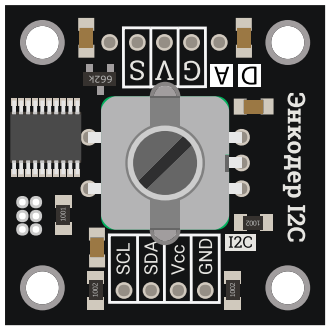
Энкодер, потенциометр, FLASH-I2C



Общие сведения:

Trema модуль - Энкодер, потенциометр, I2C-flash - является устройством ввода данных с подключением по шине I2C. У модуля есть программируемый выход, значительно расширяющий его возможности.

Модуль способен работать как энкодер (отправляя количество тактов поворота вала в одну и другую сторону), как потенциометр (отправляя точное положение вала относительно точки сброса), как тактовая кнопка (отправляя события и состояния кнопки, в т.ч. и время её удержания), а так же модуль способен работать автономно (управляя сигналом на выходе без подключения к шине I2C).

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.

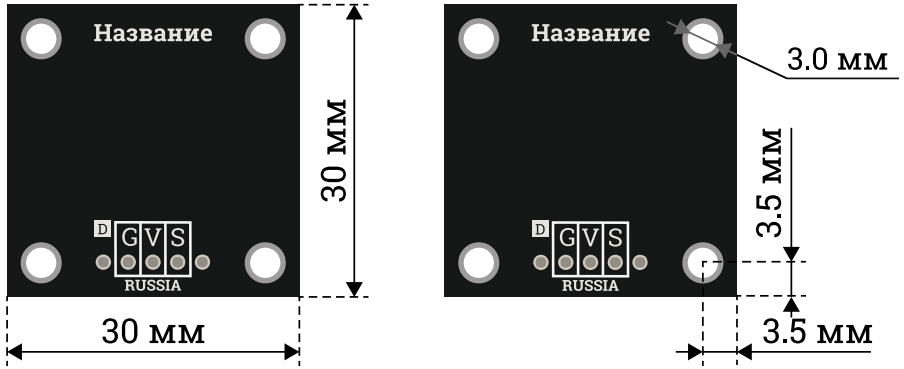
Модуль можно использовать в любых проектах где требуется тактовая кнопка, потенциометр или энкодер, а так же в тех проектах где требуется диммирование устройств без участия других микроконтроллеров.

Спецификация:

- Напряжение питания: 3,3 В или 5 В (постоянного тока).
- Потребляемый ток: до 10 мА (без нагрузки на программируемом выходе).
- 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).

- Уровень логической 1 на выходе Signal: 3,3 В.
- Аналоговый уровень на выходе Signal: от 0 до 3,3 В.
- Разрешение ШИМ: 8 бит.
- Разрешение ЦАП: 8 бит (ШИМ + RC-фильтр).
- Частота ШИМ: устанавливается программно от 1 до 12000 Гц (по умолчанию 1,5 кГц).
- Рабочая температура: от -20 до +70 °C.
- Габариты: 30 x 30 мм.
- Вес: 12 г.

Все модули линейки "Trema" выполнены в одном формате



У модуля имеются две колодки выводов: разъём **I2C** (GND, Vcc, SDA, SCL) и разъём **D/A** (GND, Vcc, Signal).

- **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.
- **Signal** - программируемый выход модуля.

Модуль подключается к Arduino по **шине I2C**.

Устройства которыми модуль управляет в автономном режиме, подключаются к колодке **D/A**.

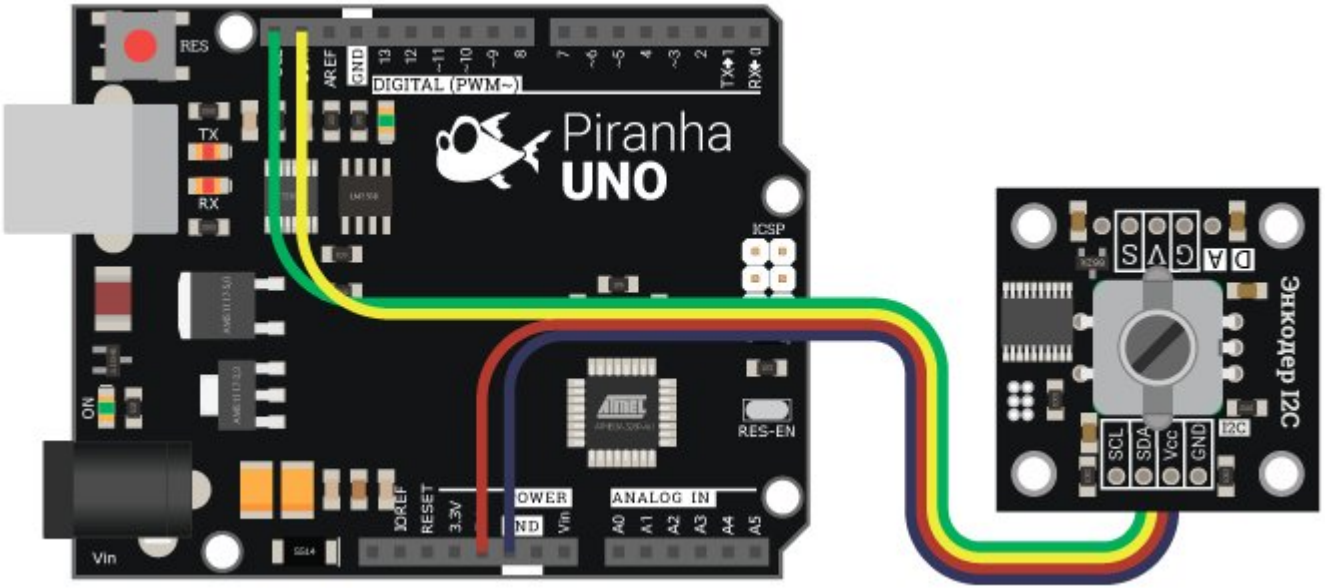
Подключение:

Модуль подключается к аппаратной или программной шине I2C Arduino и имеет адрес 0x68. Для удобства подключения, предлагаем воспользоваться TremaShield.

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

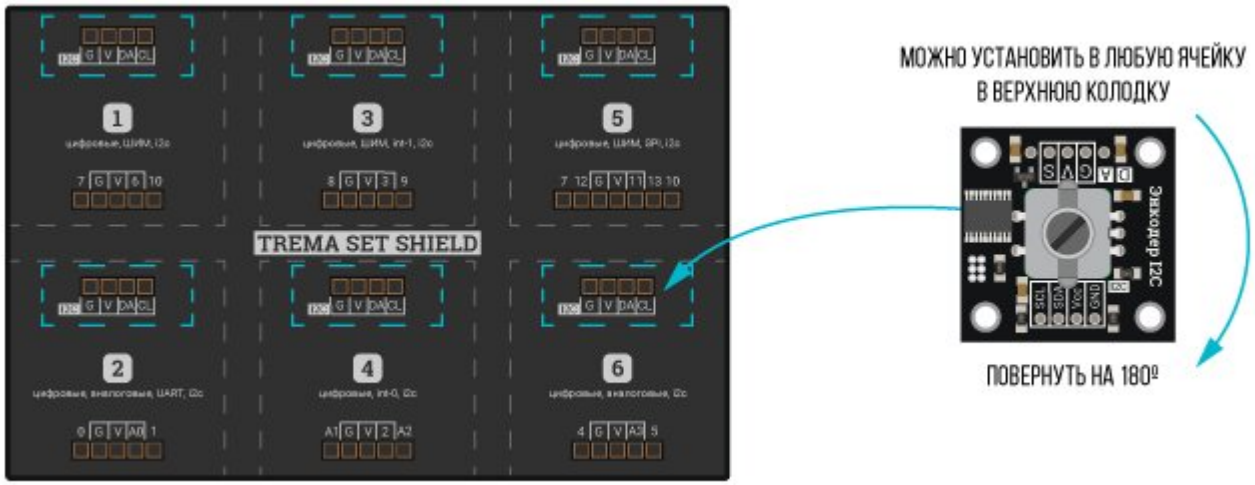
Способ - 1: Используя проводной шлейф и Piranha UNO

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



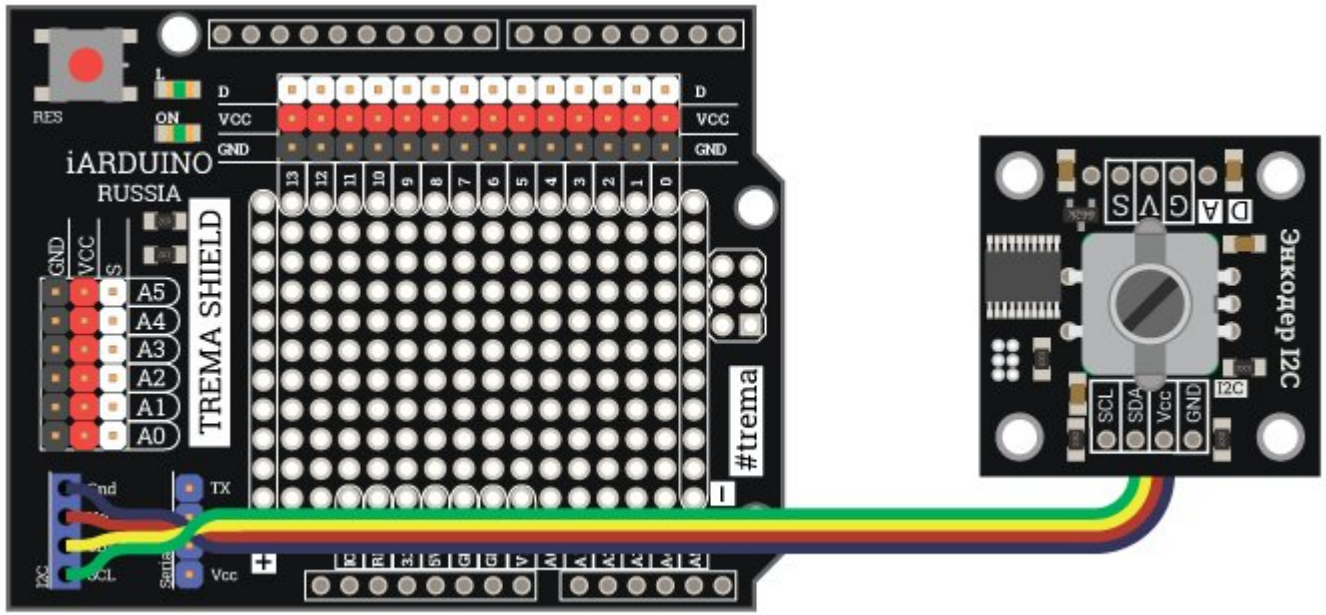
Способ - 2: Используя Trema Set Shield

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.



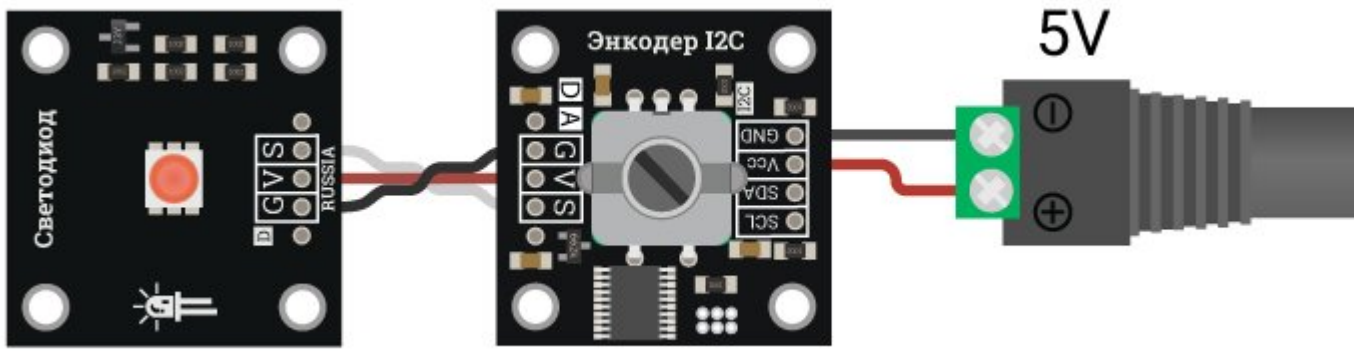
Подключение к колодке D/A:

Модуль может работать в автономном режиме, при этом необязательно подключать ведущее устройство к колодке I2C, достаточно подать питание на Vcc и GND, но, перед этим, модулю необходимо установить опцию работы вывода **S** при помощи ведущего устройства функцией PinMode(РЕЖИМ). Опция будет сохранена в энергонезависимой памяти модуля.

Режимы PinMode:

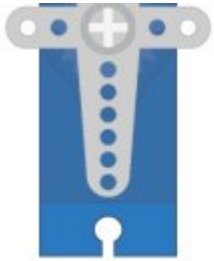
```
ENC_PIN_MODE_KEY, ENC_PIN_MODE_TRG, ENC_PIN_MODE_PWM, ENC_PIN_MODE_PWM_LOG, ENC_PIN_MODE_DAC
```

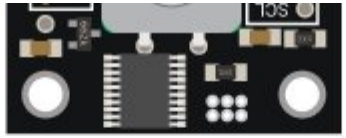
Например, можно подключить модуль светодиод, модуль силовой ключ или драйвер светодиодной ленты.



Режим `PinMode(ENC_PIN_MODE_SER)` **:**

режим `ENC_PIN_MODE_SER` сделан специально для работы с сервоприводами.





Питание:

Входное напряжение питания модуля 3,3В или 5В постоянного тока, подаётся на выводы Vcc и GND любого разъёма.

Вывод «Vcc» колодки «I2C» электрически соединён с выводом «Vcc» колодки «D/A», равно как и вывод «GND» колодки «I2C» соединён с выводом «GND» колодки «D/A». Это позволяет подать питание на колодку «I2C», а питание для устройства которым управляет модуль взять с колодки «D/A» (Digital/Analog).

Подробнее о модуле:

Модуль построен на базе микроконтроллера STM32F030F4 и снабжен собственным стабилизатором напряжения. Модуль самостоятельно обрабатывает сигналы энкодера (и его тактовой кнопки). Отличием данного модуля является наличие вывода, который можно запрограммировать для работы в качестве выхода без дальнейшего подключения модуля к шине I2C (без подключения к внешнему микроконтроллеру). Вывод модуля может работать как выход кнопки энкодера, как выход кнопочного переключателя (триггер), как выход с ШИМ (для управления моторами), как выход с логарифмической ШИМ (для управления светодиодами), как выход управления сервоприводами и как аналоговый выход (имитация ЦАП) с напряжением от 0 до 3,3 В. Режим работы выхода сохраняется и после отключения питания. Значит вывод можно однократно запрограммировать и использовать модуль в проектах без подключения к шине I2C.

Модуль позволяет:

- Получать количество тактов поворота энкодера в одну и другую сторону.
- Получать состояние и события кнопки вала энкодера.
- Получать время удержания кнопки вала энкодера.
- Получать точное положение вала энкодера относительно точки сброса.

- Сбрасывать положение вала энкодера.
- Автономно работать управляя сигналом на выходе без подключения к шине I2C.
- Задавать количество оборотов вала энкодера, как для автономной работы, так и для определения его положения.
- Указывать частоту для сигнала ШИМ, если таковой формируется на выходе модуля.

Специально для работы с Trema модулем - энкодер, потенциометр, I2C-flash , нами разработана библиотека iarduino_I2C_Encoder 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 энкодеру. После этого подключите энкодер 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 I2C.
#include <iarduino_I2C_Encoder.h>                   //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc;                           //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                    //   Если при объявлении объекта указать адрес, например, enc(0xBB), то прим
void setup(){                                       //
    Serial.begin(9600);                             //
    if( enc.begin() ){                              //   Инициируем работу с энкодером.
```

```
        Serial.print("Найден энкодер 0x");         //
        Serial.println( enc.getAddress(), HEX );  //   Выводим текущий адрес модуля.
        if( enc.changeAddress(newAddress) ){      //   Меняем адрес модуля на newAddress.
            Serial.print("Адрес изменён на 0x");   //
            Serial.println(enc.getAddress(),HEX );//   Выводим текущий адрес модуля.
        }else{                                    //
            Serial.println("Адрес не изменён!");  //
        }                                         //
    }else{                                        //
        Serial.println("Энкодер не найден!");     //
    }                                             //
}                                                 //
                                                  //
void loop(){}                                     //
```

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

Примеры работы с модулем по шине I2C:

В данном разделе раскрыты примеры получения данных от модуля по шине I2C с использованием библиотеки iarduino_I2C_Encoder. Сама библиотека содержит больше примеров, доступных из меню Arduino IDE: Файл / Примеры / iarduino I2C Encoder (энкодер - потенциометр).

Чтение разницы совершённых тактов поворота энкодера:

```
#include <Wire.h>                                 //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                 //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                   //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                  //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                     //
    Serial.begin(9600);                           //   Инициируем передачу данных в монитор последовательного порта на скорост
    enc.begin();                                  //   Инициируем работу с энкодером.
```

```
}                                                   //
                                                    //
void loop(){                                        //
    int turn=enc.getEncoder(ENC_TURN);              //   Считываем количество тактов поворота энкодера (как влево, так и вправо)
    if( turn ){ Serial.println(turn); }             //   Выводим количество совершённых тактов (в одном полном обороте 20 тактов
    delay(100);                                     //   Без задержки в мониторе будут появляться только ±1.
}                                                   //
```

После загрузки данного примера, в мониторе последовательного порта будут появляться числа: Определяющие разницу совершённых тактов поворота энкодера.

Один полный оборот энкодера состоит из 20 тактов.

Например: Если с момента появления предыдущего числа в мониторе, повернуть энкодер на 5 тактов против часовой стрелки, то в монитор будет отправлено следующее число равное -5. Если с момента появления предыдущего числа в мониторе, повернуть энкодер на 2 такта против часовой стрелки и 5 тактов по часовой, то в монитор будет отправлено следующее число равное 3.

Чтение количества совершённых тактов поворота энкодера:

```
#include <Wire.h>                                   //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                   //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                     //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                       //
    Serial.begin(9600);                             //   Инициируем передачу данных в монитор последовательного порта на скорост
    enc.begin();                                    //   Инициируем работу с энкодером.
}                                                   //
                                                    //
void loop(){                                        //
    int turnL=0, turnR=0;                           //   Определяем переменные для чтения количества тактов поворота энкодера.
    turnL = enc.getEncoder(ENC_TURN_LEFT);          //   Считываем количество тактов поворота энкодера влево  (против часовой ст
```

```
    turnR = enc.getEncoder(ENC_TURN_RIGHT);       //   Считываем количество тактов поворота энкодера вправо (по     часовой ст
//  Выводим считанные данные:                     //
    if( turnL || turnR ){                         //   Если вал энкодера повернули влево или вправо, то ...
        Serial.print(" -");                       //   Считаем что против часовой стрелки, это отрицательные значения.
        Serial.print(turnL);                      //   Выводим количество совершённых тактов (в одном полном обороте 20 тактов
        Serial.print(" +");                       //   Считаем что по часовой стрелке, это положительные значения.
        Serial.print(turnR);                      //   Выводим количество совершённых тактов (в одном полном обороте 20 тактов
        Serial.print("\r\n");                     //   Переходим на новую строку для следующего вывода данных.
    }                                             //
    delay(100);                                   //   Без задержки в мониторе будут появляться только -1/0 +1/0.
}                                                 //
```

После загрузки данного примера, в мониторе последовательного порта, десять раз в секунду, будут появляться по два числа, определяющие не разницу, а общее количество совершённых тактов поворота энкодера, против часовой стрелки (отрицательные числа), и по часовой стрелке (положительные числа).

Один полный оборот энкодера состоит из 20 тактов.

Например: Если с момента появления предыдущего числа в мониторе, повернуть энкодер на 2 такта против часовой стрелки и 5 тактов по часовой, то в монитор будут отправлены следующие два числа равные -2 и 5.

Чтение текущей позиции вала энкодера:

```
#include <Wire.h>                                 //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                 //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                   //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                  //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                     //
    Serial.begin(9600);                           //   Инициируем передачу данных в монитор последовательного порта на скорост
    enc.begin();                                  //   Инициируем работу с энкодером.
    enc.setPosSettings(5, false);                 //   Указываем считать положение энкодера до 5 полных оборотов, без отрицате
    enc.resPosition();                            //   Обнуляем счётчик положения вала энкодера. Это значит, что текущее полож
```

```
}                                                   //
                                                    //
void loop(){                                        //
    Serial.println( enc.getPosition() );            //   Выводим текущую позицию вала энкодера.
}                                                   //
```

После загрузки данного примера, в мониторе будут появляться числа, точно определяющие текущую позицию вала энкодера.

Функцией `setPosSettings()` было определено считать до 5 полных оборотов, один оборот состоит из 20 тактов, значит в монитор будут отправляться числа от 0 до 100, точно определяющие положение вала энкодера. При вращении энкодера по часовой стрелке, числа будут увеличиваться (но не смогут стать больше 100), а против часовой стрелки, уменьшаться (но не смогут стать меньше 0). Если вызвать функцию `setPosSettings()` указав в качестве второго параметра `true`, то счет будет вестись не от 0 до 100, а от 0 до ±100 (в зависимости от направления поворота вала энкодера).

Данный пример похож на работу с потенциометром, где зная значение можно точно определить положение ручки потенциометра.

Функция `resPosition()` вызванная в коде `Setup()` данного примера, ничего не делает, так как положение вала энкодера при подаче питания и так считается нулевым. Но эту функцию можно вызвать в любой части скетча, где требуется текущее положение принять за `0`.

Реакция на события кнопки вала энкодера:

```
#include <Wire.h>                                   //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                   //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                     //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                       //
    Serial.begin(9600);                             //   Инициируем передачу данных в монитор последовательного порта на скорост
    enc.begin();                                    //   Инициируем работу с энкодером.
}                                                   //
                                                    //
```

```
void loop(){                                    //
    if( enc.getButton(KEY_PUSHED) ){            //   Если кнопка энкодера нажимается, то ...
        Serial.println("Нажали");                //   Выводим текст.
    }                                           //
    if( enc.getButton(KEY_RELEASED) ){          //   Если кнопка энкодера отпускается, то ...
        Serial.println("Отпустили");             //   Выводим текст.
    }                                           //
}                                               //
```

Данный пример демонстрирует работу с событиями кнопки вала энкодера. При нажатии на вал в мониторе однократно появится надпись "Нажали", а при отпускании вала, в мониторе появится надпись "Отпустили", так же однократно.

Реакция на состояние удержания кнопки энкодера:

```
#include <Wire.h>                               //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>               //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                 //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                   //
    Serial.begin(9600);                         //   Инициируем передачу данных в монитор последовательного порта на скорост
    enc.begin();                                //   Инициируем работу с энкодером.
}                                               //
                                                //
void loop(){                                    //
    if( enc.getButton(KEY_TIME_PRESSED) > 5000 ){ //   Если время удержания кнопки возвращаемое функцией getButton больше 5000
        Serial.println("Удерживается");          //   Выводим текст.
    }                                           //
}                                               //   Максимально фиксируемое время удержания: 25500 миллисекунд = 25,5 секун
```

Данный пример демонстрирует работу с одним из состояний кнопки вала энкодера. Если нажать на вал и удерживать его в нажатом

состоянии, то через 5 секунд в мониторе последовательного порта начнут появляться надписи "Удерживается", пока не отпустить вал.

Инверсия поворота вала энкодера:

```
#include <Wire.h>                                   //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                   //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                     //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                       //
    enc.begin();                                    //   Инициируем работу с энкодером.
    enc.invEncoder( true );                         //   Установить флаг инверсии поворота.
}                                                   //
                                                    //
void loop(){}                                       //
```

После загрузки этого скетча, модуль станет воспринимать поворот вала энкодера наоборот. Поворот энкодера влево будет восприниматься как поворот энкодера в право, а поворот вправо как поворот влево. Флаг инверсии направления поворота заданный функцией `invEncoder()` сохраняется в энергонезависимой памяти модуля. Это значит что для восстановления правильной работы энкодера не достаточно просто отключить питание, а требуется загрузить скетч в котором выполняется однократное обращение к функции `invEncoder()` с параметром `false` .

Инверсия поворота вала энкодера влияет не только на те параметры которые можно прочитать по шине I2C, но и на поведение выхода модуля в автономном режиме работы.

Остальные примеры работы с модулем по шине I2C, доступны из меню Arduino IDE: Файл / Примеры / iarduino I2C Encoder (энкодер - потенциометр).

Примеры автономной работы модуля:

В данном разделе раскрывается возможность программирования выхода модуля. После того как выход запрограммирован на требуемый режим работы, модуль может работать автономно (без подключения к шине I2C). Выбранный режим работы выхода сохраняется в

энергонезависимую память модуля, значит вывод продолжит работать в указанном режиме даже после отключения и подачи питания на модуль.

Настройка вывода модуля на работу выхода с ШИМ:

```
#include <Wire.h>                                   //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                   //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                     //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                       //
    enc.begin();                                    //   Инициируем работу с энкодером.
    enc.setPinOut(ENC_PIN_MODE_PWM, 2, 5000);       //   Указываем выводу модуля работать как выход с ШИМ на частоте 5 кГц,
}                                                   //   коэффициент заполнения которого меняется от 0% до 100%
                                                    //   за 2 полных оборота вала энкодера.
void loop(){}                                       //
```

В данном примере вывод модуля программируется в режим работы выхода с ШИМ, работающем на частоте 5 кГц, а его коэффициент заполнения меняется от 0 до 100%, за 2 полных оборота вала энкодера.

Режим работы вывода задаётся функцией `setPinOut()`. Первый параметр функции задаёт режим работы, второй указывает количество полных оборотов за которое коэффициент заполнения ШИМ на выходе модуля изменится с 0 до 100%, а третий параметр указывает частоту ШИМ на выходе модуля от 1 до 12000 Гц. Обязательным является только первый параметр.

Все параметры указанные в функции `setPinOut()`, сохраняются в энергонезависимую память энкодера, значения отсутствующих параметров берутся из памяти.

Теперь если подать питание на модуль (даже не подключая его к шине I2C), на выходе модуля появится сигнал ШИМ, коэффициент заполнения которого будет увеличиваться при повороте вала энкодера по часовой стрелке, и уменьшаться при повороте против часовой стрелки. Вывод будет работать даже после отключения и подачи питания.

В данном примере используется ШИМ, коэффициент заполнения которого имеет линейную зависимость от поворота вала энкодера. Если

Вы желаете использовать выход модуля для управления яркостью осветительных приборов или громкостью звуковых, то в качестве первого аргумента функции `setPinOut()` лучше указать `ENC_PIN_MODE_PWM_LOG`, этот режим ШИМ отличается логарифмической зависимостью от поворота вала энкодера.

Настройка вывода модуля на работу в качестве аналогового выхода:

```
#include <Wire.h>                                   //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                   //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                     //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                       //
    enc.begin();                                    //   Инициируем работу с энкодером.
    enc.setPinOut(ENC_PIN_MODE_DAC, 3);             //   Указываем выводу модуля работать как аналоговый выход,
}                                                   //   напряжение которого меняется от 0 до 3.3В
                                                    //   за 3 полных оборота вала энкодера.
void loop(){}                                       //
```

Данный пример похож на предыдущий, но выход модуля программируется не на работу в режиме выхода с ШИМ, а на работу в качестве аналогового выхода, напряжение которого меняется с минимального (0В) до максимального (3,3В) за 3 полных оборота вала энкодера.

В данном примере, функцию `setPinOut()` по прежнему можно вызвать с тремя параметрами (указав частоту ШИМ), так как аналоговый сигнал получается прохождением сигнала ШИМ через сглаживающий RC-фильтр модуля. Если частота ШИМ не указана (как в примере), то будет использована частота сохранённая ранее в энергонезависимую память энкодера.

Настройка вывода модуля на работу с сервоприводами:

```
#include <Wire.h>                                   //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                   //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                     //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                       //
```

```
    enc.begin();                                    //  Инициируем работу с энкодером.
//  enc.setServoLimit(450, 2450);                   //  Задаём границы поворота сервопривода, указав ширину импульсов в мкс для
    enc.setPinOut(ENC_PIN_MODE_SER, 5);             //  Указываем выводу модуля работать как выход управления сервоприводом.
}                                                   //  Задав 5 полных оборотов вала энкодера для полного хода вала сервопривод
                                                    //
void loop(){}                                       //
```

Данный пример программирует вывод модуля на работу с сервоприводом, угол поворота которого меняется от минимального до максимального за 5 полных оборотов вала энкодера.

В данном примере, функции `setPinOut()` нельзя указать частоту ШИМ, так как управление сервоприводами осуществляется на частоте 50 Гц, эта частота не сохраняется в память модуля.

Функцией `setServoLimit()` (если её раскомментировать) можно определить границы поворота вала сервопривода, задав минимальную и максимальную ширину импульсов в мкс. Эти значения сохранятся в энергонезависимую память модуля. Для большинства сервоприводов полный ход вала определён границами от 450 до 2450 мкс.

Установка ограничений поворота вала сервопривода:

Из предыдущего примера стало ясно что функцией `setServoLimit()` можно ограничить угол вращения вала сервопривода, но данная функция принимает не углы в градусах, а ширину импульсов в микросекундах, так как разные сервоприводы рассчитаны на разные углы.

Следующий пример позволяет ограничить диапазон вращения вала сервопривода, даже если Вам не известны ни значения углов, ни ширина импульсов им соответствующая.

```
#include <Wire.h>                                   //  Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>                   //  Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                     //  Объявляем объект enc для работы с функциями и методами библиотеки iardu
const char* txt1 =                                      "Поворотом ручки энкодера добейтесь требуемого\r\nкрайнего положения ва
const char* txt2 =                                      "Поворотом ручки энкодера добейтесь другого\r\nкрайнего положения вала
uint16_t    i=450, j=2450;                          //  Определяем ширину импульсов для крайних положений вала сервопривода в м
```

```
                                                //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                   //
    Serial.begin(9600);                         //   Инициируем передачу данных в монитор последовательного порта на скорост
    enc.begin();                                //   Инициируем работу с энкодером.
    enc.setPinOut(ENC_PIN_MODE_SER, 2);         //   Указываем выводу модуля работать с сервоприводом, отведя под управление
    enc.setServoLimit(i, j);                    //   Задаём границы поворота сервопривода, указав ширину импульсов в мкс для
    Serial.println(txt1);                       //   Выводим текст информирующий о действиях которые необходимо выполнить.
    while(enc.getButton(KEY_PUSHED)==false){;}  //   Бесконечно ждём нажатия кнопки энкодера ...
    i  =  enc.getServoWidth();                  //   Получаем ширину импульсов текущего положения вала сервопривода.
    Serial.println("Ок."  );                    //
    while(enc.getButton(KEY_RELEASED)==false){;} //   Бесконечно ждём отпускания кнопки энкодера ...
    Serial.println("   ");                      //
    Serial.println(txt2);                       //   Выводим текст информирующий о действиях которые необходимо выполнить.
    while(enc.getButton(KEY_PUSHED)==false){;}  //   Бесконечно ждём нажатия кнопки энкодера ...
    j  =  enc.getServoWidth();                  //   Получаем ширину импульсов текущего положения вала сервопривода.
    Serial.println("Ок."  );                    //
    while(enc.getButton(KEY_RELEASED)==false){;} //   Бесконечно ждём отпускания кнопки энкодера ...
    enc.setServoLimit(i, j);                    //   Задаём новые границы поворота сервопривода, указав ширину импульсов в м
    Serial.println("   ");                      //
    Serial.println("Калибровка завершена!");    //
}                                               //
                                                //
void loop(){}                                   //
```

Данный скетч можно выполнить однократно, так как значения ширины импульсов (в мкс.) указываемые функцией `setServoLimit()` и режим работы выхода задаваемый функцией `setPinOut()` сохраняются в энергонезависимой памяти модуля.

Настройка вывода модуля на работу в качестве выхода кнопки:

```
#include <Wire.h>                               //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>               //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                 //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
```

```
                                                //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                   //
    enc.begin();                                //   Инициируем работу с энкодером.
    enc.setPinOut(ENC_PIN_MODE_KEY);            //   Указываем выводу модуля работать как выход кнопки энкодера.
}                                               //
                                                //
void loop(){}                                   //
```

Данный пример программирует вывод модуля на работу в качестве выхода кнопки. При нажатии на вал энкодера на выходе установится уровень логической 1, а при отпускании вала энкодера, уровень логического 0.

Отключение вывода модуля:

```
#include <Wire.h>                               //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Encoder.h>               //   Подключаем библиотеку для работы с энкодером I2C-flash.
iarduino_I2C_Encoder enc(0x09);                 //   Объявляем объект enc для работы с функциями и методами библиотеки iardu
                                                //   Если объявить объект без указания адреса (iarduino_I2C_Encoder enc;), т
void setup(){                                   //
    enc.begin();                                //   Инициируем работу с энкодером.
    enc.setPinOut(ENC_PIN_MODE_OFF);            //   Указываем выводу модуля не работать.
}                                               //   Вывод модуля перейдёт в состояние высокого импеданса.
                                                //
void loop(){}                                   //
```

Данный пример программирует выход на отключение. Выход переходит в режим высокого импеданса и на него не подаются сигналы модуля.

Остальные примеры работы выхода модуля доступны из меню Arduino IDE: Файл / Примеры / iarduino I2C Encoder (энкодер - потенциометр).

Описание функций библиотеки:

В данном разделе описаны функции библиотеки iarduino_I2C_Encoder для работы с Trema модулем - Энкодер, потенциометр, 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_Encoder.h>   // Подключаем библиотеку для работы с модулем.
iarduino_I2C_Encoder enc(0x09);     // Создаём объект enc для работы с функциями и методами библиотеки iarduino_I2C_Encoder, у
```

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

```
#include <iarduino_I2C_Encoder.h>   // Подключаем библиотеку для работы с модулем.
iarduino_I2C_Encoder enc;           // Создаём объект enc для работы с функциями и методами библиотеки iarduino_I2C_Encoder.
```

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( enc.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( enc.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( enc.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( enc.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( enc.getVersion(), HEX );
```

Функция getButton();

Назначение: Запрос времени удержания, состояния, или события кнопки энкодера.

Синтаксис: getButton(ЗАПРОС);

Параметр: uint8_t - один из перечисленных вариантов...

- KEY_PUSHED - вернуть ФЛАГ события кнопки - «нажимается».
- KEY_RELEASED - вернуть ФЛАГ события кнопки - «отпускается».
- KEY_CHANGED - вернуть ФЛАГ события кнопки - «состояние изменилось».
- KEY_PRESSED - вернуть ФЛАГ состояния кнопки - «нажата».
- KEY_TRIGGER - вернуть ФЛАГ состояния кнопки - «переключатель».

- KEY_HOLD_05 - вернуть ФЛАГ состояния кнопки - «удерживается» дольше 0,5 сек.
- KEY_HOLD_10 - вернуть ФЛАГ состояния кнопки - «удерживается» дольше 1,0 сек.
- KEY_HOLD_20 - вернуть ФЛАГ состояния кнопки - «удерживается» дольше 2,0 сек.
- KEY_TIME_PRESSED - вернуть ВРЕМЯ удержания кнопки в миллисекундах.

Возвращаемое значение: зависит от параметра функции:

- ФЛАГ наличия запрашиваемого состояния или события (true или false).
- ВРЕМЯ удержания кнопки в миллисекундах (от 0 до 25500).

Примечание:

- Если функция вызвана с параметром `KEY_TIME_PRESSED`, то она вернёт ВРЕМЯ удержания кнопки, или `0` (если кнопка отпущена). При указании любого другого параметра, функция вернёт ФЛАГ соответствующий запрашиваемому состоянию или событию.
- Если запрошено событие кнопки (нажимается, отпускается, состояние изменилось), то функция вернёт `true` только один раз после совершения запрашиваемого события.
- Если запрошено состояние кнопки (нажата, переключатель, удерживается), то функция будет возвращать `true` всё время, пока установлено запрашиваемое состояние.

Пример:

```
bool   i = enc.getButton(KEY_CHANGED);       // Переменная i будет установлена в true, если кнопка нажимается или отпускается.
uint16 j = enc.getButton(KEY_TIME_PRESSED ); // Переменная j содержит время удержания кнопки в миллисекундах.
```

Функция getEncoder();

Назначение: Запрос количества совершённых тактов оборота энкодера.

Синтаксис: getEncoder([НАПРАВЛЕНИЕ]);

Параметр: uint8_t - один из перечисленных вариантов...

- ENC_TURN_LEFT - вернуть количество тактов поворота энкодера влево (против ч.с.).
- ENC_TURN_RIGHT - вернуть количество тактов поворота энкодера вправо (по ч.с.).
- ENC_TURN - вернуть разницу межу тактами поворота энкодера влево и вправо.

Возвращаемое значение: int16_t КОЛИЧЕСТВО тактов поворота энкодера.

Примечание:

- Если функция вызвана без параметра, то применяется параметр по умолчанию `ENC_TURN` .
- Один полный оборот вала энкодера состоит из 20 тактов.
- Если функция вызвана с параметром `ENC_TURN_LEFT` или `ENC_TURN_RIGHT` , то она вернёт количество тактов поворота энкодера (от 0 до 255) совершённых после последнего обращения к функции с тем же параметром, или параметром `ENC_TURN` .
- Если с момента последнего запроса совершено более 255 тактов, то функция вернёт 255.
- Если функция вызвана с параметром `ENC_TURN` , то она вернёт разницу тактов поворота энкодера (от -255 до 255) совершённых после последнего обращения к функции.

Пример:

```
// Предположим, совершено 2 такта влево и 3 такта вправо:
int16_t i = enc.getEncoder(ENC_TURN_LEFT ); // Переменная i содержит значение 2.
// Предположим, совершено еще 2 такта влево и 3 такта вправо:
int16_t j = enc.getEncoder(ENC_TURN_LEFT ); // Переменная j содержит значение 2.
int16_t k = enc.getEncoder(ENC_TURN_RIGHT); // Переменная k содержит значение 6.
```

Функция getPosition();

- Назначение: Запрос текущей позиции вала энкодера.
- Синтаксис: getPosition();
- Параметр: Нет.
- Возвращаемое значение: int16_t ПОЗИЦИЯ - значение от 0 до ±1260.
- Примечание:

    - Функция возвращает позицию поворота вала энкодера в тактах от нулевой точки (от положения в котором вал находился при подаче питания, или при обращении к функции сброса текущей позиции `resPosition()`), до максимального количества тактов заданных функцией `setPosSettings()`.
    - Один полный оборот вала энкодера состоит из 20 тактов.
- Пример:

```
// Предположим, совершено 2 такта влево (против ч.с.) и 3 такта вправо (по ч.с.):
int16_t i = enc.getPosition(); // Переменная i содержит значение 1.
// Предположим, совершено еще 2 такта влево и 3 такта вправо:
int16_t j = enc.getPosition(); // Переменная j содержит значение 2.
```

Функция resPosition();

- Назначение: Сброс текущей позиции вала энкодера.
- Синтаксис: resPosition();
- Параметры: Нет
- Возвращаемое значение: bool - результат сброса позиции (true или false).
- Примечание:
    - Функция `getPosition()` будет вести счет текущей позиции вала энкодера от того положения в котором он находился при обращении к функции `resPosition()`.
- Пример:

```
// Предположим, совершено 2 такта влево (против ч.с.):
int16_t i = enc.getPosition(); // Переменная i содержит значение -2.
enc.resPosition();             // Принимаем текущее положение вала энкодера за позицию 0.
// Предположим, совершено 2 такта вправо (по ч.с.):
int16_t j = enc.getPosition(); // Переменная j содержит значение 2.
```

Функция setPosSettings();

- Назначение: Настройка получения позиции вала энкодера.

- Синтаксис: setPosSettings(ОБОРОТЫ [, ЗНАК]);
- Параметры:
  - uint8_t ОБОРОТЫ - максимальное количество полных оборотов вала энкодера, от 1 до 63.
  - bool ЗНАК - флаг разрешающий отрицательные значения положения вала энкодера.
- Возвращаемое значение: bool - результат сохранения настроек (true или false).
- Примечание:
  - Указанное количество оборотов определяет границу положения вала энкодера возвращаемую функцией `getPosition()`. В одном обороте 20 тактов. Если указать 3 оборота, то функция `getPosition()` будет выводить числа от 0 до 60 (достигнув значения 0 оно перестанет уменьшаться, а достигнув значения 60, оно перестанет приращаться).
  - Если в качестве второго параметра указать `true`, то функция `getPosition()` будет возвращать числа от 0 до ± количества тактов в заданном количестве оборотов.
  - Если в качестве второго аргумента указать `false` или не указывать его вообще, то минимальным значением возвращаемым функцией `getPosition()` будет `0`.
- Пример:

```
enc.setPosSettings( 5, true);  // Указываем считать до 5 оборотов от 0 со знаком.
int16_t i = enc.getPosition(); // Переменная i содержит значение от -5*20 до +5*20.
```

Функция setPinOut();

Назначение: Настройка работы выхода модуля.

Синтаксис: setPinOut(РЕЖИМ [, ОБОРОТЫ [, ЧАСТОТА]]);

Параметры:

uint8_t РЕЖИМ - режим работы выхода модуля:

- ENC_PIN_MODE_OFF - Выход модуля отключён.
- ENC_PIN_MODE_KEY - Вывод модуля работает как выход кнопки вала энкодера.
- ENC_PIN_MODE_TRG - Выход модуля работает как кнопочный переключатель (триггер).

- ENC_PIN_MODE_PWM - Выход модуля работает в режиме ШИМ.
- ENC_PIN_MODE_PWM_LOG - Выход модуля выводит логарифмический ШИМ.
- ENC_PIN_MODE_DAC - Выход работает в аналоговом режиме (ЦАП).
- ENC_PIN_MODE_SER - Выход модуля работает в режиме управления сервоприводами.

uint8_t ОБОРОТЫ - максимальное количество полных оборотов вала энкодера, от 1 до 15.

uint16_t ЧАСТОТА - частота работы ШИМ, от 1 до 12'000 Гц.

Возвращаемое значение: bool - результат сохранения настроек (true или false).

Примечание:

- Все настройки вывода сохраняются в энергонезависимую память модуля. Это значит, что вывод можно однократно запрограммировать и использовать модуль в проектах без подключения к шине I2C.
- Указанное количество оборотов определяет, за сколько полных оборотов вала энкодера, значение ШИМ или АЦП, на выходе модуля, изменится от 0 до 100%.
- Если вывод работает в режиме ШИМ или АЦП, то можно указать частоту ШИМ в Гц, от 1 до 12'000.
- Если вывод работает в режиме управления сервоприводом, то частоту ШИМ указать нельзя, так как управление сервоприводами осуществляется только на частоте 50 Гц.
- Аналоговый режим работы выхода осуществляется подачей ШИМ через сглаживающий RC-фильтр, что позволяет получить на выходе модуля напряжение регулируемое от 0 до 3,3 В.
- Режим логарифмического ШИМ лучше подходит для управления яркостью осветительных приборов (например, светодиодов), чем обычный (линейный) ШИМ.

Пример:

```
enc.setPinOut(ENC_PIN_MODE_KEY);   // Вывод работает в режиме кнопки. На выходе будет 1 пока нажат вал энкодера.
enc.setPinOut(ENC_PIN_MODE_TRG);   // Вывод работает в режиме переключателя. Логический уровень на выходе модуля будет менятьс
enc.setPinOut(ENC_PIN_MODE_DAC,2); // Вывод работает в аналоговом режиме. Напряжение на выходе модуля изменится от 0 до 3,3В з
enc.setPinOut(ENC_PIN_MODE_SER,3); // Вывод работает в режиме управления сервоприводом. Вал сервопривода поворачивается от мин
```

Функция invEncoder();

Назначение: Инверсия направления поворота вала энкодера.

Синтаксис: invEncoder(ФЛАГ);

Параметры: bool - ФЛАГ инверсии направления поворота вала энкодера.

Возвращаемое значение: bool - результат сохранения настроек (true или false).

Примечание:

- Установка флага приводит к инверсии направления поворота вала энкодера. Поворот влево будет расцениваться как поворот вправо, а поворот вправо как поворот влево.
- Флаг инверсии сохраняется в энергонезависимую память модуля. Это значит, что флаг можно задать однократно, а его значение сохранится и после отключения питания.
- Флаг инверсии направления поворота влияет не только на те данные которые можно прочитать по шине I2C, но и на поведение модуля в режиме управления выходом.

Пример:

```
enc.invEncoder( true  ); // Указываем расценивать поворот вала энкодера влево как поворот вправо, а поворот вправо как поворот
enc.invEncoder( false ); // Указываем расценивать поворот вала энкодера как есть, без инверсии, влево значит влево, вправо зна
```

Функция setServoLimit();

Назначение: Ограничение углов сервопривода при управлении им в автономном режиме.

Синтаксис: setServoLimit(ШИРИНА, ШИРИНА);

Параметры:

- uint16_t ШИРИНА - ширина импульсов ШИМ в мкс., соответствующая требуемому углу.

Возвращаемое значение: bool - результат сохранения настроек (true или false).

Примечание:

- Для управления сервоприводом необходимо настроить вывод энкодера функцией `setPinOut()` с параметром `ENC_PIN_MODE_SER` .
- Управление сервоприводом осуществляется подачей на его вход сигнала ШИМ с частотой 50 Гц. Угол сервопривода устанавливается в зависимости от ширины импульсов ШИМ. Для большинства сервоприводов минимальному углу соответствует ширина импульсов в районе 450-550 мкс, а максимальному углу ширина 2400-2500 мкс.
- Функцией `getServoWidth()` можно узнать какому углу соответствует какая ширина импульсов.
- Параметры для минимального и максимального угла можно указывать в любом порядке.
- Указанные функцией ограничения сохраняются в энергонезависимую память модуля. Это значит, что фал сервопривода будет поворачиваться в указанном диапазоне даже после отключения и подачи питания.

Пример:

```
enc.setServoLimit( 450, 2450 ); // Снимаем ограничения хода вала сервопривода.
enc.setServoLimit( 1000, 2000 ); // Ограничиваем ход вала сервопривода импульсами от 1000 мкс до 2000 мкс.
enc.setServoLimit( 2000, 1000 ); // Эта строка делает то же самое что и предыдущая.
```

Функция getServoWidth();

Назначение: Запрос ширины импульсов текущего положения вала сервопривода.

Синтаксис: getServoWidth();

Параметры: Нет.

Возвращаемое значение: uint16_t ШИРИНА - текущая ширина импульсов ШИМ в мкс.

Примечание:

- Получить ширину импульсов ШИМ управления сервоприводом можно только если выход модуля настроен на режим управления сервоприводом. Настройка вывода осуществляется обращением к функции `setPinOut()` с параметром `ENC_PIN_MODE_SER`.
- Функция получения ширины импульсов может быть полезна для ограничения хода вала сервопривода функцией `setServoLimit()`.

Пример:

```
uint16_t i = enc.getServoWidth(); // Получаем ширину импульсов шим соответствующую текущему положению вала сервопривода.
```

Ссылки:

- Wiki - Энкодер, потенциометр, I2C-flash - Datasheet.
- Библиотека iarduino_I2C_Encoder.
- Расширенные возможности библиотек iarduino для шины I2C.
- Wiki - Установка библиотек в Arduino IDE.

Обсуждение