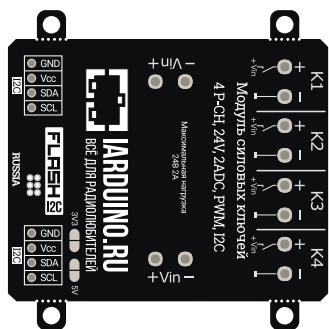


Модуль силовых ключей, 4P канала с измерением тока, FLASH-I2C



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 °С.
- 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 нагрузки тока выше коммутируемого (даже кратковременно) приведёт к безвозвратному повреждению 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):

$$I = (U - (C_e * \Phi * n)) / R$$

I - ток потребляемый коллекторным двигателем.

U - напряжение приложенное к двигателю.

R - сопротивление обмоток ротора (можно измерить омметром).

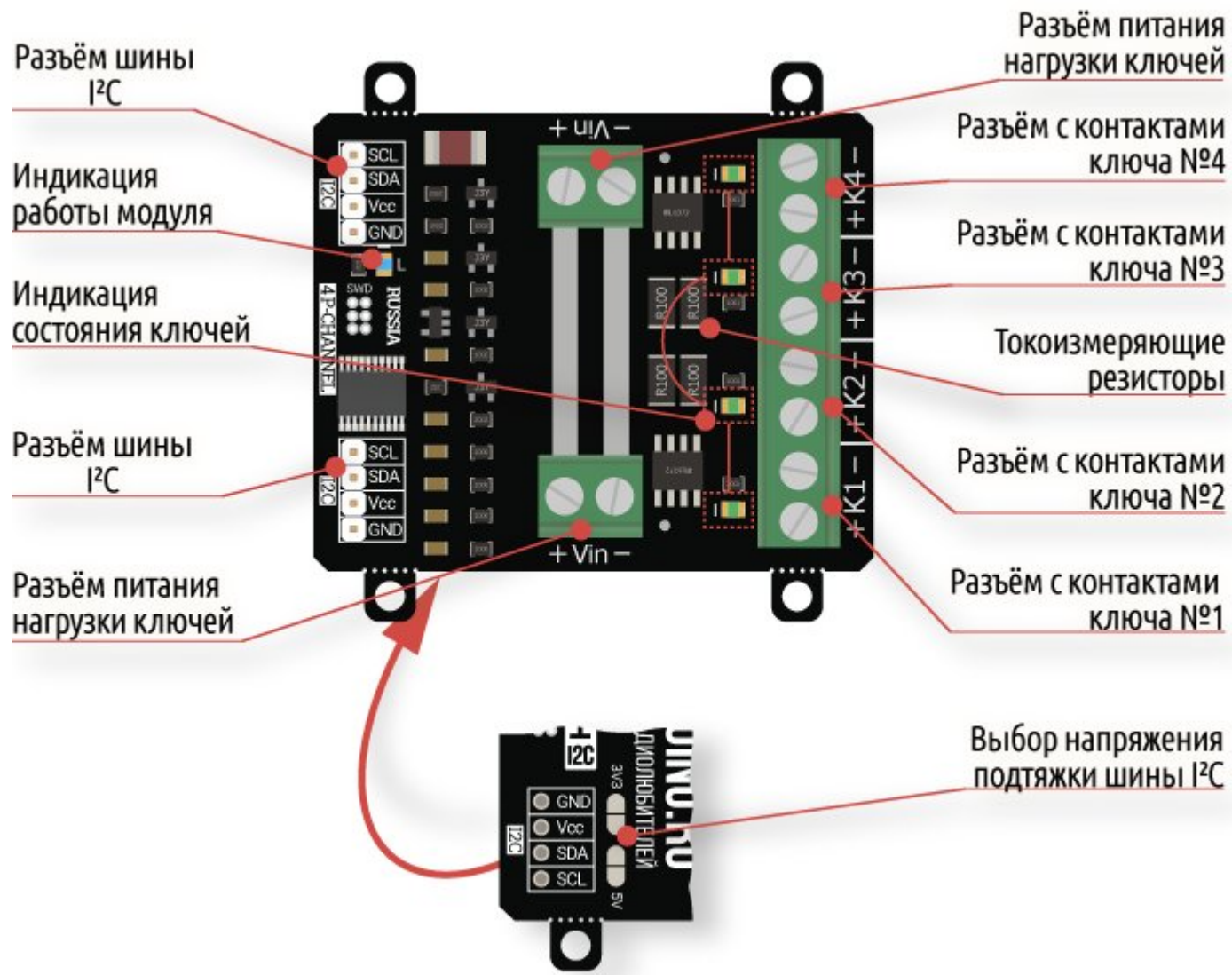
C_e - конструктивная константа (зависит от двигателя: количества полюсов, витков, зазоров и т.д.).

Φ - 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:

Перед подключением модуля ознакомьтесь с разделом "Смена адреса модуля на шине I2C" в данной 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.

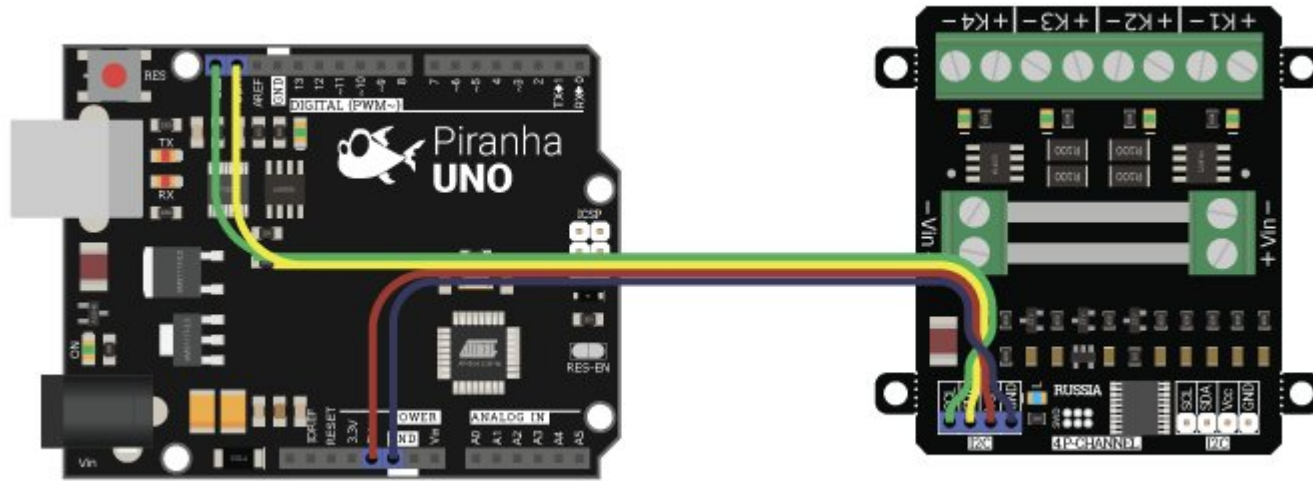
- **Vin** - вход питания (до 24В) коммутируемого на выходы модуля.
- Вывод «**-Vin**» разъема Vin и вывод **GND** разъема I2C электрически соединены на плате модуля.

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.

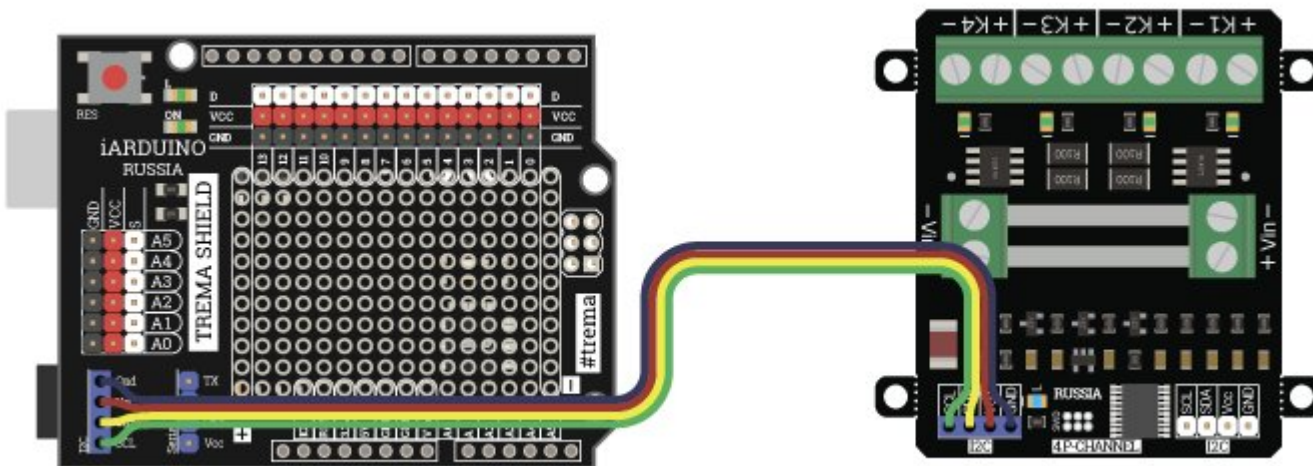
Способ - 1: Используя провода и Piranha UNO

Используя провода «Папа – Мама», подключаем напрямую к контроллеру [Piranha UNO](#)



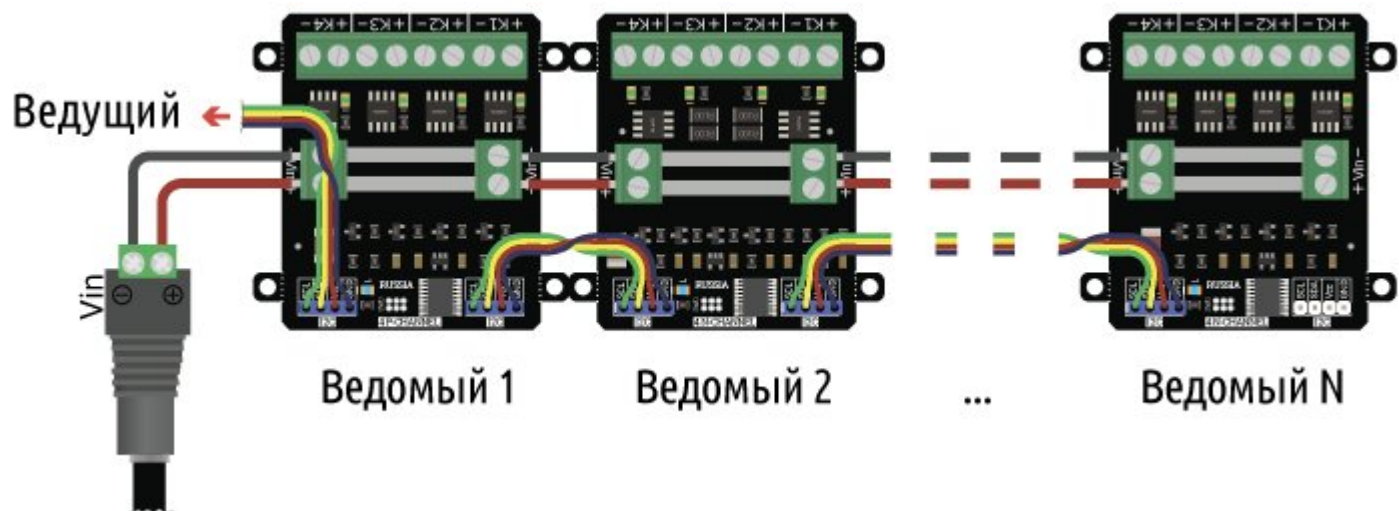
Способ - 2: Используя проводной шлейф и Trema Shield

Используя 4-х проводной шлейф подключаем к [Trema Shield](#)



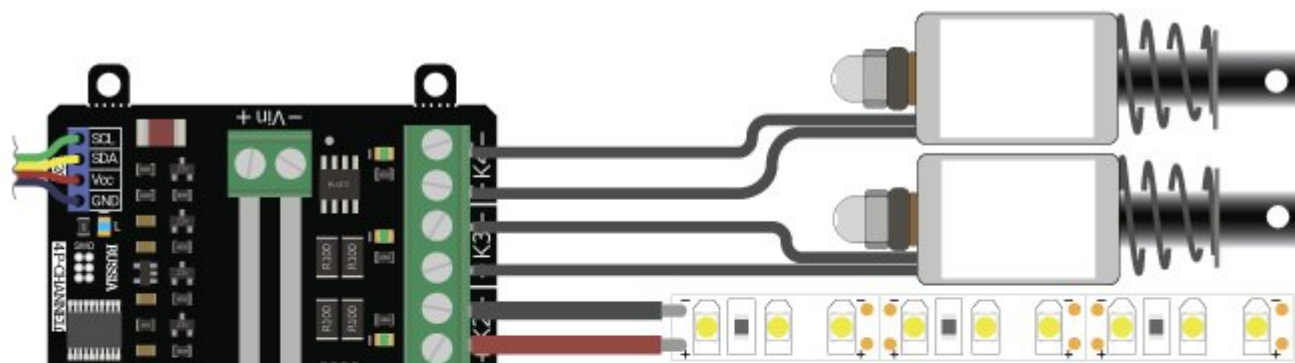
Подключение нескольких модулей:

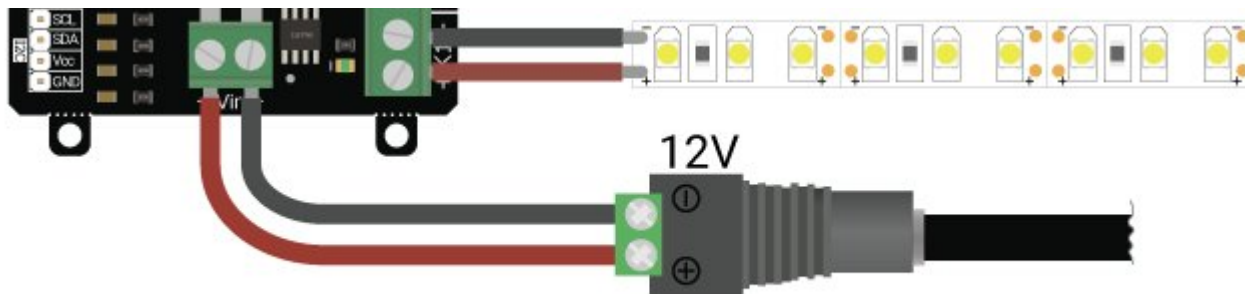
Благодаря разъёмам питания с двух сторон и двум разъёмам I2C модули можно соединять в одну цепь, предварительно назначив разные адреса:



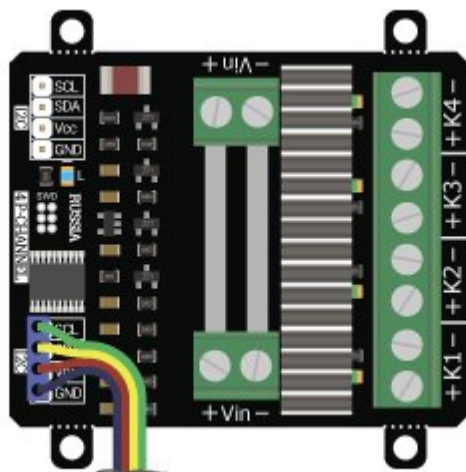
Подключение нагрузки:

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 Р-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 Р-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 разработана [библиотека iarduino_I2C_Relay](#) которая позволяет реализовать все функции модуля.

Подробнее про установку библиотеки читайте в нашей [инструкции](#).

Смена адреса модуля на шине I2C:

По умолчанию все модули FLASH-I2C имеют установленный адрес 0x09. Если вы планируете подключать более 1 модуля на шину I2C,

необходимо изменить адреса модулей таким образом, чтобы каждый из них был уникальным. Более подробно о том, как изменить адрес, а также о многом другом, что касается работы FLASH-I2C модулей, вы можете прочесть в [этой статье](#).

В первой строке скетча необходимо записать в переменную **newAddress** адрес, который будет присвоен модулю. После этого подключите модуль к контроллеру и загрузите скетч. **Адрес может быть от 0x07 до 0x7F.**

```
uint8_t newAddress = 0x09; // Назначаемый модулю адрес (0x07 < адрес < 0x7F).
                               //
#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с модулем
iarduino_I2C_Relay relay; // Объявляем объект для работы с функциями и методами би
                               // Если при объявлении объекта указать адрес, например,
void setup(){ //
    Serial.begin(9600); //
    if( relay.begin() ){ // Иницируем работу с модулем.
        Serial.print("На шине I2C найден модуль с адресом 0x"); //
        Serial.print( relay.getAddress(), HEX ); // Выводим текущий адрес модуля.
        Serial.print(" который является силовым ключом\r\n"); //
        if( relay.changeAddress(newAddress) ){ // Меняем адрес модуля на newAddress.
            Serial.print("Адрес модуля изменён на 0x"); //
            Serial.println( relay.getAddress(), HEX ); // Выводим текущий адрес модуля.
        }else{ //
            Serial.println("Адрес модуля изменить не удалось!"); //
        } //
    }else{ //
        Serial.println("модуль не найден!"); //
    } //
} //
void loop(){ //
```

Примеры:

Поочерёдное включение и выключение каналов модуля:

```
#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми ключами.
iarduino_I2C_Relay pwrkey(0x09); // Объявляем объект pwrkey для работы с функциями и методами биб
// Если объявить объект без указания адреса (iarduino_I2C_Relay
//
//
// Иницилируем работу с модулем.
// Выключаем все каналы модуля.
//
//
//
void setup(){
    pwrkey.begin();
    pwrkey.digitalWrite(ALL_CHANNEL, LOW);
}

void loop(){
    pwrkey.digitalWrite(1,HIGH); pwrkey.digitalWrite(4,LOW); // Включаем 1 канал и выключаем 4.
    delay(500); // Ждём 500 мс.
    pwrkey.digitalWrite(2,HIGH); pwrkey.digitalWrite(1,LOW); // Включаем 2 канал и выключаем 1.
    delay(500); // Ждём 500 мс.
    pwrkey.digitalWrite(3,HIGH); pwrkey.digitalWrite(2,LOW); // Включаем 3 канал и выключаем 2.
    delay(500); // Ждём 500 мс.
    pwrkey.digitalWrite(4,HIGH); pwrkey.digitalWrite(3,LOW); // Включаем 4 канал и выключаем 3.
    delay(500); // Ждём 500 мс.
}
```

Данный пример будет поочерёдно включать и выключать каждый канал модуля.

Чтение логических состояний каналов модуля:

```
#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми ключами.
iarduino_I2C_Relay pwrkey(0x09); // Объявляем объект pwrkey для работы с функциями и методами биб
```

```

// Если объявить объект без указания адреса (iarduino_I2C_Relay
//
void setup(){
  Serial.begin(9600); // Инициуем передачу данных в монитор последовательного порта
  pwrkey.begin(); // Инициуем работу с модулем.
// Включаем и выключаем каналы модуля: //
  pwrkey.digitalWrite(1, LOW); // Отключаем 1 канал.
  pwrkey.digitalWrite(2, HIGH); // Включаем 2 канал.
  pwrkey.digitalWrite(3, LOW); // Отключаем 3 канал.
  pwrkey.digitalWrite(4, HIGH); // Включаем 4 канал.
// Проверяем состояние каналов модуля в цикле: //
  for(int i=1; i<=4; i++){ Serial.print ("Канал N "); // Проходим по всем каналам модуля.
                          Serial.print (i); // Выводим номер очередного канала.
    if(pwrkey.digitalRead(i)){Serial.println(" включен ");} // Если функция digitalRead() вернула HIGH значит канал включён.
    else {Serial.println(" отключен");} // Если функция digitalRead() вернула LOW значит канал отключён.
  } Serial.println("-----"); //
} //
//
void loop(){} //
//
// ПРИМЕЧАНИЕ: //
// состояние всех каналов можно получить одним байтом: //
// uint8_t j = pwrkey.digitalRead(ALL_CHANNEL); // 4 младших бита переменной «j» соответствуют состояниям 4 кана

```

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

Установка и чтение установленного коэффициента заполнения ШИМ:

```

#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми ключами.
iarduino_I2C_Relay pwrkey(0x09); // Объявляем объект pwrkey для работы с функциями и методами биб
// Если объявить объект без указания адреса (iarduino_I2C_Relay
//
void setup(){ //

```

```

Serial.begin(9600); //
pwrkey.begin(); // Иницируем работу с модулем.
// Устанавливаем ШИМ на каналах модуля: //
pwrkey.analogWrite(1, 0x0000); // Устанавливаем ШИМ на 1 канале в значение 0x0000 = 0%.
pwrkey.analogWrite(2, 0x0555); // Устанавливаем ШИМ на 2 канале в значение 0x0555 = 33.3%.
pwrkey.analogWrite(3, 0x0AAA); // Устанавливаем ШИМ на 3 канале в значение 0x0AAA = 66.6%.
pwrkey.analogWrite(4, 0x0FFF); // Устанавливаем ШИМ на 4 канале в значение 0x0FFF = 100%.
// Проверяем состояние каналов модуля в цикле: //
for(int i=1; i<=4; i++){ // Проходим по всем каналам модуля.
    Serial.print ("ШИМ на канале "); //
    Serial.print (i); // Выводим номер очередного канала.
    Serial.print (" имеет значение 0x"); //
    Serial.println(pwrkey.analogRead(i), HEX); // Выводим значение которое вернула функция analogRead().
} //
} //
void loop(){ //

```

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

Установка ШИМ с плавным изменением коэффициента заполнения:

```

#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми ключами.
iarduino_I2C_Relay pwrkey(0x09); // Объявляем объект pwrkey для работы с функциями и методами би
// Если объявить объект без указания адреса (iarduino_I2C_Relay
// Определяем начальное аналоговое значение.
double val = 0; // Определяем флаг приращения аналогового значения (0-убывает,
bool flg = 1; // Определяем математическую функцию.
int f(int i){return abs((flg?-1:1)*2047+(flg?1:-1)*i);} //
//
void setup(){ //
    pwrkey.begin(); // Иницируем работу с модулем.
    pwrkey.analogWrite(ALL_CHANNEL, LOW); // Отключаем все каналы.

```

```

} //
//
void loop(){ //
  delay(10); // Чем выше задержка, тем плавнее меняется аналоговый уровень.
  if( flg ){ val+=50; }else{ val-=50; } // Увеличиваем или уменьшаем аналоговое значение.
  if( val<=0 ){ val = 0; flg=1;} // Меняем спад аналогового уровня на рост.
  if( val>=4095 ){ val = 4095; flg=0;} // Меняем рост аналогового уровня на спад.
  // Выводим ШИМ на все каналы модуля //
  pwrkey.analogWrite(1, val ); // Устанавливаем на 1 канале модуля ШИМ с уровнем «val».
  pwrkey.analogWrite(2,f((int)val)*(flg?-1)+(flg?0:4095)); // Устанавливаем на 2 канале модуля ШИМ с уровнем «val», отстает
  pwrkey.analogWrite(3, 4095 - val ); // Устанавливаем на 3 канале модуля ШИМ с уровнем «val», отстает
  pwrkey.analogWrite(4,f((int)val)*(flg?-1:1)+(flg?4095:0)); // Устанавливаем на 4 канале модуля ШИМ с уровнем «val», опережает
} // Допустимые значения ШИМ - от 0 до 4095.

```

В данном примере, на всех каналах модуля, устанавливаются сигналы ШИМ, уровни которых плавно нарастают до максимума и спадают до минимума. Алгоритм работы скетча устроен так, что максимальные и минимальные значения ШИМ следуют друг за другом от канала к каналу (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, а если подключить мотор, то его скорость будет плавно увеличиваться и уменьшаться.

Чтение силы тока:

```

#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми ключами.
iarduino_I2C_Relay pwrkey(0x09); // Объявляем объект pwrkey для работы с функциями и методами библиотек
// Если объявить объект без указания адреса (iarduino_I2C_Relay pwrkey)
uint8_t i = 0; //
//
void setup(){ //
  Serial.begin(9600); //
}

```

```

pwrkey.begin(); // Иницилируем работу с модулем.
pwrkey.digitalWrite(ALL_CHANNEL,LOW); // Выключаем все каналы модуля.
} //
//
void loop(){ //
// Управляем каналом N 3: //
if( i==0 ){ pwrkey.digitalWrite(3,HIGH); } // Включаем 3 канал если в переменной i хранится значение 0.
if( i==128 ){ pwrkey.digitalWrite(3, LOW); } // Отключаем 3 канал если в переменной i хранится значение 128.
// Выводим силу тока проходящего по 3 каналу: //
float j = pwrkey.currentRead(3); // Считываем силу тока с третьего канала в переменную j.
Serial.println( (String) "Сила тока "+j+" A." ); // Выводим силу тока в монитор последовательного порта.
// Приостанавливаем выполнение скетча: //
i+=4; delay(100); // Увеличиваем счётчик i на 4 и ждем 100 мс, значит до сброса пе
} //

```

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:

```

#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми ключами.
iarduino_I2C_Relay pwrkey(0x09); // Объявляем объект pwrkey для работы с функциями и методами биб
// Если объявить объект без указания адреса (iarduino_I2C_Relay
void setup(){ //
Serial.begin(9600); //

```

```

pwrkey.begin(); // Иницилируем работу с модулем.
pwrkey.digitalWrite(3,HIGH); // Включаем 3 канал модуля.
pwrkey.digitalWrite(4,HIGH); // Включаем 4 канал модуля.
// Устанавливаем защиту по току: //
pwrkey.setCurrentProtection(3, 1.0, CURRENT_LIMIT ); // Включаем функцию ограничения тока до 1А на третьем канале.
pwrkey.setCurrentProtection(4, 1.5, CURRENT_DISABLE); // Включаем функцию отключения нагрузки при повышении тока выше
} // Если для одного канала включить сразу две функции, то будет р
//
void loop(){ //
// Проверяем выполняется ли функция ограничения тока: //
if( pwrkey.getCurrentProtection(3) ){ // Если на третьем канале выполняется функция ограничения тока,
Serial.println( "Ограничивается ток на 3 канале." ); // Выводим сообщение о том что ток третьего канала ограничивается
} //
// Проверяем не отключена ли нагрузка на 4 канале: //
if( pwrkey.getCurrentProtection(4) ){ // Если на четвёртом канале выполнено отключение нагрузки в связ
Serial.println("Нагрузка 4 канала отключена." ); // Выводи сообщение о том что нагрузка четвёртого канала отключе
// Восстановление работы 4 канала: //
// pwrkey.resCurrentProtection(4); // Перезапускаем защиту 4 канала.
// pwrkey.digitalWrite(4, HIGH); // Включаем 4 канал модуля (если ток по прежнему превышает указа
} //
delay(1000); //
} //

```

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:

```
#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми
iarduino_I2C_Relay pwrkey(0x09); // Объявляем объект pwrkey для работы с функциями и методами
// Если объявить объект без указания адреса (iarduino_I2C_Relay pwrkey);
//
void setup(){ // Иницируем передачу данных в монитор последовательного порта
  Serial.begin(9600); // Иницируем работу с модулем.
  pwrkey.begin(); // Разрешаем работу сторожевого таймера модуля, задаём время ожидания
  pwrkey.enableWDT(5); // Отключить работу сторожевого таймера модуля можно
} //
void loop(){ //
// Переключаем 1 и 2 каналы модуля: //
  pwrkey.digitalWrite(1,HIGH); pwrkey.digitalWrite(2,LOW); delay(500); // Включаем 1 канал, выключаем 2 канал и ждём 500 мс.
  pwrkey.digitalWrite(2,HIGH); pwrkey.digitalWrite(1,LOW); delay(500); // Включаем 2 канал, выключаем 1 канал и ждём 500 мс.
// Сбрасываем (перезапускаем) сторожевой таймер модуля: //
  pwrkey.resetWDT(); // Теперь таймер модуля начнёт заново отсчитывать 5 секунд
// Сообщаем, что сработал сторожевой таймер: //
  if( pwrkey.getStateWDT() == false ){ Serial.println("ERROR"); } // Если таймер отключился, значит он досчитал до 0 и
} // Если модуль не отвечает (отключилась шина I2C), то
```

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:

```
uint8_t i = 0x09; // Назначаемый модулю новый адрес (0x07 < адрес < 0x7F).
#include <Wire.h> // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с реле и силовыми ключами.
iarduino_I2C_Relay pwrkey; // Объявляем объект pwrkey для работы с функциями и методами биб
// Если при объявлении объекта указать адрес, например, pwrkey(0

void setup(){
    Serial.begin(9600); // Выводим информацию о модуле:
    // Иницилируем работу с модулем: (метод pwrkey.begin(); вернёт true при успехе).
    if( pwrkey.begin() ){
        Serial.print( "На шине I2C " );
        Serial.print( "найден модуль с адресом 0x" );
        Serial.print( pwrkey.getAddress(), HEX );
        Serial.print( ", который является " );

        switch( pwrkey.getModel() ){
            case DEF_MODEL_2RM: Serial.print( "электромеханическим реле на 2-канала" );
            case DEF_MODEL_4RT: Serial.print( "твердотельным реле на 4-канала" );
            case DEF_MODEL_4NC: Serial.print( "силовым ключом на 4 N-канала с измерением тока" );
            case DEF_MODEL_4PC: Serial.print( "силовым ключом на 4 P-канала с измерением тока" );
            case DEF_MODEL_4NP: Serial.print( "силовым ключом на 4 N-канала до 10A" );
            case DEF_MODEL_4PP: Serial.print( "силовым ключом на 4 P-канала до 10A" );
            default: Serial.print( "неизвестным силовым ключом или реле" );
        }
        Serial.print( ".\r\n" );

        // Меняем адрес модуля на «i»: (метод pwrkey.changeAddress(новый адрес); вернёт true при успехе ).
        if(pwrkey.changeAddress(i)){
            Serial.print( "Адрес модуля изменён на 0x" );
            Serial.print( pwrkey.getAddress(), HEX );
            Serial.print( " и сохранён в flash память модуля." );
        }else{
            Serial.print( "Адрес модуля изменить не удалось!" ); } // Если метод relay.
```

```
    }else{          Serial.print( "нет ни силовых ключей, ни реле!"          ); } // Если метод relay.  
                  Serial.print( "\r\n"                                );  
}  
void loop(){}
```

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

В первой строке скетча определяется переменная «i» с указанием адреса которой будет присвоен модулю (это значение Вы можете изменить на то, которое требуется Вам).

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

Описание функций библиотеки:

В данном разделе описаны функции [библиотеки iarduino_I2C_Relay](#) для работы с модулями силовых ключей поддерживающих функции измерения тока.

Данная библиотека может использовать как аппаратную, так и программную реализацию шины I2C. О том как выбрать тип шины I2C рассказано в статье [Wiki - расширенные возможности библиотек iarduino для шины I2C](#).

Подключение библиотеки:

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

```
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с модулями силовых ключей.  
iarduino_I2C_Relay pwrkey(0x09); // Создаём объект pwrkey для работы с функциями и методами библиотеки iarduino_I2C_Relay, ук
```

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

```
#include <iarduino_I2C_Relay.h> // Подключаем библиотеку для работы с модулями силовых ключей.  
iarduino_I2C_Relay pwrkey; // Создаём объект pwrkey для работы с функциями и методами библиотеки iarduino_I2C_Relay.
```

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: `bool` - результат инициализации (`true` или `false`).
- Примечание: По результату инициализации можно определить наличие модуля на шине.
- Пример:

```
if( pwrkey.begin() ){ Serial.print( "Модуль найден и инициирован!" ); }  
else { Serial.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( pwrkey.reset() ){ Serial.print( "Модуль перезагружен" ); }  
else { Serial.print( "Модуль не перезагружен" ); }
```

Функция `changeAddress()`;

- Назначение: Смена адреса модуля на шине I2C.
- Синтаксис: `changeAddress(АДРЕС);`
- Параметры:

- uint8_t АДРЕС - новый адрес модуля на шине I2C (целое число от 0x08 до 0x7E)
- Возвращаемое значение: bool - результат смены адреса (true или false).
- Примечание: Текущий адрес модуля можно узнать функцией getAddress().
- Пример:

```
if( pwrkey.changeAddress(0x12) ){ Serial.print( "Адрес модуля изменён на 0x12" ); }  
else { Serial.print( "Не удалось изменить адрес" ); }
```

Функция getAddress();

- Назначение: Запрос текущего адреса модуля на шине I2C.
- Синтаксис: getAddress();
- Параметры: Нет.
- Возвращаемое значение: uint8_t 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:

```
Serial.print( "Адрес модуля на шине I2C = 0x");  
Serial.println( pwrkey.getAddress(), HEX );
```

Функция getVersion();

- Назначение: Запрос версии прошивки модуля.
- Синтаксис: getVersion();
- Параметры: Нет
- Возвращаемое значение: uint8_t ВЕРСИЯ - номер версии прошивки от 0 до 255.
- Пример:

```
Serial.print( "Версия прошивки модуля ");  
Serial.println( pwrkey.getVersion(), HEX );
```

Функция getModel();

- Назначение: Запрос типа модуля.
- Синтаксис: getModel();
- Параметры: Нет.
- Возвращаемое значение: uint8_t МОДЕЛЬ - идентификатор модуля от 0 до 255.
- Примечание: По идентификатору можно определить тип модуля (см. пример).
- Пример:

```
switch( pwrkey.getModel() ){
  case DEF_MODEL_2RM: Serial.print( "электромеханическое реле на 2-канала" );      break;
  case DEF_MODEL_4RT: Serial.print( "твердотельное реле на 4-канала" );          break;
  case DEF_MODEL_4NC: Serial.print( "силовой ключ на 4 N-канала с измерением тока" ); break;
  case DEF_MODEL_4PC: Serial.print( "силовой ключ на 4 P-канала с измерением тока" ); break;
  case DEF_MODEL_4NP: Serial.print( "силовой ключ на 4 N-канала до 10А" );      break;
  case DEF_MODEL_4PP: Serial.print( "силовой ключ на 4 P-канала до 10А" );      break;
  default:           Serial.print( "неизвестный модуль" );                      break;
}
```

Функция digitalWrite();

- Назначение: Установка логического уровня на одном или всех каналах модуля.
- Синтаксис: digitalWrite(КАНАЛ , УРОВЕНЬ);
- Параметры:
 - uint8_t КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.
 - uint8_t 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:

```
pwrkey.digitalWrite( ALL_CHANNEL, LOW ); // Отключить все каналы.  
pwrkey.digitalWrite( 2, HIGH ); // Включить второй канал.  
pwrkey.digitalWrite( 3, 1 ); // Включить третий канал.
```

Функция digitalRead();

- Назначение: Чтение логического уровня на одном канале.
- Синтаксис: digitalRead(КАНАЛ);
- Параметры:
 - uint8_t КАНАЛ - номер канала силового ключа, значение от 1 до 4.
- Возвращаемое значение: uint8_t 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:

```
                Serial.print( "На четвёртом канале установлен " );  
if( pwrkey.digitalRead( 4 ) == HIGH ){ Serial.print( "высокий " ); }  
else                { Serial.print( "низкий " ); }  
                Serial.print( "логический уровень.\r\n" ); }
```

Функция analogWrite();

- Назначение: Установка сигнала ШИМ с требуемым коэффициентом заполнения.
- Синтаксис: analogWrite(КАНАЛ , УРОВЕНЬ);
- Параметры:
 - uint8_t КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.
 - uint16_t 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, светодиодах и т.д
- Пример:

```
pwrkey.analogWrite( 1, 0 ); // Отключить первый канал.  
pwrkey.analogWrite( 1, 2047 ); // Включить первый канал на половину мощности.  
pwrkey.analogWrite( 1, 4095 ); // Включить первый канал на полную мощность.
```

Функция analogRead();

- Назначение: Чтение уровня сигнала ШИМ.
- Синтаксис: analogRead(КАНАЛ);
- Параметры:
 - uint8_t КАНАЛ - номер канала силового ключа, значение от 1 до 4.
- Возвращаемое значение: uint16_t 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:

```
Serial.print( "На четвёртом канале установлен сигнал ШИМ с уровнем ");  
Serial.print( pwrkey.analogRead( 4 ) );  
Serial.print( " из 4095.\r\n");
```

Функция currentRead();

- Назначение: Чтение силы тока проходящего через указанный канал модуля.
- Синтаксис: currentRead(КАНАЛ);
- Параметры:

- uint8_t 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:

```
Serial.print( "Сила тока на 3 канале равна " );  
Serial.print( pwrkey.currentRead(3) );  
Serial.print( " A.\r\n" );
```

Функция setCurrentProtection();

- Назначение: Установка защиты канала модуля от превышения тока.
- Синтаксис: setCurrentProtection(КАНАЛ , ТОК , РЕЖИМ);
- Параметры:
 - uint8_t КАНАЛ - номер канала силового ключа, от 1 до 4, или значение ALL_CHANNEL.
 - float ТОК - сила тока превышение которой должно вызвать защиту (от 0,1 до 25,5 А).
 - uint8_t 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:

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

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:
 - `uint8_t КАНАЛ` - номер канала силового ключа, от 1 до 4, или значение `ALL_CHANNEL`.
- Возвращаемое значение: Нет.
- Пример:

```
pwrkey.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( pwrkey.getCurrentProtection(4) ){ // Если на четвёртом канале выполнено отключение нагрузки в связи с превышением тока, то
  pwrkey.resCurrentProtection(4); // Перезапускаем защиту 4 канала.
  pwrkey.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:
 - uint8_t КАНАЛ - номер канала силового ключа, значение от 1 до 4.
- Возвращаемое значение: bool - 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) ){ Serial.println( "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:

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

Функция `currentWrite()`;

- Назначение: Калибровка измерения тока функцией `currentRead()`.
- ВАЖНО: Результат калибровки сохраняется в энергонезависимую память модуля! Вызов данной функции с некорректными значениями тока, приведет к неправильным измерениям силы тока функцией `currentRead()`, даже после отключения питания.
- Синтаксис: `currentWrite(КАНАЛ , ТОК);`
- Параметры:
 - `uint8_t КАНАЛ` - номер канала силового ключа, значение от 1 до 4.
 - `float ТОК` - сила тока проходящая через нагрузку канала (измеренная амперметром).

- Возвращаемое значение: Нет.
- Примечание:
 - Для улучшения расчёта силы тока функцией `currentRead()`, нужно скорректировать реальное значение шунтирующего сопротивления включённого в цепь канала на плате модуля, что и делает функция `currentWrite()`.
 - Для корректировки нужно подключить нагрузку (с током потребления в районе 0,8 - 1,2 А) к требуемому каналу через амперметр, включить канал функцией `digitalWrite()` и вызвать функцию `currentWrite()` указав номер канала и показания амперметра в Амперах (с указанием 2-3 знаков после запятой).
 - Результат корректировки сохраняется в энергонезависимую память модуля, а значит точность показаний дальнейших измерений тока сохранится и после отключения питания модуля
 - Скетч корректировки есть в примерах библиотеки под названием «`currentWrite`».
 - Если вызвать функцию `currentWrite()`, указав в качестве дополнительного (третьего) параметра имя переменной типа `uint8_t`, то вместо калибровки, функция рассчитает значение сопротивления шунтирующего резистора (в мОм) и вернёт его в указанную переменную. Данное значение должно находиться в районе 50 мОм.
- Пример:

```
pwrkey.currentWrite( 1 , 1.037 ); // Калибровать 1 канал, измеренное значение тока 1.037 А.
```

Функция `enableWDT()`;

- Назначение: Запуск (разрешение работы) сторожевого таймера модуля.
- Синтаксис: `enableWDT(ВРЕМЯ);`
- Параметры:
 - `uint8_t ВРЕМЯ` - количество секунд от 1 до 254.
- Возвращаемое значение: `bool` - 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:

```
pwrkey.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:

```
pwrkey.disableWDT(); // Отключение сторожевого таймера.
```

Функция `resetWDT()`;

- Назначение: Сброс (перезагрузка) сторожевого таймера.
- Синтаксис: `resetWDT()`;
- Параметры: Нет
- Возвращаемое значение: `bool` - 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:

```
pwrkey.enableWDT( 10 ); // Через 10 секунд модуль перезагрузится;
```

```
delay(9000);           // Ждём 9 секунд.  
pwrkey.resetWDT();    // Модуль перезагрузится не через 1 секунду, а через 10.
```

Функция getStateWDT();

- Назначение: Чтение состояния сторожевого таймера.
- Синтаксис: getStateWDT();
- Параметры: Нет.
- Возвращаемое значение: bool - 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( pwrkey.getStateWDT() ){ Serial.println( "таймер выполняет отсчёт времени." ); }  
else { Serial.println( "сторожевой таймер отключен." ); }
```