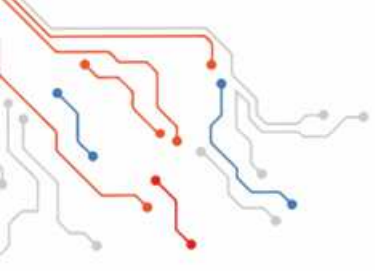
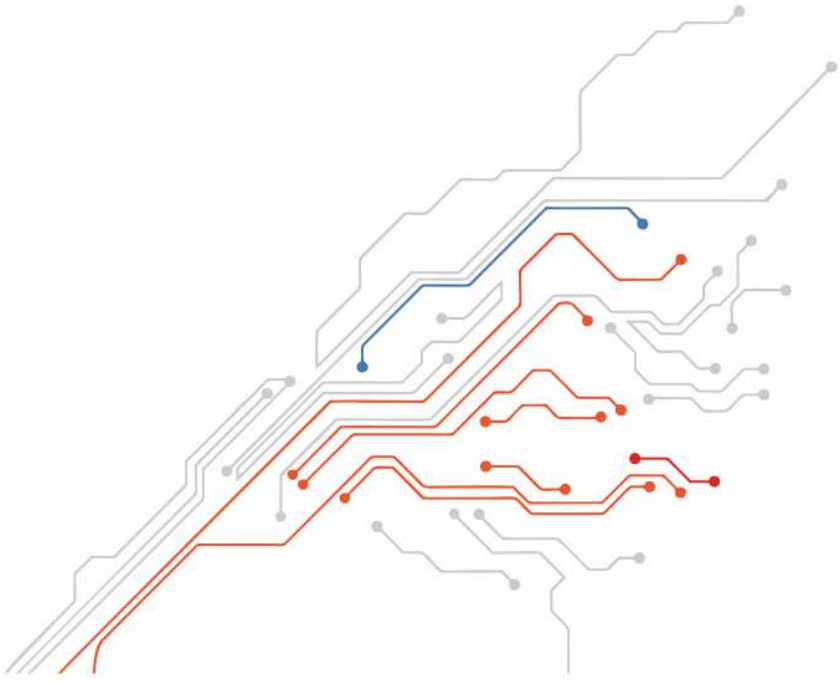


Future Forms

Начало работы с FFS / FFC



Оглавление

Базовая информация

Flexible Force Sensor - новаторская продукция от Future Forms. Датчик, способный проводить измерения силы по трем осям одновременно.

Датчик выпускается в двух модификациях, с поддержкой протоколов SPI или I2C.

Принцип работы с датчиком

Для работы с датчиком мы предлагаем библиотеку FFSensor, позволяющую считывать и обрабатывать данные, измеренные датчиком.

При использовании библиотеки понадобится специальная матрица калибровки, поставляемая вместе с датчиком. Она позволяет учитывать индивидуальные характеристики датчика и получать точные результаты измерений.

Матрицу калибровки можно получить у нас на сайте - futureforms.ru

Работа с датчиком обеспечивается двумя программными объектами. Первый (`FFS_MLX_SPI` или `FFS_MLX_I2C`) предоставляет интерфейс для общения с датчиком. Второй (FFSensor) обеспечивает обработку данных.

FFS_MLX_SPI использует протокол SPI для общения с датчиками в SPI исполнении, а FFS_MLX_I2C работает с I2C версией.

Для работы с несколькими датчиками можно использовать один FFS_MLX_SPI/FFS_MLX_I2C объект и соответствующее число объектов FFSensor.

Инициализация FFS_MLX_SPI

FFS_MLX_SPI инициализируется номером цифрового пина, обеспечивающего enable сигнал для работы с датчиком:

FFS_MLX_SPI mlx(10);

Инициализация FFS_MLX_I2C

FFS_MLX_I2C инициализируется I2C адресом. Обычно это 0x0c.

FFS_MLX_SPI mlx(0x0c);

Инициализация FFSensor

FFSensor инициализируется коммуникационным объектом (FFS_MLX_SPI или FFS_MLX_I2C) и матрицей калибровки:

FFSensor sensor(mlx, calibration);

Матрица калибровки

Матрица калибровки описывает характеристики конкретного датчика. Она состоит из двадцати чисел и должна записываться в постоянную память контроллера.

```
const float calibration48632[FFS::calibrationSize] PROGMEM = {
  -0.050739763217983125,
  0.0004592219302516076,
  0.0001370235414034915,
  -6.097381480175736 * pow(10, -9),
  -1.6357071084737373 * pow(10, -8),
  -3.435171418686141 * pow(10, -8),
  -1.5776737878482813 * pow(10, -11),
  2.0819115165347432 * pow(10, -11),
  -7.588318553423788 * pow(10, -12),
  3.231639745911275 * pow(10, -12),
  0.13553180660221642,
  -1.1537948532258646 * pow(10, -5),
  0.001061136092719961,
  1.2486110514159692 * pow(10, -7),
  -3.267484867381211 * pow(10, -8),
  -1.7091529069693332 * pow(10, -8),
  -2.4056254221181725 * pow(10, -13),
  -1.2274910008994921 * pow(10, -11),
  5.022508003905093 * pow(10, -12),
  5.438868068427872 * pow(10, -13),
};
```

Запрос измерения усилия через FFSensor

FFSensor может запросить измерение усилия через вызов requestMeasurement.

```
sensor.requestMeasurement();
```

Обработать результаты измерения можно через вызов readMeasurement.

```
sensor.readMeasurement();
```

Между requestMeasurement и readMeasurement должно пройти около 10 мс. В это время микроконтроллер может выполнять другую полезную работу.

При использовании нескольких датчиков можно сначала запросить измерения на всех датчиках, а потом обработать данные со всех сразу.

Настройка Arduino IDE

Использование отладочной платы ProMicro 3.3V

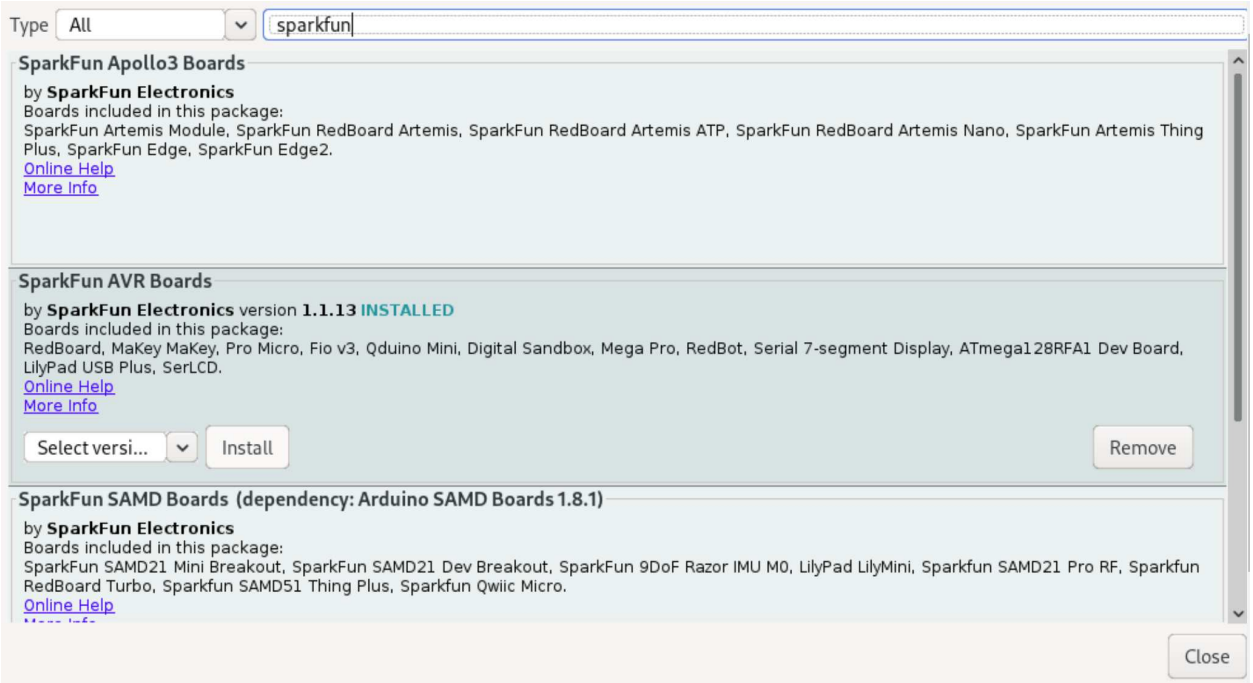
Для использования платы ProMicro 3.3V от SparkFun, включенной в стартовый набор, вам понадобится установить дополнение для среды Arduino IDE.

Подключение дополнительных данных для менеджера плат

Выберите **Файл->Настройки** (или *File->Preferences*) и добавьте строку:

https://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_Board_Manager/package_sparkfun_index.json

в поле **Дополнительные ссылки менеджера плат** (*Additional Board Manager URLs*).



Загрузка описаний дополнительных плат

Выберите **Инструменты->Плата->Менеджер плат...** (*Tools->Board->Boards Manager...*). В строке поиска введите sparkfun и установите SparkFun AVR Boards нажатием кнопки **Установить** (*Install*).

Выбор платы ProMicro 3.3V

Выберите **Инструменты->Плата->SparkFun Pro Micro** (*Tools->Board->SparkFun Pro Micro*). Укажите тип процессора **Инструменты->Процессор->ATMega32U4 (3.3V, 8MHz)** (*Tools->Processor->ATMega32U4 (3.3V, 8MHz)*).

Подключение датчика

Будьте внимательны, датчик работает с напряжением 3.3V!

Стандартные платы семейства Arduino могут повредить его. Мы советуем использовать Pro Micro 3.3V от SparkFun Electronics или подобные им платы с уровнем логики 3.3V.

Назначение проводов

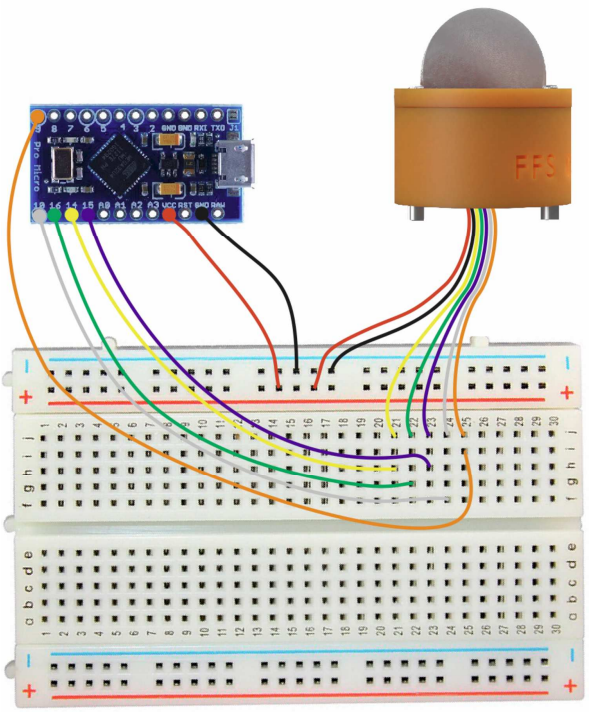
Провода дифференцированы по цвету для упрощения подключения датчика.

SPI версия датчика

SPI версия датчика оснащена семью проводами на одном разъеме:

Провод	Назначение
Красный	3.3V
Черный	0V
Желтый	MISO
Зеленый	MOSI
Синий	SCLK
Белый	EN
Оранжевый	LED

В примерах мы будем подключать SPI датчики следующим образом:

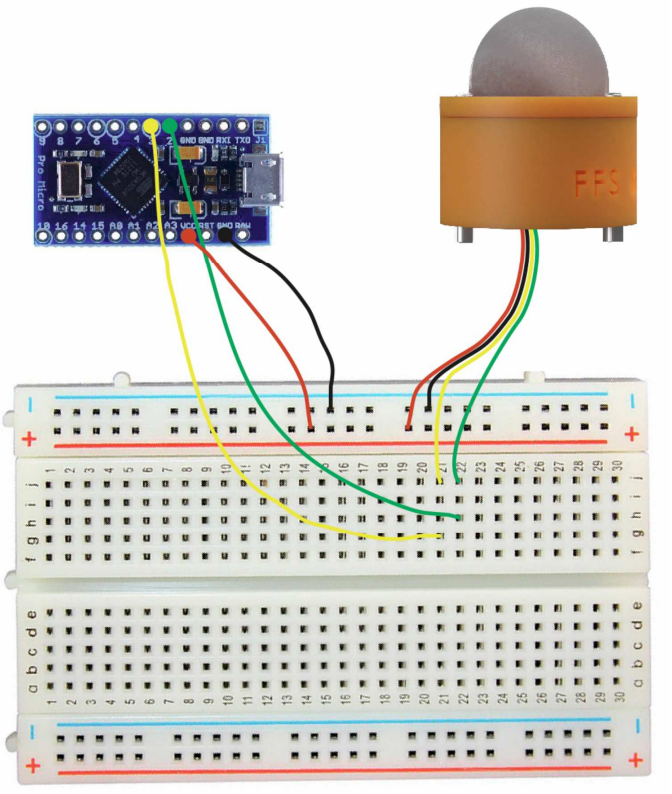


Провод	Пин
Красный	VCC
Черный	GND
Желтый	14
Зеленый	16
Синий	15
Белый	10
Оранжевый	9

I2C версия датчика

I2C версия поставляется с четырьмя проводами на одном разъеме: Провод | Назначение ---------:|:---------- Красный | 3.3V Черный | 0V Желтый | SCL Зеленый | SDA

В примерах мы будем подключать I2C датчики следующим образом:



Провод	Пин
Красный	VCC
Черный	GND
Желтый	3
Зеленый	2

Чтение данных с SPI датчика

Процесс работы с датчиком

Для получения данных об усилии с датчика необходимо воспользоваться матрицей калибровки, поставляемой вместе с датчиком. Матрица калибровки - таблица из 20 чисел, позволяющая перевести показания датчика в значения усилий.

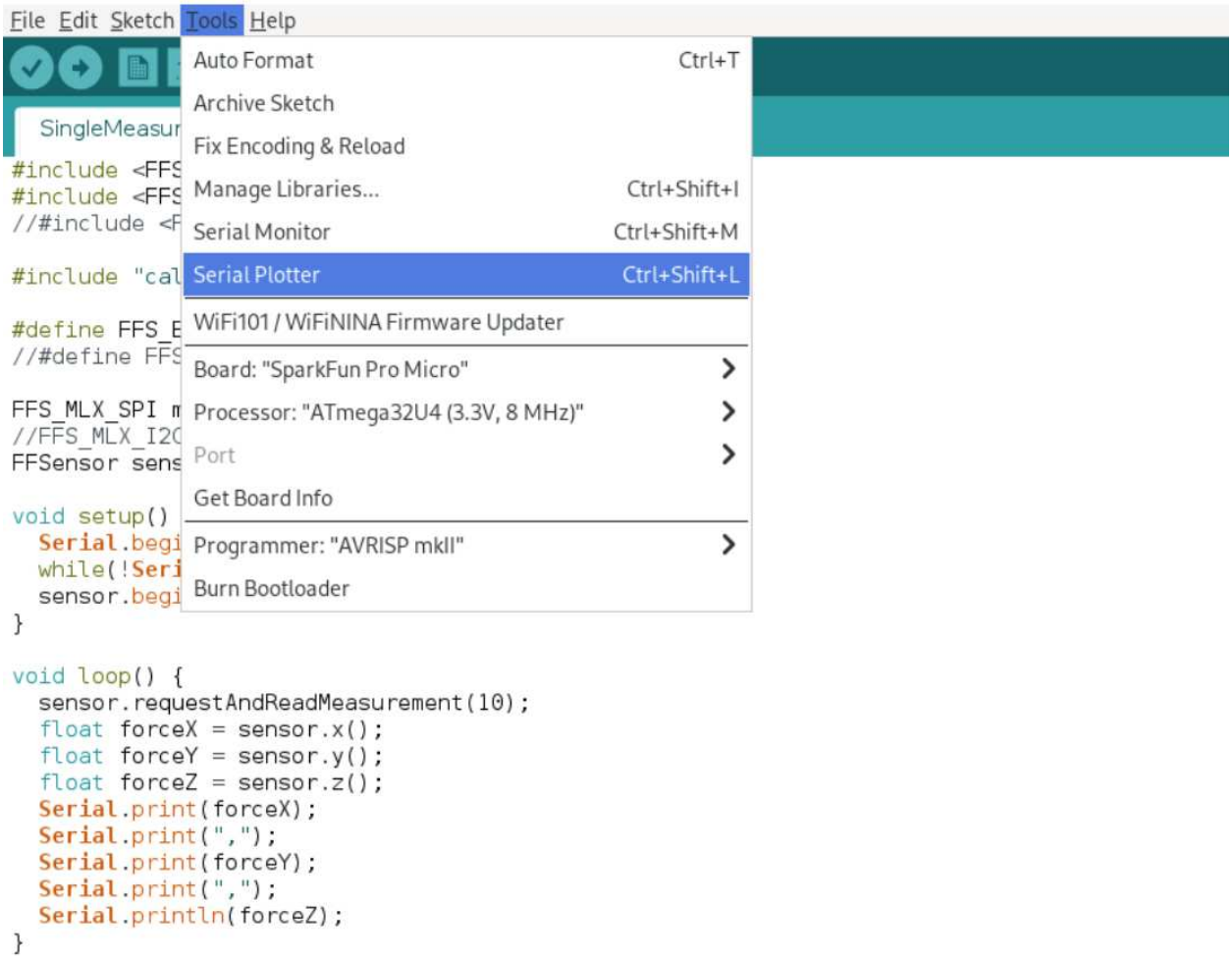
После инициализации датчика необходимо выполнить измерение и получить результат. Это делается командой requestAndReadMeasurement(delay). Значение delay отмечает время между запросом на измерение и запросом результата измерения в миллисекундах. Дело в том, что датчик не может мгновенно провести измерение, поэтому необходимо вводить разумную задержку. Рекомендуемое время ожидания при стандартных настройках - 10 миллисекунд.

Использование демонстрационного кода

Скомпилируйте и загрузите следующий код на отладочную плату:

```
#include <FFSensor.h>
#include <FFS_MLX_SPI.h>

#include "calibration.h"

#define FFS_ENABLE_PIN  10

FFS_MLX_SPI mlx(FFS_ENABLE_PIN);
FFSensor sensor(mlx, calibration);

void setup() {
  Serial.begin(9600);
  while(!Serial){}
  sensor.begin();
}

void loop() {
  sensor.requestAndReadMeasurement(10);
  float forceX = sensor.x();
  float forceY = sensor.y();
  float forceZ = sensor.z();
  Serial.print(forceX);
  Serial.print(",");
  Serial.print(forceY);
  Serial.print(",");
  Serial.println(forceZ);
}
```

Откройте плоттер и посмотрите на графики усилий.



Обратите внимание как приложение усилия к датчику влияет на значения, выводимые на экран.

Разбор демонстрационного кода

Сначала подключаем заголовочные файлы для работы с датчиком:

```
#include <FFSensor.h>
#include <FFS_MLX_SPI.h>
```

Затем подключаем данные калибровки:

```
#include "calibration.h"
```

Указываем какой вывод использовать для включения датчика:

```
#define FFS_ENABLE_PIN  10
```

Создаем объект для обмена данными с датчиком:

```
FFS_MLX_SPI mlx(FFS_ENABLE_PIN);
```

И объект для обработки данных:

```
FFSensor sensor(mlx, calibration);
```

В блоке setup указываем действия, выполняемые единожды при инициализации. Начинаем общение по последовательному порту с частотой 9600, ожидаем подключения по последовательному порту, после чего инициализируем сенсор:

```
void setup() {
  Serial.begin(9600);
  while(!Serial){}
  sensor.begin();
}
```

В блоке loop описываем повторяющиеся действия.

```
void loop() {
  sensor.requestAndReadMeasurement(10);
  float forceX = sensor.x();
  float forceY = sensor.y();
  float forceZ = sensor.z();
  Serial.print(forceX);
  Serial.print(",");
  Serial.print(forceY);
  Serial.print(",");
  Serial.println(forceZ);
}
```

Запрашиваем измерение данных датчиком и обрабатываем их, дав датчику 10 мс на измерение:

```
sensor.requestAndReadMeasurement(10);
```

Читаем обработанные данные. Для этого запрашиваем значения x, y и z:

```
float forceX = sensor.x();
float forceY = sensor.y();
float forceZ = sensor.z();
```

Выводим данные на последовательный порт:

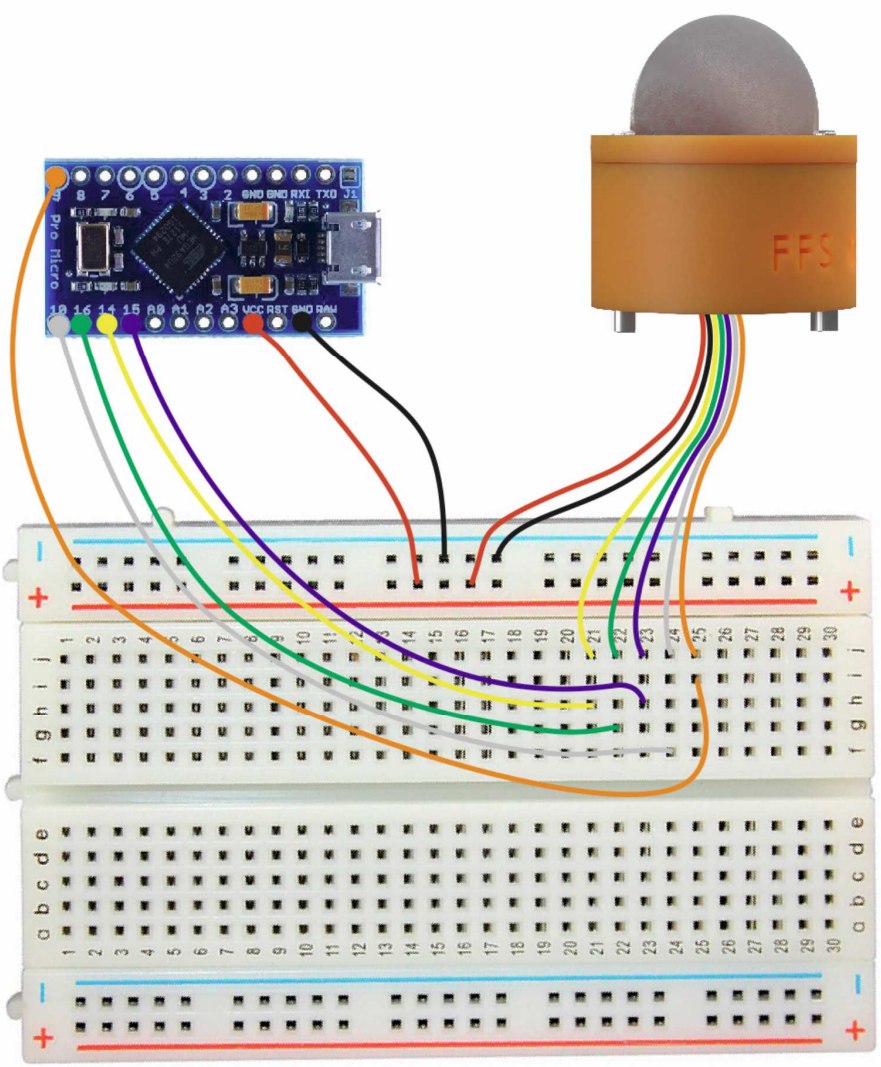
```
Serial.print(forceX);
Serial.print(",");
Serial.print(forceY);
Serial.print(",");
Serial.println(forceZ);
```

Пример подключения к плате Pro Micro 3.3V

В этом примере рассмотрим, как подключить датчик к плате Pro Micro 3.3V и проверить правильность подключения.

SPI версия датчика

Провод	Пин	Назначение
Красный	VCC	3.3V
Черный	GND	0V
Желтый	14	MISO
Зеленый	16	MOSI
Синий	15	SCLK
Белый	10	EN
Оранжевый	9	LED



```ino
// Библиотека для работы с датчиком
#include <FFSensor.h>
// Реализация протокола для общения с SPI версией FFS
#include <FFS_MLX_SPI.h>

// Подключение файла калибровочных данных
#include "calibration.h"

// Указание EN пина для активации FFS
#define FFS_ENABLE_PIN  10

// Объект для реализации протокола общения с FFS
FFS_MLX_SPI mlx(FFS_ENABLE_PIN);
// Объект для обработки данных с датчика
FFSensor sensor(mlx, calibration);

void setup() {
  // Инициализируем последовательный порт
  Serial.begin(9600);
  // Ожидаем инициализации последовательного порта
  while(!Serial){}

  // Инициализируем сенсор и записываем статус
  byte status = sensor.begin();
  // Если статус содержит бит ошибки или не получен
  if (status & FFS_MLX::ERROR_BIT || status != 0) {
    // То вывести сообщение об ошибке
    Serial.println("Проблема при подключении датчика");
  } else {
    // Иначе вывести сообщение об успехе
    Serial.println("Датчик подключен успешно");
  }
}

void loop() {

}
```

Использование

Прошейте плату тестовым кодом и откройте монитор порта. Если датчик подключен верно вы увидите соответствующее сообщение.

Управление встроенным светодиодом

FFC обладает индикационным RGB светодиодом.

Для управления им можно использовать любую подходящую библиотеку. Мы рекомендуем к использованию библиотеку FastLED, но при желании вы можете адаптировать код для Adafruit_NeoPixel.

Пример: базовое управление свечением

В этом примере демонстрируется возможности встроенного светодиода.

```
#include <FastLED.h>

#define LED_PIN     9
#define LED_COUNT   1

CRGB led[LED_COUNT];

void setup() {
    FastLED.addLeds<WS2812B, PIXEL_PIN, GRB>(leds, 1);
}

void loop() {
  float time = millis() / 1000.0;
  leds[0] = CRGB(
    128 + 127 * sin(time),
    128 + 127 * sin(time + 2 * PI / 3),
    128 + 127 * sin(time + 4 * PI / 3));
  FastLED.show();
}
```

Разбор примера

Пример начинается с подключения библиотеки для управления светодиодом:

```
#include <FastLED.h>
```

Указываем какой пин управляет светодиодом, также указываем что у нас один светодиод:

```
#define LED_PIN     9
#define LED_COUNT   1
```

В блоке setup инициализируем светодиод:

void setup() {

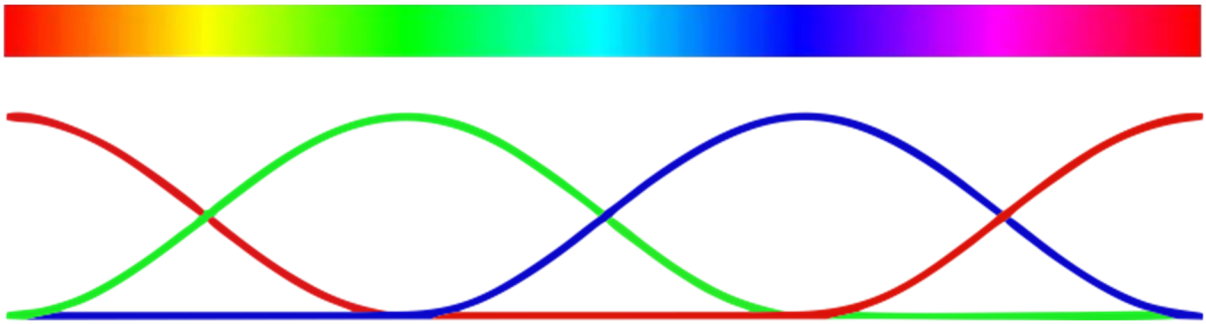
  FastLED.addLeds<WS2812B, PIXEL_PIN, GRB>(leds, 1);

```
}
```

В блоке loop определяем время, прошедшее с запуска платы:

```
void loop() {
  float time = millis() / 1000.0;
}
```

Выбираем цвет светодиода. Цвет задается тремя компонентами, красной, зеленой и синей, каждая компонента принимает значения от 0 до 255. В этом примере каждая компонента меняется волнообразно, три волны смещены друг относительно друга на равное расстояние:

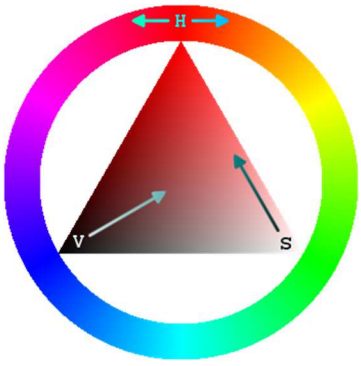


```
leds[0] = CRGB(
    128 + 127 * sin(time),
    128 + 127 * sin(time + 2 * PI / 3),
    128 + 127 * sin(time + 4 * PI / 3));
```

```
Переключаем цвет светодиода:

FastLED.show();
```

Пример: выбор HSV координаты с помощью датчика

HSV это цветовое пространство, в котором цвет описывается тремя параметрами:



Hue – Оттенок
Saturation - Насыщенность
Value – Значение