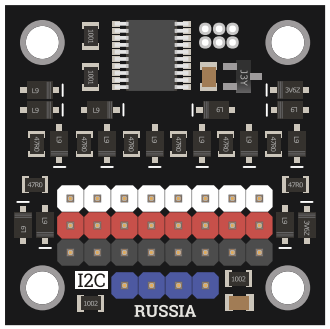
Расширитель выводов, FLASH-I2C (Trema-модуль)



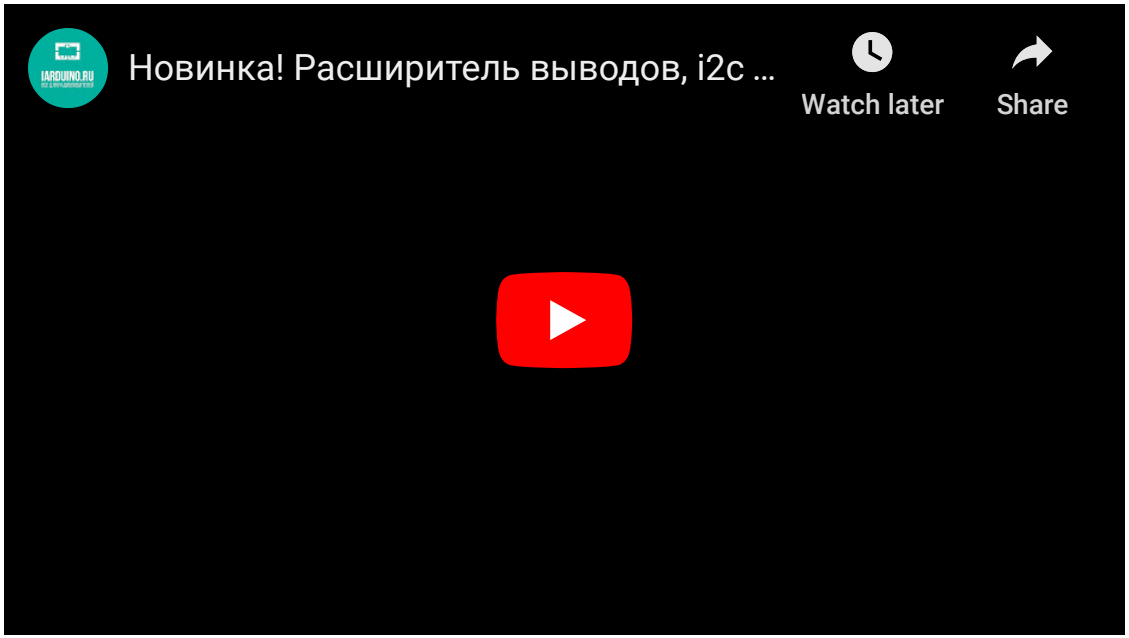
Общие сведения:

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

Модуль относится к серии «Flash», а значит к одной шине I2C можно подключить более 100 модулей, так как их адрес на шине I2C (по умолчанию 0x09), хранящийся в энергонезависимой памяти, можно менять программно.

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

Видео:

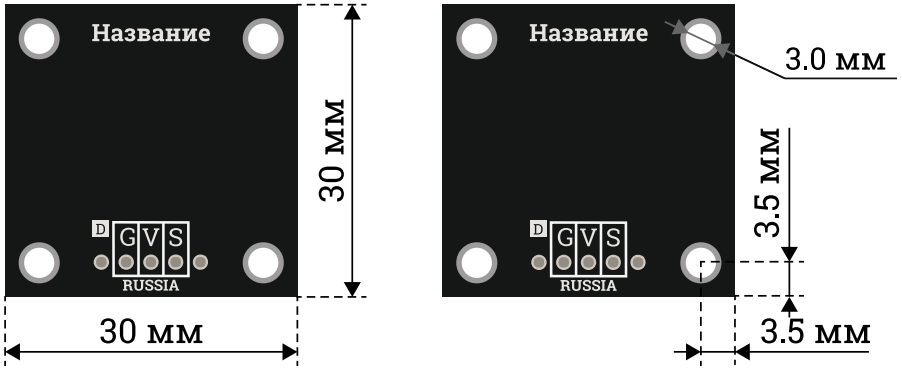


Спецификация:

- Напряжение питания: 3,3 или 5 В (постоянного тока)
- Потребляемый ток: до 6 мА.
- Напряжение логических уровней: 3,3 В (все выводы толерантны к 5 В).
- Напряжение аналоговых уровней: до 3,3 В (все выводы толерантны к 5 В).
- Разрешение АЦП: 12 бит (значение от 0 до 4095).
- Разрешение ШИМ: 12 бит (значение от 0 до 4095).
- Частота ШИМ: 1 - 12'000 Гц (по умолчанию 490 Гц).
- Количество цифровых выводов: 8 (работают как на вход, так и на выход).
- Количество аналоговых входов: 8 (АЦП).
- Количество выходов с поддержкой ШИМ: 4 (выводы № 0 - 3).

- Интерфейс: I2C.
- Скорость шины I2C: 100 кбит/с.
- Адрес на шине I2C: устанавливается программно (по умолчанию 0x09).
- Уровень логической 1 на линиях шины I2C: 3,3 В (толерантны к 5 В).
- Рабочая температура: от -40 до +65 °C.
- Габариты: 30 x 30 мм.
- Вес: 7 г.

Все модули линейки "Trema" выполнены в одном формате



Подключение:

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

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

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

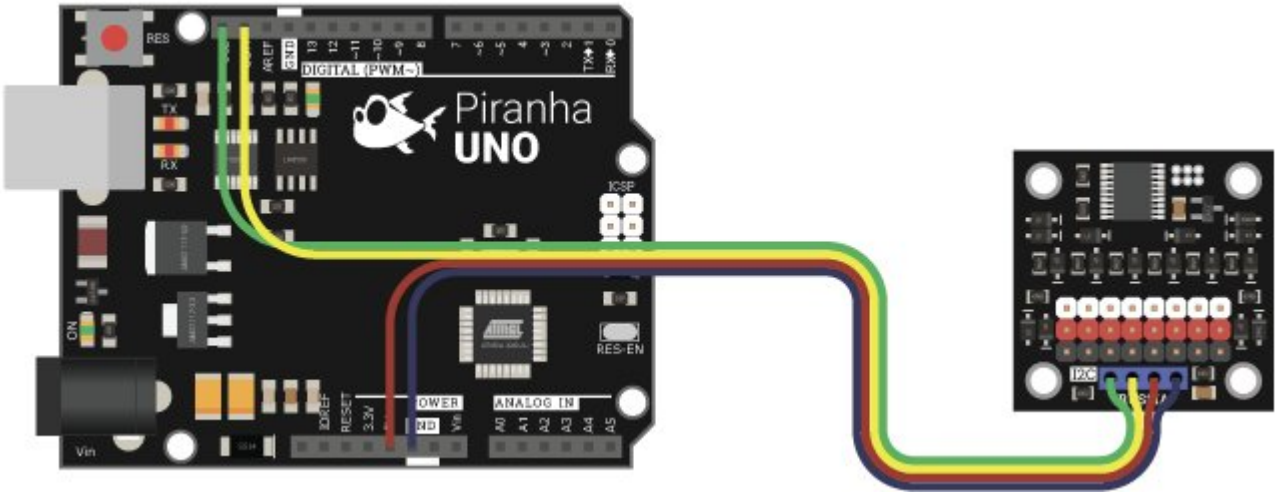
Trema модуль - Расширитель выводов, I2C-flash подключается к аппаратной или программной шине I2C Arduino.

В комплекте имеется кабель для быстрого и удобного подключения модуля к колодке I2C на Trema Shield. Если на шине I2C уже имеется другое устройство, то для подключения модуля, предлагаем воспользоваться I2C Hub.

Модуль удобно подключать 3 способами, в зависимости от ситуации:

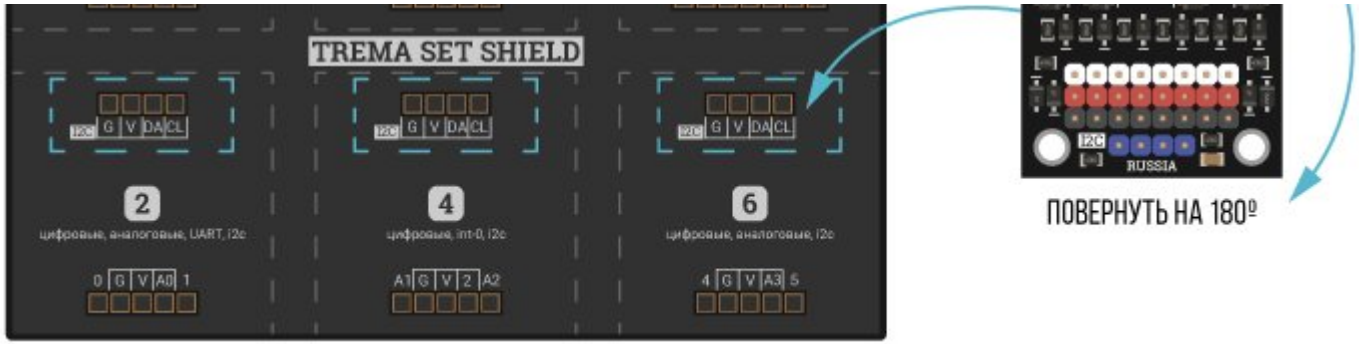
Способ - 1 : Используя проводной шлейф и Piranha UNO

Используя провода «Папа — Мама», подключаем напрямую к контроллеру Piranha UNO.



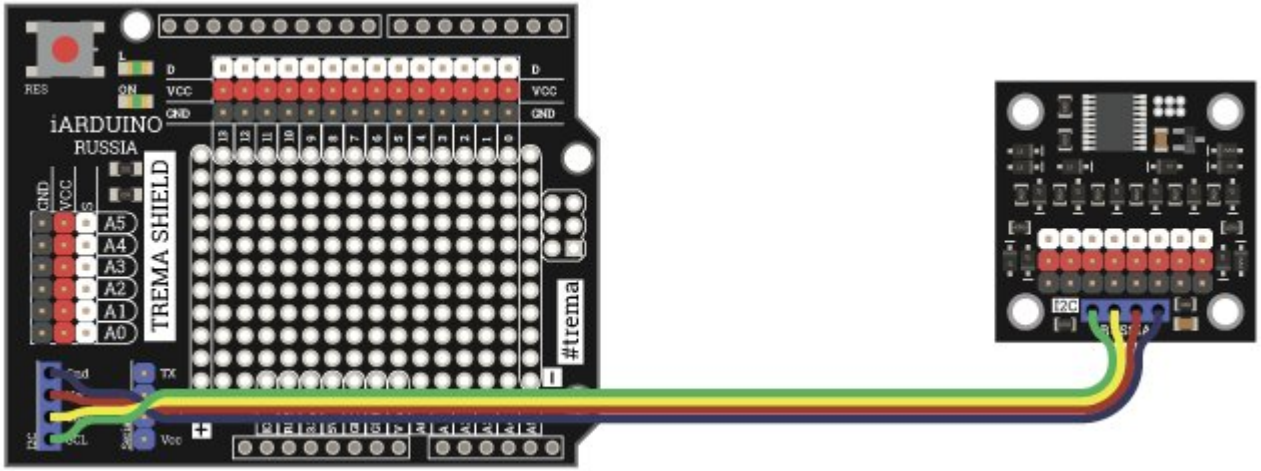
Способ - 2 : Используя Trema Set Shield

Модуль можно подключить к любому из I2C входов Trema Set Shield.



Способ - 3 : Используя проводной шлейф и Shield

Используя 4-х проводной шлейф, к Trema Shield, Trema-Power Shield, Motor Shield, Trema Shield NANO и тд.



Питание:

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

Подробнее о модуле:

Модуль построен на базе микроконтроллера STM32F030F4 и снабжен собственным стабилизатором напряжения. У модуля имеются 8

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

Модуль позволяет:

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

Специально для работы с Trema модулем - Расширитель выводов, I2C-flash , нами разработана библиотека iarduino_I2C_Expander 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_Expander.h>                                          //   Подключаем библиотеку для работы с модулем
iarduino_I2C_Expander gpio;                                                 //   Объявляем объект gpio для работы с функциями и метода
                                                                            //   Если при объявлении объекта указать адрес, например,
void setup(){                                                               //
    Serial.begin(9600);                                                     //
    if( gpio.begin() ){                                                     //   Инициируем работу с расширителем выводов.
        Serial.print("На шине I2C найден модуль с адресом 0x");           //
        Serial.print( gpio.getAddress(), HEX );                             //   Выводим текущий адрес модуля.
        Serial.print(" который является расширителем выводов\r\n");         //
        if( gpio.changeAddress(newAddress) ){                               //   Меняем адрес модуля на newAddress.
            Serial.print("Адрес модуля изменён на 0x");                     //
            Serial.println( gpio.getAddress(), HEX );                       //   Выводим текущий адрес модуля.
        }else{                                                              //
            Serial.println("Адрес модуля изменить не удалось!");           //
        }                                                                   //
    }else{                                                                  //
        Serial.println("Расширитель выводов не найден!");                   //
    }                                                                       //
}                                                                           //
                                                                            //
void loop(){}                                                               //
```

Примеры:

В данном разделе раскрыты примеры работы модуля по шине I2C с использованием библиотеки iarduino_I2C_Expander. Сама библиотека

содержит больше примеров, доступных из меню Arduino IDE: Файл / Примеры / iarduino I2C Expander (расширитель выводов).

Работа с логическими уровнями:

```
#include <Wire.h>                                                   //  Подключаем библиотеку для работы с аппаратной шиной I
#include <iarduino_I2C_Expander.h>                                  //  Подключаем библиотеку для работы с расширителем вывод
iarduino_I2C_Expander gpio(0x09);                                   //  Объявляем объект gpio для работы с функциями и метода
                                                                    //  Если объявить объект без указания адреса (iarduino_I2
void setup(){                                                       //
    gpio.begin();                                                   //  Инициируем работу с расширителем выводов.
    gpio.pinMode(0, INPUT,  DIGITAL);                               //  Конфигурируем вывод 0 на работу в качестве цифрового
//  gpio.pinPull(0, PULL_UP);                                       //  Подтягиваем вход 0 до уровня VCC через внутренний рез
//  gpio.pinPull(0, PULL_DOWN);                                     //  Прижимаем   вход 0 к  уровню GND через внутренний рез
    gpio.pinMode(1, OUTPUT, DIGITAL);                               //  Конфигурируем вывод 1 на работу в качестве цифрового
}                                                                   //
                                                                    //
void loop(){                                                        //
    bool level;                                                     //  Объявляем переменную «level».
    level = gpio.digitalRead(0);                                    //  Читаем логический уровень с вывода №0 в переменную «l
    gpio.digitalWrite(1, level);                                    //  Устанавливаем на выводе №1 уровень равный значению «l
}                                                                   //
```

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

Работа с аналоговыми уровнями:

```
#include <Wire.h>                                                   //  Подключаем библиотеку для работы с аппаратной шиной I
#include <iarduino_I2C_Expander.h>                                  //  Подключаем библиотеку для работы с расширителем вывод
iarduino_I2C_Expander gpio(0x09);                                   //  Объявляем объект gpio для работы с функциями и метода
                                                                    //  Если объявить объект без указания адреса (iarduino_I2
```

```
void setup(){                                                    //
    gpio.begin();                                                //   Инициируем работу с расширителем выводов.
    gpio.pinMode(0, INPUT,  ANALOG);                             //   Конфигурируем вывод 0 на работу в качестве аналоговог
    gpio.pinMode(1, OUTPUT, ANALOG);                             //   Конфигурируем вывод 1 на работу в качестве аналоговог
}                                                                //
                                                                 //
void loop(){                                                     //
    uint16_t level;                                              //   Объявляем переменную «level».
    level = gpio.analogRead(0);                                  //   Читаем аналоговый уровень с вывода №0 в переменную «l
    gpio.analogWrite(1, level);                                  //   Устанавливаем на выводе №1 уровень равный значению «l
}                                                                //
```

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

Чтение логического уровня с аналогового входа:

```
#include <Wire.h>                                                //   Подключаем библиотеку для работы с аппаратной шиной I
#include <iarduino_I2C_Expander.h>                               //   Подключаем библиотеку для работы с расширителем вывод
iarduino_I2C_Expander gpio(0x09);                                //   Объявляем объект gpio для работы с функциями и метода
                                                                 //   Если объявить объект без указания адреса (iarduino_I2
void setup(){                                                    //
    gpio.begin();                                                //   Инициируем работу с расширителем выводов.
    gpio.pinMode(0, INPUT,  ANALOG );                            //   Конфигурируем вывод 0 на работу в качестве аналоговог
    gpio.pinMode(1, OUTPUT, DIGITAL);                            //   Конфигурируем вывод 1 на работу в качестве цифрового
    gpio.levelWrite(512);                                        //   Указываем аналоговый уровень разделяющий уровень логи
    gpio.levelHyst (12);                                         //   Указываем гистерезис аналогового уровня в ±12.
}                                                                //
                                                                 //
void loop(){                                                     //
```

```
    bool level;                                                    //   Объявляем переменную «level».
    level = gpio.levelRead(0);                                     //   Читаем логический уровень с аналогового вывода №0 в г
    gpio.digitalWrite(1, level);                                   //   Устанавливаем на выводе №1 уровень равный значению «l
}                                                                  //
```

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

Управление сервоприводами:

```
#include <Wire.h>                                                  //   Подключаем библиотеку для работы с аппаратной шиной I
#include <iarduino_I2C_Expander.h>                                 //   Подключаем библиотеку для работы с расширителем вывод
iarduino_I2C_Expander gpio(0x09);                                  //   Объявляем объект gpio для работы с функциями и метода
                                                                   //   Если объявить объект без указания адреса (iarduino_I2
void setup(){                                                      //
    gpio.begin();                                                  //   Инициируем работу с расширителем выводов.
    gpio.pinMode(0, OUTPUT, SERVO);                                //   Конфигурируем вывод 0 на работу в качестве выхода для
    gpio.pinMode(1, OUTPUT, SERVO);                                //   Конфигурируем вывод 1 на работу в качестве выхода для
//  gpio.servoAttach(0, 500, 2500, 0, 180);                        //   Определяем параметры сервопривода: вывод 0, длительно
//  gpio.servoAttach(1, 513, 2430, 0, 180);                        //   Определяем параметры сервопривода: вывод 1, длительно
}                                                                  //
                                                                   //
void loop(){                                                       //
    gpio.servoWrite(0, 170); gpio.servoWrite(1, 100); delay(500);  //   Поворачиваем сервопривод подключённый к выводу 0 в уг
    gpio.servoWrite(0,  10); gpio.servoWrite(1,  10); delay(500);  //   Поворачиваем оба сервопривода (на выводах 0 и 1) в уг
}                                                                  //
```

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

Описание функций библиотеки:

В данном разделе описаны функции библиотеки iarduino_I2C_Expander для работы с Trema модулем - Расширитель выводов, I2C-flash.

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_Expander.h>  // Подключаем библиотеку для работы с модулем.
iarduino_I2C_Expander gpio(0x09);   // Создаём объект gpio для работы с функциями и методами библиотеки iarduino_I2C_Expander,
```

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

```
#include <iarduino_I2C_Expander.h>  // Подключаем библиотеку для работы с модулем.
iarduino_I2C_Expander gpio;         // Создаём объект gpio для работы с функциями и методами библиотеки iarduino_I2C_Expander.
```

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( gpio.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( gpio.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( gpio.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( gpio.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( gpio.getVersion() );
```

Функция pinMode();

- Назначение: Конфигурирование вывода на требуемый режим работы.
- Синтаксис: pinMode(НОМЕР ВЫВОДА , НАПРАВЛЕНИЕ РАБОТЫ , ТИП СИГНАЛА);
- Параметры:
  - uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7, или значение ALL_PIN.

    - uint8_t НАПРАВЛЕНИЕ РАБОТЫ - может принимать одно из двух значений:
        - INPUT - вывод работает как вход (по умолчанию).
        - OUTPUT - вывод работает как выход.
    - uint8_t ТИП СИГНАЛА (не обязательный параметр):
        - DIGITAL - вывод работает как цифровой вход или выход (по умолчанию).
        - ANALOG - вывод работает как аналоговый вход, или выход с ШИМ.
        - SERVO - вывод работает с сервоприводом.
        - Если тип сигнала не указан, то он остаётся без изменений.
- Возвращаемое значение: Нет.
- Примечание:
    - При конфигурировании хотя бы одного вывода на работу с сервоприводом, частота ШИМ на всех выводах поддерживающих ШИМ упадёт до 50 Гц.
    - Функция pinMode() является обязательной только для работы с сервоприводами. В остальных случаях, при попытке чтения с выхода или записи на вход, вывод будет автоматически переконфигурирован (но на это уйдёт время необходимое для отправки одного дополнительного пакета по шине I2C).
- Пример:

```
gpio.pinMode( ALL_PIN, INPUT, ANALOG ); // Конфигурируем все выводы на работу в качестве аналоговых входов.
gpio.pinMode( 2, OUTPUT );              // Переконфигурируем вывод №  2 на работу в качестве выхода, так как тип сигнала не ук
```

Функция pinPull();

- Назначение: Внутрисхемное подключение резистора к выводу.
- Синтаксис: pinPull(НОМЕР ВЫВОДА , РЕЗИСТОР);
- Параметры:
    - uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7, или значение ALL_PIN.
    - uint8_t РЕЗИСТОР - может принимать одно из двух значений:
        - PULL_UP - подключить резистор подтягивающий вывод до уровня логической 1.
        - PULL_DOWN - подключить резистор прижимающий вывод к уровню логического 0.

- Возвращаемое значение: Нет.
- Примечание:
  - Подключать внутрисхемные резисторы можно к любому выводу, в т.ч. и сконфигурированному в качестве выхода, например, по схеме с открытым стоком.
- Пример:

```
gpio.pinPull( 5, PULL_UP ); // Подтянуть вывод № 5 до уровня Vcc.
```

Функция pinOutScheme();

- Назначение: Выбор внутренней схемы включения выхода.
- Синтаксис: pinOutScheme(НОМЕР ВЫВОДА , СХЕМА);
- Параметры:
  - uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7, или значение ALL_PIN.
  - uint8_t СХЕМА - может принимать одно из двух значений:
    - OUT_PUSH_PULL - двухтактная схема (по умолчанию).
    - OUT_OPEN_DRAIN - схема с открытым стоком.
- Возвращаемое значение: Нет.
- Примечание:
  - Схему можно указать как до, так и после конфигурирования вывода на работу в качестве выхода.
  - При двухтактной схеме работы, что на выход записывается, то на нём и появляется.
  - При использовании схемы с открытым стоком, выход работает по иному. Если записать 0 то и на выходе появится 0, а если запись 1, то выход перейдёт в состояние высокого импеданса. В таких схемах часто подтягивают выход до уровня логической 1, это можно сделать функцией pinPull().
- Пример:

```
gpio.pinOutScheme( 0, OUT_OPEN_DRAIN ); // Если вывод № 0 сконфигурирован как выход, то он является выходом с открытым стоком.
```

Функция digitalRead();

- Назначение: Чтение логического уровня с вывода.
- Синтаксис: digitalRead(НОМЕР ВЫВОДА);
- Параметр: uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7, или значение ALL_PIN.
- Возвращаемое значение: uint8_t ЛОГИЧЕСКИЙ УРОВЕНЬ - значение HIGH (1), или LOW (0).
- Примечание:
  - Если вывод был сконфигурирован как выход, то он будет автоматически переконфигурирован на работу в качестве входа.
  - Если вывод был сконфигурирован как аналоговый, то он будет автоматически переконфигурирован как цифровой.
  - Если в качестве параметра указать не номер вывода, а значение ALL_PIN, то функция вернёт не логический уровень, а число, каждый бит которого соответствует логическому уровню считанному с вывода номер которого совпадает с номером бита.
- Пример:

```
bool i = gpio.digitalRead( 0 ); // Считать логический уровень с вывода № 0.
bool j = gpio.digitalRead( ALL_PIN ) & bit(0); // Узнать логический уровень на выводе № 0.
```

Функция digitalWrite();

- Назначение: Установка логического уровня на выводе.
- Синтаксис: digitalWrite(НОМЕР ВЫВОДА , ЛОГИЧЕСКИЙ УРОВЕНЬ);
- Параметры:
  - uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7, или значение ALL_PIN.
  - uint8_t ЛОГИЧЕСКИЙ УРОВЕНЬ - значение HIGH (1), или LOW (0).
- Возвращаемое значение: Нет.
- Примечание:
  - Если вывод был сконфигурирован как вход, то он будет автоматически переконфигурирован на работу в качестве выхода.
  - Если вывод был сконфигурирован как аналоговый, то он будет автоматически переконфигурирован как цифровой.
- Пример:

```
gpio.digitalWrite( 7, LOW ); // Установить низкий логический уровень на выводе № 7.
```

Функция analogRead();

- Назначение: Чтение аналогового (АЦП) уровня с вывода.
- Синтаксис: analogRead(НОМЕР ВЫВОДА);
- Параметр: uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7.
- Возвращаемое значение: uint16_t УРОВЕНЬ - целое число от 0 до 4095.
- Примечание:
  - Если вывод был сконфигурирован как выход, то он будет автоматически переконфигурирован на работу в качестве входа.
  - Если вывод был сконфигурирован как цифровой, то он будет автоматически переконфигурирован как аналоговый.
- Пример:

```
uint16_t i = gpio.analogRead( 7 ); // Считать аналоговый уровень с вывода № 7.
```

Функция analogWrite();

- Назначение: Установка аналогового (ШИМ) уровня на выводе.
- Синтаксис: analogWrite(НОМЕР ВЫВОДА , УРОВЕНЬ);
- Параметры:
  - uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7, или значение ALL_PIN.
  - uint16_t УРОВЕНЬ - целое число от 0 до 4095.
- Возвращаемое значение: Нет.
- Примечание:
  - Если вывод был сконфигурирован как вход, то он будет автоматически переконфигурирован на работу в качестве выхода.
  - Если вывод был сконфигурирован как цифровой, то он будет автоматически переконфигурирован как аналоговый.
  - Сигналы ШИМ поддерживают только выводы 0 - 3, попытка записи аналогового (ШИМ) уровня на выводы 4 - 7 приведёт к установке на них логических уровней:
    - Значения от 0 до 2047 приведут к установке уровня логического 0.
    - Значения от 2048 до 4095 приведут к установке уровня логической 1.
- Пример:

```
gpio.analogWrite( 1, 1023 ); // Установить аналоговый (ШИМ) сигнал с 25% заполнением.
```

Функция analogAveraging();

- Назначение: Установка коэффициента усреднения показаний АЦП.
- Синтаксис: analogAveraging(УСРЕДНЕНИЕ);
- Параметр: uint8_t УСРЕДНЕНИЕ - значение от 0 (не усреднять) до 255 (макс. усреднение).
- Возвращаемое значение: Нет.
- Примечание:
  - Чем выше значение усреднения, тем плавнее будет меняться аналоговый уровень считываемый функцией analogRead(). Слишком высокое усреднение приведёт к большой инерционности показаний, а слишком маленькое усреднение приведёт к «скачкам» показаний. Значение по умолчанию 127 (50% от максимального значения).
  - Усреднение применяется ко всем выводам модуля.
- Пример:

```
gpio.analogAveraging( 10 ); // Установить небольшое усреднение показаний АЦП.
```

Функция levelRead();

- Назначение: Чтение логического уровня с аналогового входа.
- Синтаксис: levelRead(НОМЕР ВЫВОДА);
- Параметр: uint8_t НОМЕР ВЫВОДА - целое число от 0 до 7, или значение ALL_PIN.
- Возвращаемое значение: uint8_t ЛОГИЧЕСКИЙ УРОВЕНЬ - значение HIGH (1), или LOW (0).
- Примечание:
  - Если вывод был сконфигурирован как выход, то он будет автоматически переконфигурирован на работу в качестве входа.
  - Если вывод был сконфигурирован как цифровой, то он будет автоматически переконфигурирован как аналоговый.
  - Если в качестве параметра указать не номер вывода, а значение ALL_PIN, то функция вернёт не логический уровень, а число, каждый бит которого соответствует логическому уровню считанному с вывода номер которого совпадает с номером бита.
  - Данная функция представляет интерес тем, что граница между логическим 0 и логической 1 может быть установлена функцией

levelWrite(), что позволяет считывать данные со схем, логические уровни которых ниже 3,3 В.

- Пример: Указан ниже, сразу для трёх функций: levelRead(), levelWrite() и levelHyst().

Функция levelWrite();

- Назначение: Указание аналогового уровня являющегося границей между логическим 0 и 1.
- Синтаксис: levelWrite(ГРАНИЦА);
- Параметр: uint16_t ГРАНИЦА - целое число от 0 до 4095 (по умолчанию 2047).
- Возвращаемое значение: Нет.
- Примечание:
  - Данная функция указывает границу по которой функция levelRead() преобразует считанное с вывода аналоговое значение в логические уровни. Если аналоговое значение ниже границы и гистерезиса - это 0, а если выше границы и гистерезиса - это 1.
  - Гистерезис это значение указываемое функцией levelHyst(), оно расширяет границу.
- Пример: Указан ниже, сразу для трёх функций: levelRead(), levelWrite() и levelHyst().

Функция levelHyst();

- Назначение: Указание гистерезиса для границы АЦП между логическими 0 и 1.
- Синтаксис: levelHyst(ГИСТЕРЕЗИС);
- Параметр: uint16_t ГИСТЕРЕЗИС - целое число от 0 до 4095 (по умолчанию 205).
- Возвращаемое значение: Нет.
- Примечание:
  - Данная функция указывает гистерезис который функция levelRead() добавляет к границе для определения логической 1 и вычитает из границы для определения логического 0.
- Пример:

```
gpio.levelWrite(3000);      // Указываем границу равную 3000, для определения 0 и 1.
gpio.levelHyst (100);       // Указываем гистерезис границы равный ±100.
bool i = gpio.levelRead(2); // Читаем логический уровень с аналогового вывода № 2 в переменную «i».
/*
 * Если значение АЦП на выводе №2 меньше 3000 - 100 то функция levelRead() вернёт 0.
```

```
 * Если значение АЦП на выводе №2 больше 3000 + 100 то функция levelRead() вернёт 1.
 * Если значение АЦП на выводе №2 в пределах от 3000-100 до 3000+100 то
 * функция levelRead() вернёт значение которое она возвращала ранее для того же вывода.
 */
```

Функция freqPWM();

- Назначение: Установка частоты ШИМ.
- Синтаксис: freqPWM(ЧАСТОТА);
- Параметр: uint16_t ЧАСТОТА - значение от 1 до 12'000 Гц.
- Возвращаемое значение: Нет.
- Примечание:
  - Частота устанавливается для всех выводов поддерживающих ШИМ.
  - Частота ШИМ не влияет на коэффициент заполнения ШИМ каждого вывода.
  - Частоту можно менять как до установки ШИМ на выводе (выводах), так и после.
- Пример:

```
gpio.freqPWM( 1000 ); // Установить частоту ШИМ в 1 кГц.
```

Функция servoAttach();

- Назначение: Указание настроек для сервопривода на выводе.
- Синтаксис: servoAttach(ВЫВОД , ШИРИНА_MIN , ШИРИНА_MAX [, УГОЛ_MIN , УГОЛ_MAX]);
- Параметры:
  - uint8_t ВЫВОД - целое число от 0 до 3 определяющее № вывода, или значение ALL_PIN.
  - uint16_t ШИРИНА_MIN - целое число от 0 до 20'000 определяющее минимальную длительность импульса ШИМ в микросекундах для управления сервоприводом (значение по умолчанию 500).
  - uint16_t ШИРИНА_MAX - целое число от 0 до 20'000 определяющее максимальную длительность импульса ШИМ в микросекундах для управления сервоприводом (значение по умолчанию 2'500).
  - int16_t УГОЛ_MIN - целое число от -32'768 до +32'767 определяющее минимальный угол поворота сервопривода в градусах (значение

по умолчанию 0°).
    - int16_t УГОЛ_MAX - целое число от -32'768 до +32'767 определяющее максимальный угол поворота сервопривода в градусах (значение по умолчанию 180°).
- Возвращаемое значение: Нет.
- Примечание:
    - Данная функция является необязательной, так как значения по умолчанию подходят для большинства сервоприводов.
    - Функция настроек сервоприводов может быть полезна, если сервопривод имеет угол поворота отличный от 180°, или он не точно поворачивается (не доворачивается или перекручивается на несколько градусов), или Вы желаете ограничить угол поворота сервопривода.
    - Настройки являются индивидуальными для каждого вывода.
- Пример:

```
gpio.pinMode    ( 3, OUTPUT, SERVO );          // Конфигурируем вывод № 3 на работу с сервоприводом.
gpio.servoAttach( 3 , 500 , 2500 , -90 , 90 ); // Настраиваем сервопривод подключённый к выводу № 3.
gpio.servoWrite ( 3 , -90 );     delay( 500 ); // Поворачиваем сервопривод в угол -90° и ждём пол секунды.
gpio.servoWrite ( 3 ,  90 );     delay( 500 ); // Поворачиваем сервопривод в угол  90° и ждём пол секунды.
```

Функция servoWrite();

- Назначение: Поворот ротора сервопривода в указанный угол.
- Синтаксис: servoWrite(НОМЕР ВЫВОДА , УГОЛ ПОВОРОТА);
- Параметры:
    - uint8_t НОМЕР ВЫВОДА - целое число от 0 до 3, или значение ALL_PIN.
    - int16_t УГОЛ ПОВОРОТА - целое число от -32768 до +32767°.
- Возвращаемое значение: Нет.
- Примечание:
    - Перед использованием функции servoWrite() необходимо сконфигурировать вывод модуля на работу с сервоприводом функцией pinMode().
    - Угол поворота сервопривода не должен выходить за пределы указанные функцией servoAttach(), по умолчанию от 0 до 180°.
- Пример:

```
gpio.pinMode    ( 1, OUTPUT, SERVO );        // Конфигурируем вывод № 1 на работу с сервоприводом.
gpio.servoWrite ( 1 ,   0 );  delay( 1000 ); // Поворачиваем сервопривода в угол 0°   и ждём секунду.
gpio.servoWrite ( 1 , 180 );  delay( 1000 ); // Поворачиваем сервопривода в угол 180° и ждём секунду.
```

Функция servoWriteMicroseconds();

- Назначение: Поворот ротора сервопривода указанием ширины импульсов ШИМ.
- Синтаксис: servoWriteMicroseconds(НОМЕР ВЫВОДА , ШИРИНА ИМПУЛЬСОВ);
- Параметры:
  - uint8_t НОМЕР ВЫВОДА - целое число от 0 до 3, или значение ALL_PIN.
  - uint16_t ШИРИНА ИМПУЛЬСОВ - целое число от 0 до 20'000 микросекунд.
- Возвращаемое значение: Нет.
- Примечание:
  - Сервопривод управляется сигналом ШИМ с частотой 50 Гц. Функция servoWrite(), используя настройки указанные servoAttach(), преобразует полученный угол в ширину импульсов ШИМ и устанавливает этот сигнал на указанном выводе. Это удобно, но минимальный угол поворота сервопривода становится равным 1 градусу. Функция servoWriteMicroseconds() позволяет более точно управлять сервоприводом, но Вам самим прийдётся преобразовывать угол поворота в ширину импульсов.
  - Перед использованием функции servoWrite() необходимо сконфигурировать вывод модуля на работу с сервоприводом функцией pinMode().
  - Ширина импульсов не может быть выше 20'000 микросекунд, так как на частоте 50 Гц это время является (периодом) максимальным коэффициентом заполнения ШИМ.
    T = 1/f = 1/50 = 0,02 сек = 20 мс = 20'000 мкс.
- Пример:

```
gpio.pinMode( 1, OUTPUT, SERVO );                       // Конфигурируем вывод № 1 на работу с сервоприводом.
gpio.servoWriteMicroseconds( 1 ,  500 );  delay( 1000 ); // Поворачиваем сервопривода в угол 0°   и ждём секунду.
gpio.servoWriteMicroseconds( 1 , 2500 );  delay( 1000 ); // Поворачиваем сервопривода в угол 180° и ждём секунду.
```

Ссылки:

- Библиотека iarduino_I2C_Expander.
- Расширенные возможности библиотек iarduino для шины I2C.
- Wiki - Установка библиотек в Arduino IDE.