Джойстик, FLASH-I2C

Общие сведения:

Trema модуль - Джойстик, I2C-flash - является устройством ввода данных.

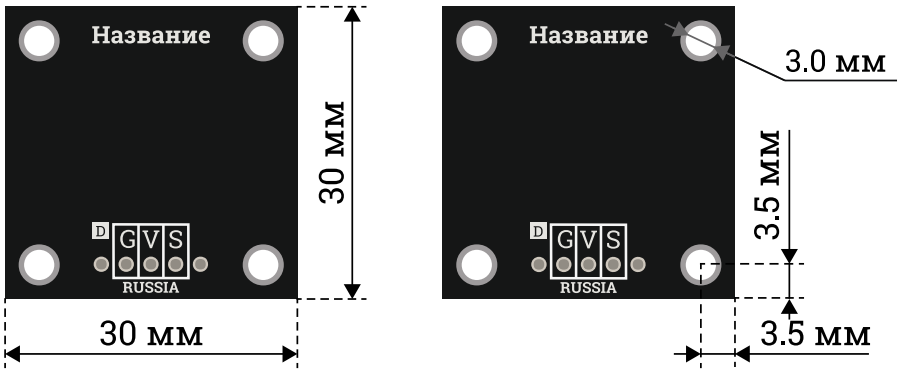
Модуль относится к серии «Flash», а значит к одной шине I2C можно подключить более 100 модулей, так как их адрес на шине I2C (по умолчанию 0x09), хранящийся в энергонезависимой памяти, можно менять программно.

Модуль можно использовать для управления роботами, движущимися механизмами, станками с ЧПУ, для создания игр и многих других проектов.

Спецификация:

- Напряжение питания: 3,3 В или 5 В (постоянного тока).
- Потребляемый ток: до 15 мА.
- Интерфейс: I2C.
- Скорость шины I2C: 100 кбит/с.
- Адрес на шине I2C: устанавливается программно (по умолчанию 0x09).
- Уровень логической 1 на линиях шины I2C: равно напряжению питания модуля (3,3 В или 5 В) .
- Разрешение АЦП: 12 бит.
- Диапазон положений джойстика по осям X и Y: -100 ... 0 ... +100.
- Рабочая температура: от -20 до +70 °C.
- Габариты: 30 x 30 мм.
- Вес: 6 г.

Все модули линейки "Trema" выполнены в одном формате



Подключение:

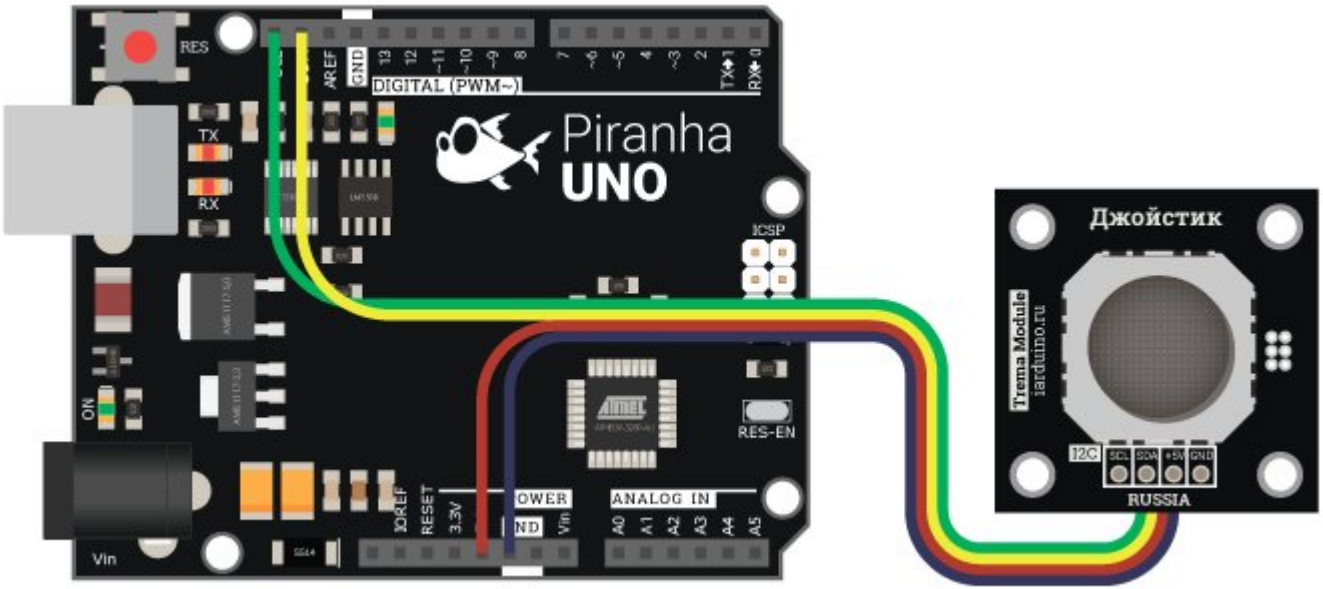
Перед подключением модуля ознакомьтесь с разделом "Смена адреса модуля на шине I2C" в данной статье.

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.

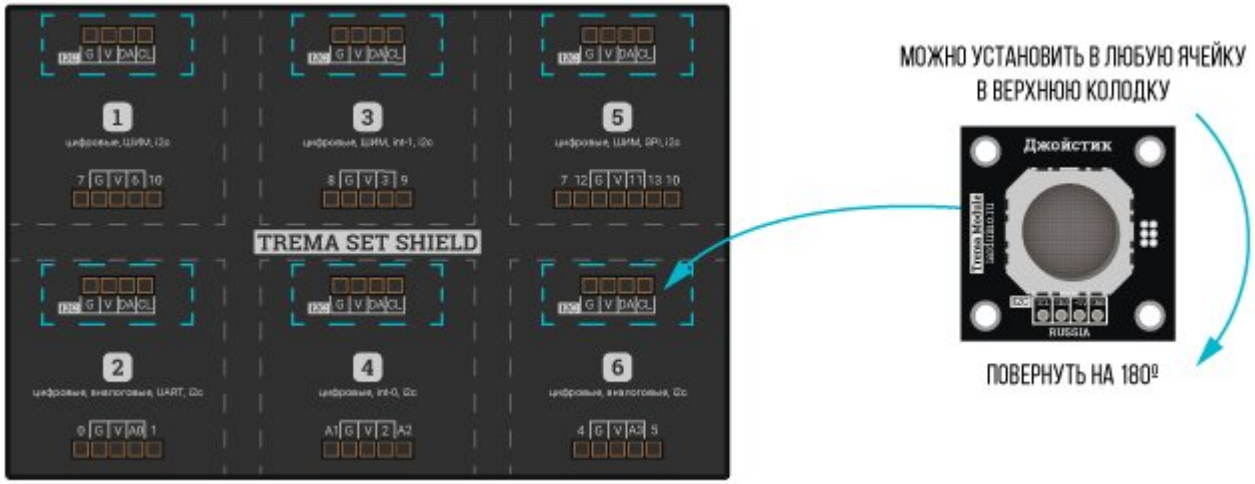
Способ - 1: Используя проводной шлейф и Piranha UNO

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



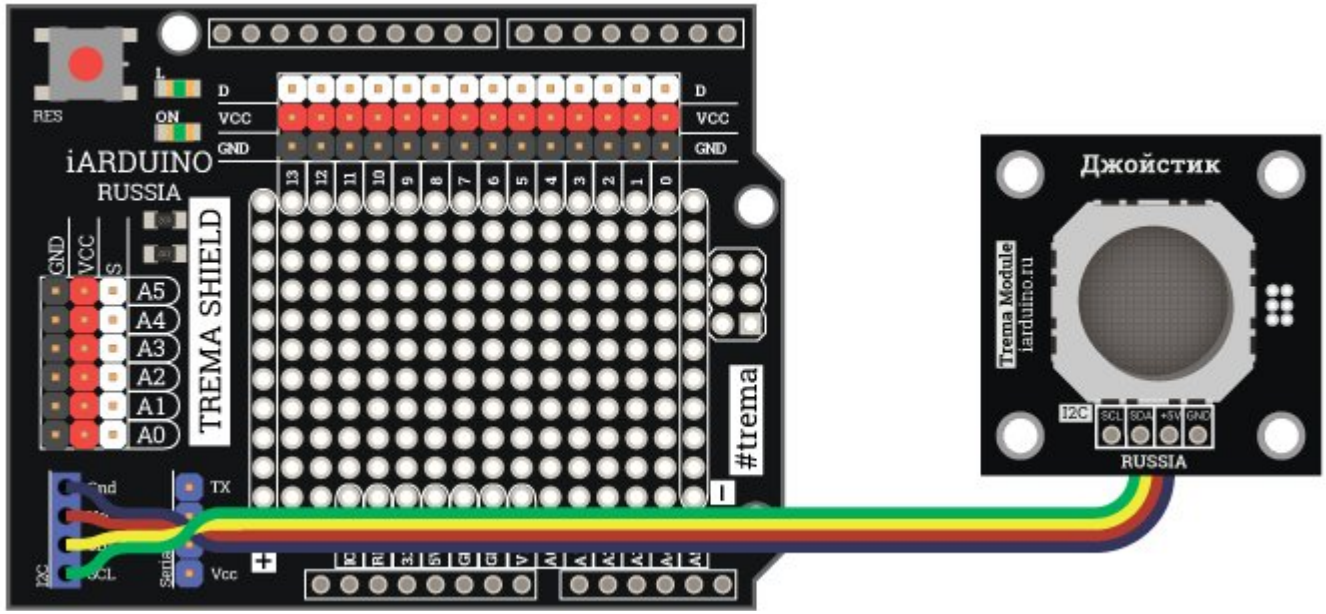
Способ - 2: Используя Trema Set Shield

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. Модуль самостоятельно считывает состояние джойстика, обрабатывает полученные данные и по запросу выводит точные координаты.

Модуль позволяет:

- Менять свой адрес на шине I2C.
- Получать текущее положение джойстика по осям X и Y.
- Получить необработанные значения АЦП (12-бит) считанные с потенциометров джойстика.
- Откалибровать джойстик.
- Задать чувствительность джойстика.
- Задать величину мёртвой зоны у центрального положения.
- Для модуля с кнопкой, можно определить события и состояния кнопки.

Специально для работы с Trema модулем - Джойстик, I2C-flash , нами разработана библиотека iarduino_I2C_Joystick 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 I2C.
#include <iarduino_I2C_Joystick.h>                  //   Подключаем библиотеку для работы с джойстиком I2C-flash.
iarduino_I2C_Joystick joy;                          //   Объявляем объект joy для работы с функциями и методами библиотеки iardu
                                                    //   Если при объявлении объекта указать адрес, например, joy(0xBB), то прим
void setup(){                                       //
    Serial.begin(9600);                             //
    if( joy.begin() ){                              //   Инициируем работу с джойстиком.
        Serial.print("Найден модуль с адр. 0x");     //
        Serial.println( joy.getAddress(), HEX );    //   Выводим текущий адрес модуля.
        if( joy.changeAddress(newAddress) ){        //   Меняем адрес модуля на newAddress.
            Serial.print("Адрес изменён на 0x");    //
            Serial.println(joy.getAddress(),HEX );//   Выводим текущий адрес модуля.
        }else{                                      //
            Serial.println("Адрес не изменён!");    //
        }                                           //
    }else{                                          //
        Serial.println("Модуль не найден!");        //
    }                                               //
}                                                   //
                                                    //
void loop(){}                                       //
```

Примеры:

В данном разделе раскрыты примеры получения данных от модуля по шине I2C с использованием библиотеки iarduino_I2C_Joystick. Сама

библиотека содержит больше примеров, доступных из меню Arduino IDE: Файл / Примеры / iarduino I2C Joystick (джойстик).

Получение координат джойстика:

```
#include <Wire.h>                                   //   Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Joystick.h>                  //   Подключаем библиотеку для работы с джойстиком I2C-flash.
iarduino_I2C_Joystick joy(0x09);                    //   Объявляем объект joy для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Joystick joy;),
void setup(){                                       //
    delay(500);                                     // * Ждём завершение переходных процессов связанных с подачей питания.
    Serial.begin(9600);                             //
    while(!Serial){;}                               // * Ждём завершения инициализации шины UART.
    joy.begin();                                    //   Инициируем работу с джойстиком.
}                                                   //
                                                    //
void loop(){                                        //
    Serial.print( "X=" );                           //
    Serial.print( joy.getPosition_X() );            //   Выводим координату по оси X, значение от -100 до 100.
    Serial.print( ", Y=" );                         //
    Serial.print( joy.getPosition_Y() );            //   Выводим координату по оси Y, значение от -100 до 100.
    Serial.print( "\r\n" );                        //
}                                                   //
```

После загрузки данного примера, в мониторе последовательного порта будут появляться строки с текущими координатами джойстика.

Координаты можно получать как по отдельности: `getPosition_X()`, `getPosition_Y()`, так и одним запросом используя функцию `getPosition()`.

Получение направления джойстика в градусах:

```
#include <Wire.h>                                   // * Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Joystick.h>                  //   Подключаем библиотеку для работы с джойстиком I2C-flash.
```

```
iarduino_I2C_Joystick joy(0x09);                    //   Объявляем объект joy для работы с функциями и методами библиотеки iardu
                                                    //   Если объявить объект без указания адреса (iarduino_I2C_Joystick joy;),
int x,y,a;                                          //   Объявляем переменные для получения координат «x»,«y» и рассчёта угла см
                                                    //
void setup(){                                       //
    delay(500);                                     // * Ждём завершение переходных процессов связанных с подачей питания.
    Serial.begin(9600);                             //
    while(!Serial){;}                               // * Ждём завершения инициализации шины UART.
    joy.begin();                                    //   Инициируем работу с джойстиком.
}                                                   //
                                                    //
void loop(){                                        //
    joy.getPosition(x,y);                           //   Получаем координаты джойстика в переменные x и y.
    a = atan((float)abs(y)/(float)abs(x))*57.3;     //   Вычисляем угол смещения джойстика от   0° до  90°.
    if( x<0  && y>=0 ){a=180-a;}                    //   Вычисляем угол смещения джойстика от  90° до 180°.
    if( x<0  && y<0  ){a=180+a;}                    //   Вычисляем угол смещения джойстика от 180° до 270°.
    if( x>=0 && y<0  ){a=360-a;}                    //   Вычисляем угол смещения джойстика от 270° до 360°.
    if(  x || y  ){                                 //   Если джойстик отклонён от центра, то ...
        Serial.print("Угол = ");                    //
        Serial.print(a);                            //
        Serial.print( "°\r\n" );                    //
    }                                               //
}                                                   //
```

После загрузки данного примера, в мониторе последовательного порта будут появляться строки с указанием угла поворота джойстика, но только если он отклонён от центра.

Изменение направления координат по осям X и Y:

```
#include <Wire.h>                                   // * Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Joystick.h>                  //   Подключаем библиотеку для работы с джойстиком I2C-flash.
iarduino_I2C_Joystick joy(0x09);                    //   Объявляем объект joy для работы с функциями и методами библиотеки iardu
```

```
                                                //   Если объявить объект без указания адреса (iarduino_I2C_Joystick joy;),
int a, b, c;                                    //
                                                //
                                                //
void setup(){                                   //
    delay(500);                                 // * Ждём завершение переходных процессов связанных с подачей питания.
    Serial.begin(9600);                         //
    while(!Serial){;}                           // * Ждём завершения инициализации шины UART.
    joy.begin();                                //   Инициируем работу с джойстиком.
//  Изменение направления оси X:                //
    joy.getCalibration_X(a, b, c);              //   Читаем установленные  калибровочные значения для оси X в переменные a,b
    joy.setCalibration_X(c, b, a);              //   Сохраняем прочитанные калибровочные значения для оси X, но указываем ap
//  Изменение направления оси Y:                //
    joy.getCalibration_Y(a, b, c);              //   Читаем установленные  калибровочные значения для оси Y в переменные a,b
    joy.setCalibration_Y(c, b, a);              //   Сохраняем прочитанные калибровочные значения для оси Y, но указываем ap
}                                               //
                                                //
void loop(){                                    //
    joy.getPosition(a,b);                       //   Получаем координаты в переменные a и b.
    Serial.println( (String)"X:Y="+a+":"+b );   //   Выводим  координаты джойстика.
}                                               //
```

После загрузки данного примера, в мониторе последовательного порта будут появляться строки с текущими координатами джойстика. Но с каждым новым запуском скетча направление координат будет меняться (слева отрицательные значения, справа положительные и наоборот).

Обратите внимание на то, что направление координат по осям сохраняется в энергонезависимой памяти модуля, значит если Вам нужно сменить направление осей, запустите этот скетч однократно.

Описание функций библиотеки:

В данном разделе описаны функции библиотеки iarduino_I2C_Joystick для работы с Trema модулем - Джойстик, 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_Joystick.h>  // Подключаем библиотеку для работы с модулем.
iarduino_I2C_Joystick joy(0x09);    // Создаём объект joy для работы с функциями и методами библиотеки iarduino_I2C_Joystick,
```

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

```
#include <iarduino_I2C_Joystick.h>  // Подключаем библиотеку для работы с модулем.
iarduino_I2C_Joystick joy;          // Создаём объект joy для работы с функциями и методами библиотеки iarduino_I2C_Joystick.
```

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( joy.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( joy.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:
  - Адрес модуля сохраняется в энергонезависимую память, а значит будет действовать и после отключения питания.
  - Текущий адрес модуля можно узнать функцией getAddress().
- Пример:

```
if( joy.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( joy.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( joy.getVersion() );
```

Функция getPosition_X();

- Назначение: Запрос текущей координаты джойстика по оси X.
- Синтаксис: getPosition_X();
- Параметры: Нет.
- Возвращаемое значение: int - координата от -100 (лево) до +100 (право), значение 0 (центр).
- Пример:

```
Serial.print( "Координата по оси X = " );
Serial.println( joy.getPosition_X() );
```

Функция getPosition_Y();

- Назначение: Запрос текущей координаты джойстика по оси Y.

- Синтаксис: getPosition_Y();
- Параметры: Нет.
- Возвращаемое значение: int - координата от -100 (низ) до +100 (верх), значение 0 (центр).
- Пример:

```
Serial.print( "Координата по оси Y = " );
Serial.println( joy.getPosition_Y() );
```

Функция getPosition();

- Назначение: Запрос текущих координат джойстика по осям X и Y.
- Синтаксис: getPosition(X , Y);
- Параметры:
  - X - переменная в которую требуется сохранить координату по оси X.
  - Y - переменная в которую требуется сохранить координату по оси Y.
- Возвращаемое значение: bool - результат чтения координат (true или false).
- Примечание:
  - Данная функция позволяет получить те же координаты, что и функции `getPosition_X()` и `getPosition_Y()`, но получение двух координат одним запросом выполняется быстрее.
  - В качестве параметров можно указывать переменные любых типов способных хранить числа со знаком.
- Пример:

```
int a, b;
joy.getPosition(a,b);
Serial.println( (String) "Координаты X:Y = " + a + ":" + b );
```

Функция getADC_X();

- Назначение: Запрос сырого значения АЦП снятого с потенциометра по оси X.
- Синтаксис: getADC_X();

- Параметры: Нет.
- Возвращаемое значение: int - значение АЦП (12 бит) от 0 до 4095.
- Примечание: Показания АЦП снятые с потенциометра по оси X, можно использовать как альтернативный метод получения координаты джойстика, а так же для калибровки джойстика, указывая значения АЦП в качестве параметров функции `setCalibration_X()` .
- Пример:

```
Serial.print( "Текущему положению джойстика по оси X соответствует значение АЦП = " );
Serial.println( joy.getADC_X() );
```

Функция getADC_Y();

- Назначение: Запрос сырого значения АЦП снятого с потенциометра по оси Y.
- Синтаксис: getADC_Y();
- Параметры: Нет.
- Возвращаемое значение: int - значение АЦП (12 бит) от 0 до 4095.
- Примечание: Показания АЦП снятые с потенциометра по оси Y, можно использовать как альтернативный метод получения координаты джойстика, а так же для калибровки джойстика, указывая значения АЦП в качестве параметров функции `setCalibration_Y()` .
- Пример:

```
Serial.print( "Текущему положению джойстика по оси Y соответствует значение АЦП = " );
Serial.println( joy.getADC_Y() );
```

Функция getADC();

- Назначение: Запрос сырых значений АЦП снятых с потенциометров по осям X и Y.
- Синтаксис: getADC(X , Y);
- Параметры:
  - X - переменная в которую требуется сохранить значение АЦП для оси X.
  - Y - переменная в которую требуется сохранить значение АЦП для оси Y.
- Возвращаемое значение: bool - результат чтения значений АЦП (true или false).

- Примечание:
  - Данная функция позволяет получить те же значения, что и функции `getADC_X()` и `getADC_Y()`, но получение двух значений одним запросом выполняется быстрее.
  - В качестве параметров можно указывать переменные любых типов способных хранить числа от 0 до 4095.
- Пример:

```
int a, b;
joy.getADC(a,b);
Serial.print( "Текущему положению джойстика по осям X и Y соответствуют значения АЦП = " );
Serial.println( (String) a + ":" + b );
```

Функция getButton();

- Назначение: Запрос времени удержания, состояния, или события кнопки джойстика.
- Синтаксис: getButton(ЗАПРОС);
- Параметр: uint8_t - один из перечисленных вариантов...
  - KEY_PUSHED - вернуть событие кнопки - «нажимается».
  - KEY_RELEASED - вернуть событие кнопки - «отпускается».
  - KEY_CHANGED - вернуть событие кнопки - «состояние изменилось».
  - KEY_PRESSED - вернуть состояние кнопки - «нажата».
  - KEY_TRIGGER - вернуть состояние кнопки - «переключатель».
  - KEY_HOLD_05 - вернуть состояние кнопки - «удерживается» дольше 0,5 сек.
  - KEY_HOLD_10 - вернуть состояние кнопки - «удерживается» дольше 1,0 сек.
  - KEY_HOLD_20 - вернуть состояние кнопки - «удерживается» дольше 2,0 сек.
  - KEY_TIME_PRESSED - вернуть время удержания кнопки в миллисекундах.
- Возвращаемое значение: uint16_t - значение зависит от параметра функции:
  - ФЛАГ наличия запрашиваемого состояния или события (true или false).
  - ВРЕМЯ удержания кнопки в миллисекундах (от 0 до 25500).
- Примечание:
  - Функция доступна только для джойстика с кнопкой.

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

Функция setCalibration_X();

- Назначение: Калибровка координат по оси X.
- Синтаксис: setCalibration_X(ЛЕВО , ЦЕНТР , ПРАВО);
- Параметры:
  - int ЛЕВО - значение АЦП соответствующее крайнему левому положению джойстика.
  - int ЦЕНТР - значение АЦП соответствующее центральному положению джойстика.
  - int ПРАВО - значение АЦП соответствующее крайнему правому положению джойстика.
- Возвращаемое значение: bool - результат сохранения калибровочных значений (true или false).
- Примечание:
  - Значения АЦП для оси X можно получить функцией `getADC_X()` или `getADC()`, предварительно установив джойстик в требуемое положение.
  - Калибровочные значения, указанные функцией `setCalibration_X()` сохраняются в энергонезависимой памяти модуля, а значит будут действовать и после отключения питания.
  - Калибровка координат по оси X влияет только на значения которые, после калибровки, будут возвращать функции `getPosition_X()` и `getPosition()`.
  - После калибровки, координаты положения джойстика будут рассчитываться модулем следующим образом:

    - Если джойстик находится левее центра, то координата джойстика `getPosition_X()` будет равна значению АЦП `getADC_X()` преобразованному от диапазона `ЛЕВО...ЦЕНТР` к диапазону `-100...0`.
    - Если джойстик находится в центре (в положении где значение АЦП `getADC_X()` равно значению `ЦЕНТР`), то координата джойстика `getPosition_X()` будет равна `0`.
    - Если джойстик находится правее центра, то координата джойстика `getPosition_X()` будет равна значению АЦП `getADC_X()` преобразованному от диапазона `ЦЕНТР...ПРАВО` к диапазону `0...100`.
  - Если требуется повернуть направление координат по оси X (так что бы крайнему левому положению соответствовала координата 100, а крайнему правому, координата -100), то функцию необходимо вызвать с параметрами указанными в обратном порядке `setCalibration_X( ПРАВО , ЦЕНТР , ЛЕВО )`.
- Пример:

```
joy.setCalibration_X( 0 , 2047 , 2095 );
```

Функция setCalibration_Y();

- Назначение: Калибровка координат по оси Y.
- Синтаксис: setCalibration_Y(НИЗ , ЦЕНТР , ВЕРХ);
- Параметры:
  - int НИЗ - значение АЦП соответствующее крайнему нижнему положению джойстика.
  - int ЦЕНТР - значение АЦП соответствующее центральному положению джойстика.
  - int ВЕРХ - значение АЦП соответствующее крайнему верхнему положению джойстика.
- Возвращаемое значение: bool - результат сохранения калибровочных значений (true или false).
- Примечание:
  - Значения АЦП для оси Y можно получить функцией `getADC_Y()` или `getADC()`, предварительно установив джойстик в требуемое положение.
  - Калибровочные значения, указанные функцией `setCalibration_Y()` сохраняются в энергонезависимой памяти модуля, а значит будут действовать и после отключения питания.
  - Калибровка координат по оси Y влияет только на значения которые, после калибровки, будут возвращать функции `getPosition_Y()` и `getPosition()`.

- После калибровки, координаты положения джойстика будут рассчитываться модулем следующим образом:
  - Если джойстик находится ниже центра, то координата джойстика `getPosition_Y()` будет равна значению АЦП `getADC_Y()` преобразованному от диапазона `НИЗ...ЦЕНТР` к диапазону `-100...0` .
  - Если джойстик находится в центре (в положении где значение АЦП `getADC_Y()` равно значению `ЦЕНТР`), то координата джойстика `getPosition_Y()` будет равна `0` .
  - Если джойстик находится выше центра, то координата джойстика `getPosition_Y()` будет равна значению АЦП `getADC_Y()` преобразованному от диапазона `ЦЕНТР...ВЕРХ` к диапазону `0...100` .
- Если требуется повернуть направление координат по оси Y (так что бы крайнему нижнему положению соответствовала координата 100, а крайнему верхнему, координата -100), то функцию необходимо вызвать с параметрами указанными в обратном порядке `setCalibration_Y( ВЕРХ , ЦЕНТР , НИЗ )` .

- Пример:

```
joy.setCalibration_Y( 0 , 2047 , 2095 );
```

Функция getCalibration_X();

- Назначение: Запрос калибровочных значений используемых модулем для расчёта координат оси X.
- Синтаксис: getCalibration_X(ЛЕВО , ЦЕНТР , ПРАВО);
- Параметры:
  - ЛЕВО - переменная в которую требуется сохранить АЦП левого положения.
  - ЦЕНТР - переменная в которую требуется сохранить АЦП центрального положения.
  - ПРАВО - переменная в которую требуется сохранить АЦП правого положения.
- Возвращаемое значение: bool - результат чтения значений (true или false).
- Примечание:
  - Функция `getCalibration_X()` читает калибровочные значения установленные функцией `setCalibration_X()` .
  - В качестве параметров можно указывать переменные любых типов способных хранить числа от 0 до 4095.
- Пример:

```
int a,b,c;                    // Объявляем переменные тип которых может хранить числа от 0 до 4095.
```

```
joy.getCalibration_X(a,b,c); // Читаем установленные калибровочные значения для оси X.
joy.setCalibration_X(c,b,a); // Устанавливаем прочитанные значения, но в обратном порядке.
```

Результатом приведённого примера будет смена направления координат по оси X. Если крайнему левому положению соответствовала координата -100, а крайнему правому +100, то теперь крайнему левому положению будет соответствовать координата +100, а крайнему правому -100. Центральное положение оставлено без изменений.

Функция getCalibration_Y();

- Назначение: Запрос калибровочных значений используемых модулем для расчёта координат оси Y.
- Синтаксис: getCalibration_Y(НИЗ , ЦЕНТР , ВЕРХ);
- Параметры:
  - НИЗ - переменная в которую требуется сохранить АЦП нижнего положения.
  - ЦЕНТР - переменная в которую требуется сохранить АЦП центрального положения.
  - ВЕРХ - переменная в которую требуется сохранить АЦП верхнего положения.
- Возвращаемое значение: bool - результат чтения значений (true или false).
- Примечание:
  - Функция `getCalibration_Y()` читает калибровочные значения установленные функцией `setCalibration_Y()` .
  - В качестве параметров можно указывать переменные любых типов способных хранить числа от 0 до 4095.
- Пример:

```
int a,b,c;                  // Объявляем переменные тип которых может хранить числа от 0 до 4095.
joy.getCalibration_Y(a,b,c); // Читаем установленные калибровочные значения для оси Y.
joy.setCalibration_Y(c,b,a); // Устанавливаем прочитанные значения, но в обратном порядке.
```

Результатом приведённого примера будет смена направления координат по оси Y. Если крайнему нижнему положению соответствовала координата -100, а крайнему верхнему +100, то теперь крайнему нижнему положению будет соответствовать координата +100, а крайнему верхнему -100. Центральное положение оставлено без изменений.

Функция setDeadZone();

- Назначение: Установка мертвой зоны центрального положения джойстика.
- Синтаксис: setDeadZone(ЗОНА);
- Параметр: float ЗОНА - значение от 0.0% до 25.5%, с шагом 0,1.
- Возвращаемое значение: bool - результат применения новой зоны (true или false).
- Примечание:
  - Устанавливаемое значение определяет размер мёртвой зоны у центрального положения джойстика. Этот показатель определяет, на сколько % требуется отклонить джойстик от центрального положения (по любой оси), что бы его координаты начали меняться. Пока джойстик находится в мёртвой зоне, его координаты будут равны 0:0.
  - Мёртвая зона, это зона не чувствительности к отклонению джойстика. Она позволяет избежать "дрожание" (jitter) координат джойстика в центральном положении.
  - Мёртвую зону можно использовать в проектах, где управление устройствами должно начинаться после значительного сдвига джойстика от центра.
  - Размер мёртвой зоны сохраняется в энергонезависимую память модуля, а значит будет действовать и после отключения питания.
- Пример:

```
joy.setDeadZone(23.4); // Установить мёртвую зону в 23,4% от полного хода джойстика из центра к любому краю.
```

Функция setAveraging();

- Назначение: Установка коэффициента усреднения показаний.
- Синтаксис: setAveraging(КОЭФФИЦИЕНТ);
- Параметр: uint8_t КОЭФФИЦИЕНТ - значение от 0 до 255, определяющее сглаживание показаний АЦП и координат по осям X и Y.
- Возвращаемое значение: bool - результат применения нового усреднения (true или false).
- Примечание:
  - Чем выше сглаживание, тем плавнее будут меняться показания АЦП, а значит плавнее будут меняться координаты положения джойстика, даже если джойстик отклоняется резко.
  - Значение 0 означает что сглаживание показаний отключено. Значение по умолчанию 2.
  - Сглаживание показаний может понадобиться в проектах, где требуется плавное управление.
  - Коэффициент усреднения показаний сохраняется в энергонезависимую память модуля, а значит будет действовать и после отключения

питания.

- Пример:

```
joy.setAveraging(150); // Увеличить сглаживание до 150, это приведёт к большой инерционности получаемых координат.
```