.digitalRead(i + 1):      # Если функция digitalRead() вернула True
        print(" включен\t", end='')    # значит канал включён.
    else:                              # Если функция digitalRead() вернула False
        print(" отключен\t", end='')   # значит канал отключён.
print("--------")                      #
```

Данный пример считывает логическое состояние каждого канала и выводит его в монитор.

Установка и чтение установленного коэффициента заполнения ШИМ:

```
from pyiArduinoI2Crelay import *   # Подключаем модуль для работы с ключём
pwrfet = pyiArduinoI2Crelay()      # Объявляем объект pwrfet
                                   #
#  Устанавливаем ШИМ на каналах модуля:
                                   #
pwrfet.analogWrite(1, 0x0000)      # Устанавливаем 0% ШИМ на 1 канале
pwrfet.analogWrite(2, 0x0555)      # Устанавливаем 33.3% ШИМ на 2 канале
pwrfet.analogWrite(3, 0x0AAA)      # Устанавливаем 66.6% ШИМ на 3 канале
pwrfet.analogWrite(4, 0x0FFF)      # Устанавливаем 100% ШИМ на 4 канале
```

```
                                    #
#  Проверяем состояние каналов модуля в цикле:
                                    #
for i in range(1, 5):               # Проходим по всем каналам модуля.
    print("ШИМ на канале %d" % i,   # Выводим номер очередного канала.
          "имеет значение %#.4x"    # Выводим значение которое вернула
          % pwrfet.analogRead(i))   # функция analogRead().
```

Данный пример выводит коэффициент заполнения ШИМ (от 0 до 4095) установленный на каждом канале модуля.

Установка ШИМ с плавным изменением коэффициента заполнения:

```
from math import sin, cos, radians              # Из модуля математических функций импортируем синус, косинус и радианы
from time import sleep                          # Импортируем функцию ожидания
from pyiArduinoI2Crelay import *                # Подключаем модуль для работы с ключём
pwrfet = pyiArduinoI2Crelay(0x09)               # Объявляем объект pwrfet
                                                # Если объявить объект без указания адреса (pwrfet = pyiArduinoI2Crelay()), то
val = [0, 0, 0, 0]                              # Определяем начальные аналоговые значение в списке.
channels = (1, 2, 3, 4)                         # Определяем номера каналов в кортеже
                                                #
pwrfet.analogWrite(ALL_CHANNEL, LOW)            # Отключаем все каналы.
print("Меняем сигнал ШИМ"                      #
      " на на всех каналах."                    #
      " Нажмите ctrl+c для"                     #
      " остановки")                             #
                                                #
try:                                            #
    while True:                                 #
        for x in range(1, 360):                 # От 1 до 360 градусов
            sleep(0.001)                        # Чем выше задержка, тем плавнее меняется аналоговый уровень.
            val[0] = cos(                       # Берем косинус угла
                        radians(x * x))         # Преобразуем углы в радианы
            val[1] = sin(                       # Сдвиг на 90 градусов относительно val[0]
```

```
                        radians(x * x))         #
            val[2] = -cos(                      # Сдвиг на 180 градусов относительно val[0]
                         radians(x * x))        #
            val[3] = -sin(                      # Сдвиг на 270 градусов относительно val[0]
                         radians(x * x))        #
            for i, j in zip(channels, val):     # Вызываем функцию zip для итерирования двух контейнеров
                pwm = int(1 - j) * 2047         # Преобразуем интервал от -1 до 1 в интервал от 0 до 4094
                pwrfet.analogWrite(i, pwm)      # Допустимые значения ШИМ - от 0 до 4095.

except KeyboardInterrupt:                       # Если сценарий прерван с клавиатуры (ctrl+c)
    pwrfet.analogWrite(ALL_CHANNEL, LOW)        # Выключаем все каналы
    print()
    print("программа остановлена,"
          " все каналы отключены")
```

В данном примере, на всех каналах модуля, устанавливаются сигналы ШИМ, уровни которых плавно нарастают до максимума и спадают до минимума по формуле sin(x^2). Алгоритм работы сценария устроен так, что максимальные и минимальные значения ШИМ смещены от канала к каналу на 90 градусов (если на 1 канале максимум, то на 3 - минимум, если на 2 канале минимум, то на 4 - максимум).

Если подключить к любому каналу лампочку, то она будет плавно включаться и выключаться с нарастающей скоростью, а если подключить мотор, то его скорость будет плавно увеличиваться и уменьшаться.

Чтение силы тока:

```
from pyiArduinoI2Crelay import *        # Подключаем модуль для работы с ключём
from time import sleep                  # Подключаем функцию ожидания из модуля времени
pwrfet = pyiArduinoI2Crelay(0x09)       # Объявляем объект pwrfet
i = 0                                   #
                                        #
pwrfet.digitalWrite(ALL_CHANNEL,LOW)    #  Выключаем все каналы модуля.
                                        #
```

```
while True:                             #
#  Управляем каналом N 3:
    if i==0:                            # Если в переменной хранится значение 0
        pwrfet.digitalWrite(4,HIGH)     # включаем 3 канал
    if i==128:                          # если в переменной хранится значение 128
        pwrfet.digitalWrite(4, LOW)     # отключаем 3 канал
                                        #
#  Выводим силу тока
#  проходещего по 3 каналу:
    j = pwrfet.currentRead(4)           # Считываем силу тока с третьего канала в переменную j.
    print("Сила тока %.3f" % j          #
          +" A.", end = '\r')           # Выводим силу тока в stdout.
#  Приостанавливаем
#  выполнение скетча:
    i+=4                                # Увеличиваем счётчик i на 4
    sleep(.1)                           # и ждем 100 мс, до сброса переменной i пройдет = 6,4 сек.
    if i > 255: i = 0                   # Сбрасываем i при достижении 256
```

Данный пример включает и отключает третий канал модуля, при этом считывает и выводит в монитор силу тока проходящего по указанному каналу. Если к третьему каналу подключить нагрузку, то в мониторе будет отображаться ток потребляемый нагрузкой в то время пока канал включён. А пока канал выключен ток будет равен 0.00 А. Так же возможна калибровка измерения силы тока. Подробнее по этой ссылке.

На заметку: При высоких нагрузках возможен нагрев шунтирующих резисторов и отклонение показаний силы тока от реальных

Защита каналов от перегрузки по заданному току:

```
from pyiArduinoI2Crelay import *            # Подключаем модуль для работы с ключём
from time import sleep                      # Подключаем функцию ожидания из модуля времени
pwrfet = pyiArduinoI2Crelay(0x09)           # Объявляем объект pwrfet
pwrfet.digitalWrite(1,LOW)                  # Отключаем 1 канал модуля.
```

```
pwrfet.digitalWrite(2,LOW)                  # Отключаем 2 канал модуля.
pwrfet.digitalWrite(3,HIGH)                 # Включаем  3 канал модуля.
pwrfet.digitalWrite(4,HIGH)                 # Включаем  4 канал модуля.
#  Устанавливаем защиту по току:
pwrfet.setCurrentProtection(3, 1.0,
                            CURRENT_LIMIT)  # Включаем функцию ограничения тока до 1А на третьем канале.
pwrfet.setCurrentProtection(4, 1.5,
                            CURRENT_DISABLE) # Включаем функцию отключения нагрузки при повышении тока выше 1,5А на четвёртом
                                            # Если для одного канала включить сразу две функции, то будет работать только пос
                                            #
while True:                                 #
#  Проверяем выполняется ли функция ограничения тока, включенная на 3 канале #
    if pwrfet.getCurrentProtection(3):      # Если на третьем канале выполняется функция ограничения тока, то ...
        print("На 3 канале "               # Выводим сообщение о том что ток третьего канала
              "выполняется функция "        # ограничивается модулем при помощи ШИМ.
              "ограничения тока." )         #
                                            #
#  Проверяем не отключена ли нагрузка на 4 канале в связи с превышением тока
    if pwrfet.getCurrentProtection(4):      # Если на четвёртом канале выполнено отключение нагрузки
        print("Нагрузка 4 канала "         # Выводи сообщение о том что нагрузка четвёртого канала отключена.
              "отключена из-за "
              "превышения тока.")
    # Для восстановления работы нагрузки
    # разкомментируйте следующие строки:
    #  pwrfet.resCurrentProtection(4)       # * Перезапускаем защиту 4 канала.
    #  pwrfet.digitalWrite(4,HIGH)          # * Включаем 4 канал модуля (если ток по прежнему превышает
                                            # указанные ранее 1,5А, то нагрузка вновь отключится).
    sleep(1)                                #
```

Данный пример защищает третий и четвёртый каналы модуля от перегрузки по заданному току.

Если ток на третьем канале превысит 1.0 А, то модуль начнет самостоятельно ограничивать ток до 1А при помощи формирования ШИМ.

Если ток на 4 канале превысит 1.5 А, то модуль отключит нагрузку данного канала.

Если канал отключён из-за превышения тока, то для восстановления его работы необходимо, либо отключить защиту функцией `delCurrentProtection()` , либо перезапустить защиту функцией `resCurrentProtection()` .

Внимание: Убедитесь что устройства подключённые к модулю не потребляют токи выше указанных в характеристиках модуля. Защита каналов от перегрузки работает только в пределах коммутируемого тока модуля.

Защита от зависаний Arduino и отключения шины I2C:

```
from pyiArduinoI2Crelay import *        # Подключаем библиотеку для работы с реле и силовыми ключами.
from time import sleep                  # Импортируем функцию ожидания
pwrfet = pyiArduinoI2Crelay(0x09)       # Объявляем объект pwrfet для работы с функциями и методами библиотеки iarduino_I2C_Re
                                        # Если объявить объект без указания адреса (pwrfet = pyiArduinoI2Crelay()), то адрес б
pwrfet.enableWDT(5)                     # Разрешаем работу сторожевого таймера модуля, задав время до перезагрузки 5 сек.
                                        # Отключить работу сторожевого таймера модуля можно функцией disableWDT().
print("Проверка сторожевого таймера"    #
      ", нажмите ctrl+c для выхода")    #
                                        #
while True:                             #
#  Переключаем 1 и 2 каналы модуля:
    pwrfet.digitalWrite(1, HIGH)        # Включаем 1 канал
    pwrfet.digitalWrite(2, LOW)         # выключаем 2 канал
    sleep(.5)                           # ждём 500 мс.
    pwrfet.digitalWrite(2, HIGH)        # Включаем 2 канал
    pwrfet.digitalWrite(1, LOW)         # выключаем 1 канал
    sleep(.5)                           # ждём 500 мс.
#  Сбрасываем (перезапускаем) сторожевой таймер модуля:
    pwrfet.resetWDT()                   #   Теперь таймер модуля начнёт заново отсчитывать 5 секунд до перезагрузки.
```

```
#  Сообщаем, что сработал сторожевой таймер:
    if not pwrfet.getStateWDT():        # Если таймер отключился, значит он досчитал до 0
        print("ERROR")                  # и перезагрузил модуль отключив все его каналы.
                                        # Если модуль не отвечает (отключилась шина I2C), то функция getStateWDT() так же верн
```

При нормальной работе данного примера, в первую половину секунды включён первый канал, а во вторую половину секунды включён второй канал модуля.

Если отключить от модуля вывод SDA или SCL шины I2C, то его каналы перестанут переключаться, но один из каналов останется включённым. Подождав от 4 до 5 секунд, сработает сторожевой таймер модуля и все каналы отключатся. Время ожидания зависит от того, в каком месте выполнения кода был отключён вывод.

Примечание: Время назначенное сторожевому таймеру функцией `enableWDT()` должно быть больше чем время между вызовами функции `resetWDT()` .

Определение модуля на шине I2C и изменение его адреса:

```
from pyiArduinoI2Crelay import *                # Подключаем библиотеку для работы с реле и силовыми ключами.
                                                #
i = 0x09                                        # Назначаемый модулю новый адрес (0x07 < адрес < 0x7F).
                                                #
choices = {                                     # Создаём словарь для сравнения методом get()
    DEF_MODEL_2RM:                              #
        "электромеханическим"                   #
        " реле на 2-канала",                    #
    DEF_MODEL_4RT:                              #
        "твердотельным реле на 4-канала",       #
    DEF_MODEL_4NC:                              #
        "силовым ключом на "                    #
        "4 N-канала с измерением тока",         #
    DEF_MODEL_4PC:                              #
```

```
        "силовым ключом на 4 P-канала"          #
        " с измерением тока",                   #
    DEF_MODEL_4NP:                              #
        "силовым ключом на "                    #
        "4 N-канала до 10A",                    #
    DEF_MODEL_4PP:                              #
        "силовым ключом на"                     #
        " 4 P-канала до 10A"                    #
}                                               #
                                                #
pwrfet = pyiArduinoI2Crelay()                   # Создаём объект pwrfet для работы
print("На шине I2C ", end='')                   # с функциями и методами библиотеки iarduino_I2C_Relay.
if pwrfet.begin():                              # Инициируем работу с модулем реле или силовыми ключами.
    address = pwrfet.getAddress()               # Если при объявлении объекта указать адрес, например, relay(0xBB),
    model = pwrfet.getModel()                   # то пример будет работать с тем модулем, адрес которого был указан.
    model = choices.get(model,                  # Сравниваем модель модуля с константами, если ни одна
                        "неизвестным силовым"   # не совпала - выводим строку по умолчанию
                        " ключом или реле")     #
                                                #
    print("найден модуль с адресом %s,"        # Выводим текущий адрес модуля.
          " который является %s"                #
          % (address, model))                   #
    if pwrfet.changeAddress(i):                 # Меняем адрес модуля на указанный в переменной i.
        print("Адрес модуля изменён на %s"      # Выводим текущий адрес модуля.
              % (pwrfet.getAddress()))          #
    else:                                       # Если функция changeAddress() вернула false,
        print("Адрес модуля "                   # значит не удалось сменить адрес модуля.
              "изменить не удалось!")           #
else:                                           # Если функция begin() вернула false,
    print("нет ни силовых ключей, ни реле!")    # значит не удалось найти модуль реле или силовых ключей.
```

Для работы этого примера необходимо что бы на шине I2C был только один модуль.

Сценарий рассчитан на ввод нового адреса из stdin.

Данный пример определяет тип модуля, его текущий адрес и присваивает модулю новый адрес на шине I2C.

Описание функций библиотеки:

В данном разделе описаны функции модуля pyiArduinoI2Crelay для работы с силовыми ключами без поддержки измерения тока.

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

```
sudo pip3 install pyiArduinoI2Crelay
```

Подключение библиотеки:

- Если адрес модуля известен (в примере используется адрес 0x09):

```
from pyiArduinoI2Crelay import *    #  Подключаем библиотеку для работы с реле
pwrfet = pyiArduinoI2Crelay()       #  Объявляем объект pwrfet
```

- Если адрес модуля неизвестен (адрес будет найден автоматически):

```
from pyiArduinoI2Crelay import *    #  Подключаем библиотеку для работы с реле
pwrfet = pyiArduinoI2Crelay()       #  Объявляем объект pwrfet
```

При создании объекта без указания адреса, на шине должен находиться только один модуль.

Функция begin()

- Назначение: Инициализация работы с модулем.

- Синтаксис: begin();
- Параметры: Нет.
- Возвращаемое значение: результат инициализации (true или false).
- Примечание: Выполняется в конструкторе объекта при его объявлении.
- Пример:

```
if pwrfet.begin():
    print("Модуль найден и инициирован!")
else:
    print("Модуль не найден на шине I2C" )
```

Функция reset()

- Назначение: Перезагрузка модуля.
- Синтаксис: reset();
- Параметры: Нет.
- Возвращаемое значение: bool - результат перезагрузки (true или false).
- Пример:

```
if pwrfet.reset():
    print("Модуль перезагружен")
else:
    print("Модуль не перезагружен")
```

Функция changeAddress()

- Назначение: Смена адреса модуля на шине I2C.
- Синтаксис: changeAddress(АДРЕС);
- Параметры:
  - АДРЕС — новый адрес модуля на шине I2C (целое число от 0x08 до 0x7E)
- Возвращаемое значение: результат смены адреса (true или false).

- Примечание: Текущий адрес модуля можно узнать функцией getAddress().
- Пример:

```
if pwrfet.changeAddress(0x12)
    print("Адрес модуля изменён на 0x12")
else:
    print("Не удалось изменить адрес")
```

Функция getAddress()

- Назначение: Запрос текущего адреса модуля на шине I2C.
- Синтаксис: getAddress();
- Параметры: Нет.
- Возвращаемое значение: АДРЕС — текущий адрес модуля на шине I2C (от 0x08 до 0x7E)
- Примечание: Функция может понадобиться если адрес модуля не указан при создании объекта, а обнаружен библиотекой.
- Пример:

```
print("Адрес модуля на шине I2C", end="")
print(hex(pwrfet.getAddress()))
```

Функция getVersion()

- Назначение: Запрос версии прошивки модуля.
- Синтаксис: getVersion();
- Параметры: Нет
- Возвращаемое значение: ВЕРСИЯ — номер версии прошивки от 0 до 255.
- Пример:

```
print("Версия прошивки модуля")
print(hex(pwrfet.getVersion()))
```

Функция getModel()

- Назначение: Запрос типа модуля.
- Синтаксис: getModel();
- Параметры: Нет.
- Возвращаемое значение: МОДЕЛЬ — идентификатор модуля от 0 до 255.
- Примечание: По идентификатору можно определить тип модуля (см. пример).
- Пример:

```
model = pwrfet.getModel()                                  #  Записываем модель
if model == DEF_MODEL_2RM:                                 #  Сравниваем с константами
    model = "электромеханическим реле на 2-канала"         #  и записываем в ту же переменную
elif model == DEF_MODEL_4RT:                               #
    model = "твердотельным реле на 4-канала"               #
elif model == DEF_MODEL_4NC:                               #
    model = "силовым ключом на 4 N-канала с измерением тока"  #
elif model == DEF_MODEL_4PC:                               #
    model = "силовым ключом на 4 P-канала с измерением тока"  #
elif model == DEF_MODEL_4NP:                               #
    model = "силовым ключом на 4 n-канала до 10a"          #
elif model == DEF_MODEL_4PP:                               #
    model = "силовым ключом на 4 P-канала до 10A"          #
else:                                                      #
    model = "неизвестным силовым ключом или реле"          #
```

Функция digitalWrite()

- Назначение: Установка логического уровня на одном или всех каналах модуля.
- Синтаксис: digitalWrite(КАНАЛ , УРОВЕНЬ);
- Параметры:
  - КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.

    - УРОВЕНЬ - значение HIGH или LOW. Можно указать True или False, 1 или 0.
- Возвращаемое значение: Нет.
- Примечание: Функция включает или отключает канал без использования ШИМ.
- Пример:

```
pwrfet.digitalWrite(ALL_CHANNEL, LOW)  # Отключить все каналы.
pwrfet.digitalWrite(2, HIGH)           # Включить второй канал.
pwrfet.digitalWrite(3, 1)              # Включить третий канал.
```

Функция digitalRead()

- Назначение: Чтение логического уровня на одном канале.
- Синтаксис: digitalRead(КАНАЛ);
- Параметры:
    - КАНАЛ - номер канала силового ключа, значение от 1 до 4.
- Возвращаемое значение: УРОВЕНЬ - значение HIGH или LOW.
- Примечание:
    - Функция возвращает логический уровень установленный функцией digitalWrite().
    - Если на канале установлен не логический уровень, а сигнал ШИМ, то функция вернёт HIGH если коэффициент заполнения выше 50%, иначе, вернёт LOW.
- Пример:

```
print( "На четвёртом канале установлен ")
if pwrkey.digitalRead(4) == HIGH:
    print("высокий ")
else:
    print("низкий")
print( "логический уровень.")
```

Функция analogWrite()

- Назначение: Установка сигнала ШИМ с требуемым коэффициентом заполнения.
- Синтаксис: analogWrite(КАНАЛ , УРОВЕНЬ);
- Параметры:
  - КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.
  - УРОВЕНЬ - коэффициент заполнения ШИМ от 0 до 4095 (12 бит).
- Возвращаемое значение: Нет.
- Примечание:
  - Функция позволяет не просто включить или отключить канал, а включить канал на указанный уровень (мощность), что даёт возможность управлять скоростью вращения моторов, яркостью свечения ламп, светодиодов и т.д
- Пример:

```
pwrfet.analogWrite(1, 0   ) # Отключить первый канал.
pwrfet.analogWrite(1, 2047) # Включить  первый канал на половину мощности.
pwrfet.analogWrite(1, 4095) # Включить  первый канал на полную   мощность.
```

Функция analogRead()

- Назначение: Чтение уровня сигнала ШИМ.
- Синтаксис: analogRead(КАНАЛ);
- Параметры:
  - КАНАЛ - номер канала силового ключа, значение от 1 до 4.
- Возвращаемое значение: УРОВЕНЬ - коэффициент заполнения ШИМ от 0 до 4095.
- Примечание:
  - Функция возвращает уровень ШИМ установленный функцией analogWrite().
  - Если на канале установлен не сигнал ШИМ, а логический уровень, то функция вернёт значение 0 (если установлен LOW) или 4095 (если установлен HIGH).
- Пример:

```
print("На четвёртом канале установлен сигнал ШИМ с уровнем "
      + str(pwrfet.analogRead(4))
```

```
      + " из 4095.")
```

*****

Функция currentRead()

- Назначение: Чтение силы тока проходящего через указанный канал модуля.
- Синтаксис: currentRead(КАНАЛ);
- Параметры:
  - КАНАЛ - номер канала силового ключа, значение от 1 до 4.
- Возвращаемое значение: float - сила тока в Амперах.
- Примечание:
  - Шаг измерений тока составляет 10мА.
  - Для улучшения точности показаний воспользуйтесь функцией currentWrite().
- Пример:

```
print("Сила тока на 3 канале равна "
      + str(pwrkey.currentRead(3))
      + " А.")
```

Функция setCurrentProtection()

- Назначение: Установка защиты канала модуля от превышения тока.
- Синтаксис: setCurrentProtection(КАНАЛ , ТОК , РЕЖИМ);
- Параметры:
  - КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.
  - ТОК - сила тока превышение которой должно вызвать защиту (от 0,1 до 25,5 А).
  - РЕЖИМ - одно из двух значений CURRENT_LIMIT или CURRENT_DISABLE:
    - CURRENT_LIMIT - ограничивать ток. Не дать току подняться выше заданного значения.
    - CURRENT_DISABLE - отключить канал. Нагрузка отключится при превышении тока.

- Возвращаемое значение: Нет.
- Примечание:
  - Превышение тока выше указанного приводит к срабатыванию защиты.
  - Срабатывание защиты в режиме ограничения тока заключается в том, что модуль самостоятельно не даёт подняться току выше указанного значения при помощи ШИМ. Модуль самостоятельно перестаёт ограничивать ток (если ток снизится ниже указанного) и возобновляет ограничение тока (если ток вновь превысит указанное значение). Так продолжается пока защита не будет отключена функцией delCurrentProtection().
  - Срабатывание защиты в режиме отключения канала заключается в том, что модуль самостоятельно отключает нагрузку при превышении указанного тока. После того как защита сработала необходимо либо отключить защиту функцией delCurrentProtection(), либо перезапустить защиту функцией resCurrentProtection(). В противном случае канал останется отключённым и не будет реагировать на функции digitalWrite() и analogWrite().
  - О том что сработала защита (вне зависимости от выбранного режима), можно узнать при помощи функции getCurrentProtection().
  - На одном канале может быть установлен только один режим защиты. Запуск другого режима автоматически отменяет предыдущий.
- Пример:

```
pwrfet.setCurrentProtection(3, 1.0, CURRENT_LIMIT  ) # Включаем функцию ограничения тока до 1А на третьем канале.
pwrfet.setCurrentProtection(4, 1.5, CURRENT_DISABLE) # Включаем функцию отключения нагрузки при повышении тока выше 1,5А на че
```

Внимание! Убедитесь что устройства подключённые к модулю не потребляют токи выше указанных в характеристиках модуля. Защита каналов от перегрузки работает только в пределах коммутируемого тока модуля.

Функция delCurrentProtection()

- Назначение: Отключение защиты канала модуля от превышения тока.
- Синтаксис: delCurrentProtection(КАНАЛ);
- Параметры:
  - КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.

- Возвращаемое значение: Нет.
- Пример:

```
pwrfet.delCurrentProtection(3) # Отключить защиту третьего канала.
```

Функция resCurrentProtection()

- Назначение: Перезапуск защиты канала модуля от превышения тока.
- Синтаксис: resCurrentProtection(КАНАЛ);
- Параметры:
  - uint8_t КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.
- Возвращаемое значение: Нет.
- Примечание:
  - Перезапуск защиты выполняет действия, аналогичные вызову функций delCurrentProtection() и setCurrentProtection() с предыдущими параметрами.
- Пример:

```
if pwrfet.getCurrentProtection(4): # Если на четвёртом канале выполнено отключение нагрузки в связи с превышением тока, то ...
    pwrfet.resCurrentProtection(4) # Перезапускаем защиту 4 канала.
    pwrfet.digitalWrite(4, HIGH)   # Включаем 4 канал модуля.
                                   # Если ток по прежнему превышает значение защиты, то нагрузка вновь отключится.
```

Функция getCurrentProtection()

- Назначение: Проверка выполнения защиты на канале модуля.
- Синтаксис: getCurrentProtection(КАНАЛ);
- Параметры:
  - КАНАЛ - номер канала силового ключа, значение от 1 до 4.
- Возвращаемое значение: результат проверки (True или False).
- Примечание:

    - Функция возвращает true если сработала защита вне зависимости от режима защиты.
    - В режиме ограничения тока, функция вернёт true если её вызвать пока модуль ограничивает ток.
    - В режиме отключения канала, функция вернёт true если её вызвать в любой момент после срабатывания защиты и до обращения к функциям отключения или перезапуска защиты.
- Пример:

```
if pwrkey.getCurrentProtection(1):
    print("Сработала защита на 1 канале.")
```

Дополнительные функции библиотеки:

Данные функции не являются основными, но могут быть полезны.

Функция freqPWM()

- Назначение: Установка частоты ШИМ для всех каналов модуля.
- Синтаксис: freqPWM(ЧАСТОТА);
- Параметры:
    - uint16_t ЧАСТОТА - значение от 1 до 12000, определяет новую частоту в Гц.
- Возвращаемое значение: Нет.
- Примечание:
    - Частота ШИМ по умолчанию 490 Гц.
    - Функция analogWrite() позволяет задавать уровень ШИМ (коэффициент заполнения) не влияя на частоту. Чем выше уровень ШИМ, тем длиннее импульсы и короче спады. А функция freqPWM() позволяет изменить частоту ШИМ (период следования импульсов) не влияя на уровень ШИМ.
- Пример:

```
pwrfet.analogWrite(1, 2047) # Включить первый канал на половину мощности.
pwrfet.freqPWM(1000)        # Установить частоту ШИМ в 1000 Гц.
# ПРИМЕЧАНИЕ:                 На первом канале по прежнему половина мощности.
```

Функция enableWDT()

- Назначение: Запуск (разрешение работы) сторожевого таймера модуля.
- Синтаксис: enableWDT(ВРЕМЯ);
- Параметры:
  - ВРЕМЯ - количество секунд от 1 до 254.
- Возвращаемое значение: результат запуска сторожевого таймера (True или False).
- Примечание:
  - После выполнения функции, сторожевой таймер начнёт отсчитывать время, от указанного до 0. Обнуление сторожевого таймера приведёт к перезагрузке модуля и, как следствие, отключению всех его каналов.
  - Если в процессе выполнения скетча, постоянно обращаться к функции enableWDT() (или resetWDT(), см. ниже), то таймер не сможет досчитать до 0, пока корректно работает скетч и шина I2C.
- Пример:

```
pwrfet.enableWDT(10) # Через 10 секунд модуль перезагрузится.
```

Функция disableWDT()

- Назначение: Отключение (запрет работы) сторожевого таймера модуля.
- Синтаксис: disableWDT();
- Параметры: Нет.
- Возвращаемое значение: bool - результат отключения сторожевого таймера (True или False).
- Примечание:
  - Обращение к функции отключает сторожевой таймер без перезагрузки модуля.
- Пример:

```
pwrfet.disableWDT() # Отключение сторожевого таймера.
```

Функция resetWDT()

- Назначение: Сброс (перезагрузка) сторожевого таймера.
- Синтаксис: resetWDT();
- Параметры: Нет
- Возвращаемое значение: результат перезагрузки сторожевого таймера (True или False).
- Примечание:
  - Функция сбрасывает время сторожевого таймера в значение которое было определено ранее функцией enableWDT(), значит таймер начнёт отсчёт времени заново.
  - К функции resetWDT() можно обращаться только если сторожевой таймер уже запущен, в противном случае функция resetWDT() вернёт False.
- Пример:

```
pwrfet.enableWDT(10) # Через 10 секунд модуль перезагрузится;
sleep(9);            # Ждём 9 секунд.
pwrfet.resetWDT();   # Модуль перезагрузится не через 1 секунду, а через 10.
```

Функция getStateWDT()

- Назначение: Чтение состояния сторожевого таймера.
- Синтаксис: getStateWDT();
- Параметры: Нет.
- Возвращаемое значение: выполняется отсчёт времени (True или False).
- Примечание:
  - Если таймер запущен функцией enableWDT() и не отключался функцией disableWDT(), а функция getStateWDT() вернула False, значит таймер досчитал до 0, и перезагрузил модуль.
  - Если модуль не отвечает, например, отключилась шина I2C, то функция getStateWDT() так же вернёт False.
- Пример:

```
if pwrfet.getStateWDT():
    print("таймер выполняет отсчёт времени.")
else:
```

```
    print("сторожевой таймер отключен.")
```