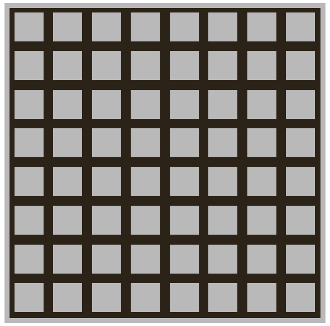
# LED Матрица 8x8 - i2c (Trema-модуль)



## Общие сведения:

*Trema-модуль I2C LED матрица 8x8* - это модуль с монохромной светодиодной матрицей размерами 8x8 пикселей. Модуль подключается по шине I2C и предназначен для вывода небольших изображений, цифр, символов или бегущей строки. Адрес для шины I2C назначается программно и сохраняется в энергонезависимой памяти модуля, благодаря чему к одной шине I2C можно подключить более 100 модулей (из которых можно создавать составные картинки).

## Спецификация:

- Напряжение питания: 5 В (постоянного тока)
- Потребляемый ток: до 35 мА.
- Тип матрицы: монохромная.
- Разрешение матрицы: 8x8.
- Количество светодиодов: 64.
- Цвет свечения: красный.
- Развёртка: задаётся от 10 до 255 кадров/сек (по умолчанию 100).
- Шаг регулировки яркости: 1/255.
- Буфер для хранения бегущей строки: до 512 символов.
- Скорость бегущей строки: задаётся от 0,06 до 15 секунд на символ.
- Таблица символов: CP866 (поддерживает Кириллицу).
- 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 выводах ADR: 3,3 В (толерантны к 5 В).
- Рабочая температура: от -40 до +65 °С.
- Габариты: 32x32 мм.
- Вес: 7 г.

## Подключение:

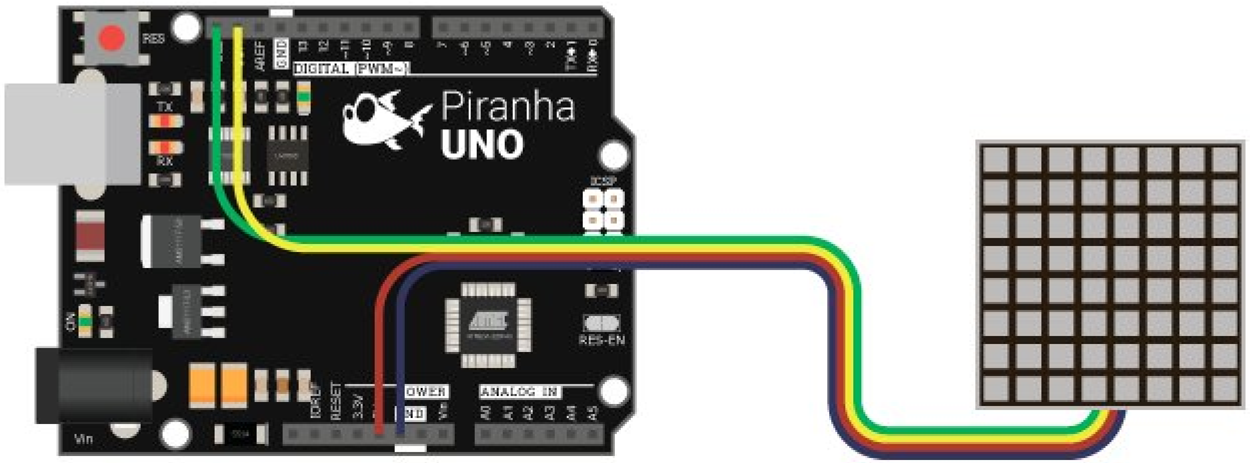
Trema-модуль I2C LED матрица 8x8 подключается к аппаратной или программной шине I2C Arduino.

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

Модуль удобно подключать 3 способами, в зависимости от ситуации:

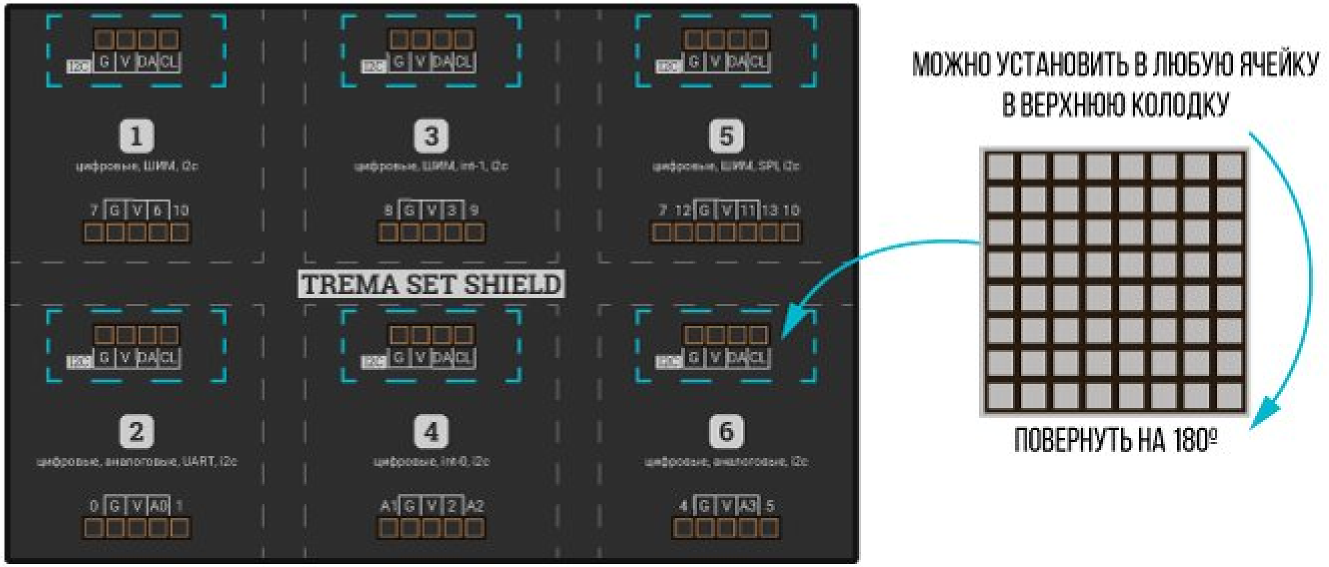
### Способ - 1 : Используя проводной шлейф и Piranha UNO

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



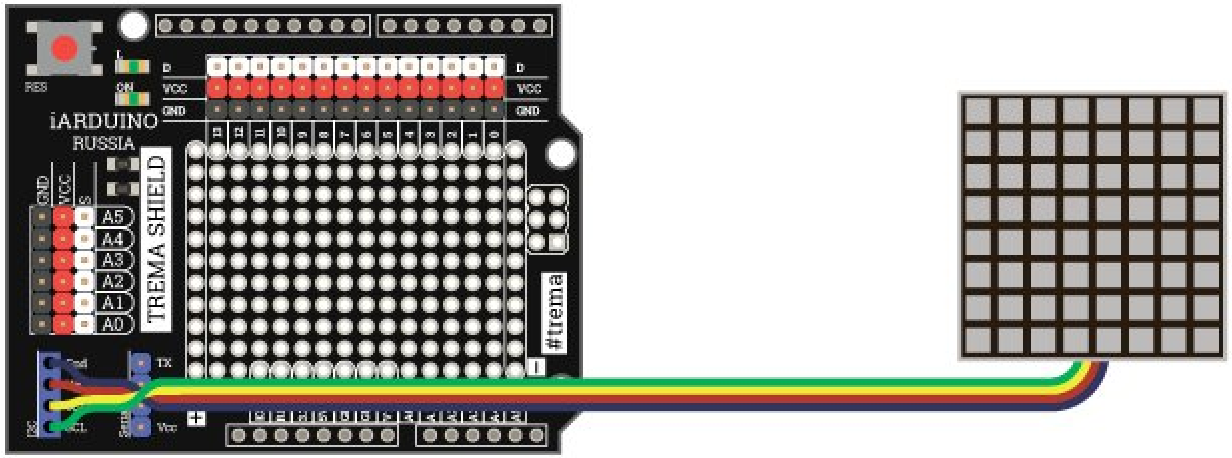
## Способ - 2 : Используя Trema Set Shield

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




## Способ - 3 : Используя проводной шлейф и Shield

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



При подключении Trema-модуля I2C LED матрица 8x8 к другим платам, например, WEMOS D1 mini или WEMOS D1 mini Pro на базе микроконтроллера ESP8266, и т.д. То перед подключением библиотеки iarduino_I2C_Matrix_8x8, нужно подключить библиотеку Wire, как это описано в разделе Wiki - расширенные возможности библиотек iarduino для шины I2C.

**Модуль не поддерживает горячее подключение:** Подключайте модуль только при отсутствии питания и данных на шине I2C. В противном случае потребуется отключить питание при уже подключённом модуле.

## Питание:

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

Модуль сохраняет работоспособность при снижении напряжения питания до 3,3 В, при незначительном снижении яркости светодиодов матрицы.

## Подробнее о модуле:

Trema-модуль I2C LED матрица 8x8 построен на базе микроконтроллера STM32F030K6 и снабжен собственным стабилизатором напряжения. Модуль отрисовывает изображение последовательно включая по одному светодиоду матрицы с частотой 100 кадров/сек. При такой частоте глаз не замечает мерцаний светодиодов и видит всё изображение матрицы целиком. В памяти модуля имеется таблица изображений символов в кодировке CP866 (поддерживает Кириллицу). Модуль имеет буфер на 512 символов для хранения текста бегущей строки.

Модуль позволяет:

- Залить и стереть дисплей матрицы, с возможностью анимации указанных действий.
- Вывести изображение размером 8x8 из массива размером в 8 байт.
- Вывести символ по его коду в кодировке CP866 (в т.ч. символ Кириллицы).
- Записать и вывести текст бегущей строки (до 512 символов) с указанием её скорости.
- Установить яркость матрицы.

- Установить частоту обновления экрана от 10 до 255 кадров/сек.
- Установить угол поворота изображений матрицы на 0°, 90°, 180° и 270°.
- Включить функцию анимации появления/исчезания изображения, символа или текста.
- Изменить изображение любого символа, задать ширину и межсимвольные интервалы.
- Установить текст бегущей строки в указанную позицию и пошагово её сдвигать.
- Задать направление и режим автопрокрутки бегущей строки.
- Задать адрес для шины I2C с сохранением в энергонезависимую память модуля.

Для работы с Trema-модулем I2C LED матрица 8x8 предлагаем воспользоваться разработанной нами библиотекой iarduino_I2C_Matrix_8x8, которая позволяет реализовать все функции модуля.

Подробнее про установку библиотеки читайте в нашей инструкции.

## Примеры:

### Мигание:

```
#include <Wire.h>                                   // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                 // Объявляем объект disp для работы с LED матрицей 8x8, указь
                                                    //
void setup(){                                       //
    delay(500);                                     // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                   // Инициируем работу с LED матрицей 8x8.
}                                                   //
                                                    //
void loop(){                                        //
    disp.fillScr(); delay(1000);                    // Заливаем экран (включаем  все светодиоды).
    disp.clrScr();  delay(1000);                    // Чистим   экран (выключаем все светодиоды).
}                                                   //
```

Данный пример включает и выключает все светодиоды матрицы. Дисплей светится 1 секунду и потухает на 1 секунду.

Обратите внимание на то, что адрес светодиодной матрицы `9` указан при объявлении объекта `disp`. Если на шине I2C имеется только одна светодиодная матрица, то её адрес можно не указывать (объявить объект `disp` без скобочек с адресом), тогда библиотека сама определит адрес светодиодной матрицы, даже если на шине есть другие устройства.

### Анимация:

```
#include <Wire.h>                                                  // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                               // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                                // Объявляем объект disp для работы с LED матрицей 8x8, указь
                                                                   //
void setup(){                                                      //
    delay(500);                                                    // Ждём завершение переходных процессов связанных с подачей г
    disp.begin();                                                  // Инициируем работу с LED матрицей 8x8.
}                                                                  //
                                                                   //
void loop(){                                                       //
    disp.fillScr(X8_RIPPLES); delay(1000);                         // Заливаем экран рябью (светодиоды включаются  хаотично и с
    disp.clrScr (X8_RIPPLES); delay(1000);                         // Чистим   экран рябью (светодиоды выключаются хаотично и с
    disp.fillScr(X8_DOWN   ); delay(1000);                         // Заливаем экран сверху вниз (светодиоды включаются  построч
    disp.clrScr (X8_DOWN   ); delay(1000);                         // Чистим   экран сверху вниз (светодиоды выключаются построч
    disp.fillScr(X8_TOP    ); delay(1000);                         // Заливаем экран снизу вверх (светодиоды включаются  построч
    disp.clrScr (X8_TOP    ); delay(1000);                         // Чистим   экран снизу вверх (светодиоды выключаются построч
}                                                                  //
```

Данный пример включает и выключает все светодиоды матрицы, но не разом (как в предыдущем примере), а постепенно (анимируя включение и выключение светодиодов).

### Изменение адреса на шине I2C:

```
uint8_t newAddress = 0x0A;                                         // Назначаемый модулю адрес.
```

```
                                                                  //
#include <wire.h>                                                 // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_i2c_matrix_8x8.h>                              // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp;                                     // Объявляем объект disp для работы с функциями и методами би
                                                                  // Если при объявлении объекта указать адрес, например, disp(
void setup(){                                                     //
    Serial.begin(9600);                                           //
    while(!Serial){;}                                             //
    delay(500);                                                   // Ждём завершение переходных процессов связанных с подачей п
    if( disp.begin() ){                                           // Инициируем работу с LED матрицей 8x8.
        Serial.print("На шине I2C найден модуль с адресом 0x");   //
        Serial.print( disp.getAddress(), HEX );                   // Выводим текущий адрес модуля.
        Serial.print(" который является LED матрицей 8x8\r\n");   //
        if( disp.changeAddress(newAddress) ){                     // Меняем адрес модуля на newAddress.
            Serial.print("Адрес модуля изменён на 0x");           //
            Serial.println( disp.getAddress(), HEX );             // Выводим текущий адрес модуля.
        }else{                                                    //
            Serial.println("Адрес модуля изменить не удалось!");  //
        }                                                         //
    }else{                                                        //
        Serial.println("Модуль LED матрица 8x8 не найден!");      //
    }                                                             //
}                                                                 //
                                                                  //
void loop(){                                                      //
}                                                                 //</iarduino_i2c_matrix_8x8.h></wire.h>
```

Данный пример присвоит светодиодной матрице адрес 10, который будет сохранён в энергонезависимой памяти модуля, а значит останется и после выключения питания.

Обратите внимание на то, что в данном примере (в отличии от всех остальных) объект `disp` был объявлен без указания изначального адреса светодиодной матрицы, значит этот пример будет работать только когда на шине только одна светодиодная матрица (и Вам не обязательно знать её изначальный адрес). Если на шине несколько светодиодных матриц, то объект `disp` необходимо объявлять с указанием адреса той матрицы с которой Вы желаете работать.

### Вывод цифр:

```
int i=9;                                                         // Переменная цифру которой мы будем выводить.
                                                                 //
#include <Wire.h>                                                // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                             // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                              // Объявляем объект disp для работы с LED матрицей 8x8, указы
                                                                 //
void setup(){                                                    //
    delay(500);                                                  // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                                // Инициируем работу с LED матрицей 8x8.
}                                                                //
                                                                 //
void loop(){                                                     //
    if(millis()%1000<100){ delay(100);                           // Выполняем код оператора if один раз в секунду.
        i++;                                                     // Увеличиваем значение переменной i.
        if(i>9){i=0;}                                            // Но не даём значению i выйти за пределы 9.
        disp.print(i);                                           // Выводим значение i на дисплей.
    }                                                            //
}                                                                //
```

Данный пример постоянно выводит цифры от 0 до 9 меняя их каждую секунду. Вывод цифр можно анимировать указав название анимации в качестве второго аргумента функции `print()` .

### Вывод символов:

```
#include <Wire.h>                                                // Подключаем библиотеку для работы с аппаратной шиной I2C.
```

```
#include <iarduino_I2C_Matrix_8x8.h>                                 // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                                  // Объявляем объект disp для работы с LED матрицей 8x8, указь
                                                                     //
void setup(){                                                        //
    delay(500);                                                      // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                                    // Инициируем работу с LED матрицей 8x8.
}                                                                    //
                                                                     //
void loop(){                                                         //
    disp.print('i');         delay(300);                             // Выводим символ 'i'.
    disp.print('A');         delay(300);                             // Выводим символ 'A'.
    disp.print('r');         delay(300);                             // Выводим символ 'r'.
    disp.print('d');         delay(300);                             // Выводим символ 'd'.
    disp.print('u');         delay(300);                             // Выводим символ 'u'.
    disp.print( char(105) ); delay(300);                             // Выводим символ 'i', по коду этого символа.
    disp.print('n');         delay(300);                             // Выводим символ 'n'.
    disp.print('o');         delay(300);                             // Выводим символ 'o'.
    disp.print(' ');         delay(500);                             // Выводим символ пробела.
}                                                                    //
```

Данный пример постоянно выводит символы слова «iArduino», задерживаясь на каждом символе по 0,3 сек.

### Вывод символов с анимацией:

```
#include <Wire.h>                                                    // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                                 // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                                  // Объявляем объект disp для работы с LED матрицей 8x8, указь
                                                                     //
void setup(){                                                        //
    delay(500);                                                      // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                                    // Инициируем работу с LED матрицей 8x8.
}                                                                    //
```

```
                                                           //
void loop(){                                               //
    disp.print('i', X8_FILLED_TOP); delay(300);            // Выводим символ 'i', с анимацией появления снизу вверх из з
    disp.print('A');                delay(300);            // Выводим символ 'A'.
    disp.print('r');                delay(300);            // Выводим символ 'r'.
    disp.print('d');                delay(300);            // Выводим символ 'd'.
    disp.print( char(117) );        delay(300);            // Выводим символ 'u', по коду этого символа.
    disp.print( i );                delay(300);            // Выводим символ 'i'.
    disp.print('n');                delay(300);            // Выводим символ 'n'.
    disp.print('o');                delay(300);            // Выводим символ 'o'.
    disp.fillScr(X8_TOP);                                  // Заливаем дисплей снизу вверх.
}                                                          //
```

Данный пример, как и предыдущий, выводит символы слова «iArduiono», но первый символ 'i' выводится с анимацией появления сверху вниз из закрашенного фона, а вместо вывода пробела используется функция заливки дисплея `fillScr()` с анимацией снизу вверх.

## Вывод символов Кириллицы:

```
#include <Wire.h>                                          // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                       // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                        // Объявляем объект disp для работы с LED матрицей 8x8, указы
                                                           //
void setup(){                                              //
    delay(500);                                            // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                          // Инициируем работу с LED матрицей 8x8.
    disp.codingDetect("п");                                // Выполняем автоопределение кодировки скетча.
}                                                          //
                                                           //
void loop(){                                               //
    disp.print("П");          delay(300);                  // Выводим символ 'П'.
    disp.print("р");          delay(300);                  // Выводим символ 'р'.
    disp.print("и");          delay(300);                  // Выводим символ 'и'.
```

```
    disp.print("в");         delay(300);                       // Выводим символ 'в'.
    disp.print("е");         delay(300);                       // Выводим символ 'е'.
    disp.print("\321\202");  delay(300);                       // Выводим символ 'т', по коду этого символа.
    disp.print(' ');         delay(500);                       // Выводим символ пробела.
}                                                              //
```

Данный пример отличается от предыдущего тем, что в нём однократно вызвана функция автоопределения кодировки скетча `codingDetect("п")` позволяющая библиотеке корректно определять символы Кириллицы. Дело в том, что в разных версиях Arduino IDE используется разная кодировка, а так же кодировка сохранённого скетча отличается от кодировки не сохранённого. Еще одним отличием является то, что символы в функции `print()` указываются в двойных, а не одинарных кавычках, так как если Ваш скетч использует кодировку UTF-8, то один символ Кириллицы занимает два байта строки.

### Вывод бегущей строки однократно:

```
#include <Wire.h>                                              // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                           // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                            // Объявляем объект disp для работы с LED матрицей 8x8, указь
                                                               //
void setup(){                                                  //
    delay(500);                                                // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                              // Инициируем работу с LED матрицей 8x8.
    disp.print("Hello World");                                 // Загружаем текст для бегущей строки (но не выводим её).
    disp.autoScroll(240);                                      // Выводим (разрешаем прокрутить) бегущую строку однократно с
}                                                              //
                                                               //
void loop(){                                                   //
}                                                              //
```

Данный пример однократно выводит текст "Hello World" бегущей строкой со скоростью 240. Функция `autoScroll()` не ждёт завершение прокрутки бегущей строки, так как прокрутка осуществляется модулем не используя ресурсов Arduino.

### Вывод бегущей строки постоянно:

```
#include <Wire.h>                                   // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                 // Объявляем объект disp для работы с LED матрицей 8x8, указы
                                                    //
void setup(){                                       //
    delay(500);                                     // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                   // Инициируем работу с LED матрицей 8x8.
    disp.print("Hello World");                      // Загружаем текст для бегущей строки (но не выводим её).
    disp.autoScroll(240, 2000);                     // Постоянно прокручиваем бегущую строку со скоростью 240 и в
}                                                   //
                                                    //
void loop(){                                        //
}                                                   //
```

Данный пример выводит ту же строку что и предыдущий пример, но делает это постоянно с задержкой в 2000 миллисекунд между прокрутками строк. В данном примере, сразу после кода `setup` начинает выполняться код `loop` (если бы он там был), так как бегущая строка прокручивается модулем самостоятельно, не используя ресурсов Arduino.

### Вывод изображений:

```
#include <Wire.h>                                   // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                // Подключаем библиотеку для работы с LED матрицей 8x8.
iarduino_I2C_Matrix_8x8 disp(0x09);                 // Объявляем объект disp для работы с LED матрицей 8x8, указы
                                                    // Создаём изображение "смайлик".
byte myImage_1[8] = { 0b00111100,                   //                         ####
                      0b01000010, /* Посмотрите, изображение */  //           #    #
                      0b10100101, /* можно увидеть прямо в   */  //          # #  # #
                      0b10000001, /* коде!!!                 */  //          #      #
                      0b10100101,                   //                      # #  # #
```

```
                      0b10011001,                                //                      #  ##  #
                      0b01000010,                                //                       #    #
                      0b00111100 };                              //                        ####
                                                                 // Создаём изображение "телевизор".
byte myImage_2[8] = { 0b01000100,                                //                       #   #
                      0b00101000,                                //                        # #
                      0b00010000,                                //                         #
                      0b11111111,                                //                      ########
                      0b10000011,                                //                      #     ##
                      0b10000011,                                //                      #     ##
                      0b10000011,                                //                      #     ##
                      0b11111111 };                              //                      ########
void setup(){                                                    //
    delay(500);                                                  // Ждём завершение переходных процессов связанных с подачей п
    disp.begin();                                                // Инициируем работу с LED матрицей 8x8.
}                                                                //
                                                                 //
void loop(){                                                     //
    disp.drawImage(myImage_1); delay(2000);                      // Выводим на дисплей изображение массива myImage_1 и ждём па
    disp.drawImage(myImage_2); delay(2000);                      // Выводим на дисплей изображение массива myImage_2 и ждём па
}                                                                //
```

Данный пример поочерёдно выводит изображения «телевизор» и «смайлик».

Изображения могут выводиться с анимацией появления (как символы). Если использовать несколько светодиодных матриц, то на них можно вывести большое изображение, где каждый дисплей будет отображать свою часть изображения.

### Вывод бегущей строки на несколько дисплеев:

```
#include <Wire.h>                                                // Подключаем библиотеку для работы с аппаратной шиной I2C.
#include <iarduino_I2C_Matrix_8x8.h>                             // Подключаем библиотеку для работы с LED матрицей 8x8.
                                                                 //
```

```
iarduino_I2C_Matrix_8x8 disp[] = {9, 10, 11};                          // Объявляем массив объектов disp для работы с LED матрицами 
uint16_t                k, p;                                          // Объявляем переменные: k-количество дисплеев, p-счетчик сов
String                  ticker = "Интернет магазин iarduino.ru расположен по адресу г.Москва, Леснорядский пер., 18с2, БЦ \"ДМ
                                                                       //
void setup(){                                                          //
    delay(500);                                                        // Ждём завершение переходных процессов связанных с подачей п
    k = sizeof(disp)/sizeof(iarduino_I2C_Matrix_8x8); p=65500;         // Определяем количество дисплеев 'k' и устанавливаем счётчик
    for(int i=0; i<k; i++){                                            // Проходим по всем дисплеям ...
        disp[i].begin();                                               // Инициируем работу с LED матрицей 8x8.
        disp[i].bright(255);                                           // Устанавливаем максимальную яркость.
        disp[i].codingDetect("п");                                     // Выполняем автоопределение кодировки скетча.
        disp[i].print(ticker);                                         // Загружаем текст бегущей строки в матрицу. Текст только заг
    }                                                                  //
}                                                                      //
                                                                       //
void loop(){                                                           //
    p++;                                                               // Увеличиваем счётчик совершённых сдвигов на 1 шаг.
    if(p>=disp[0].getScrollWidth()+disp[0].getCharIndent()-8*k){ // Если строка прокручена полностью (количество сдвигов равно
        p = 0;                                                         // Сбрасываем счётчик совершённых сдвигов.
        delay(1000);                                                   // Ждём 1 сек (задержка на последнем символе строки).
        for(int i=0; i<k; i++){disp[i].clrScr();      }                // Чистим каждый дисплей.
        delay(1000);                                                   // Ждём 1 сек (пауза между прокрутками бегущих строк).
        for(int i=0; i<k; i++){disp[i].scrollPos(i*8);}                // Выводим бегущую строку на дисплей в позиции зависящей от п
        delay(1000);                                                   // Ждём 1 сек (задержка на первом символе строки).
    }                                                                  //
    for(int i=0; i<k; i++){ disp[i].scrollStep(); }                    // Сдвигаем строку каждого дисплея на один шаг.
    delay(50);                                                         // Приостанавливаем выполнение кода, чем больше задержка, тем
}                                                                      //
```

Данный пример выводит бегущую строку «ticker» на несколько дисплеев. Обратите внимание на то, что в примере создаются не несколько объектов библиотеки [iarduino_I2C_Matrix_8x8](), а один массив объектов «disp», что позволяет легко менять количество дисплеев, просто изменив количество элементов массива. Значения с которыми был определён массив объектов являются адресами дисплеев на шине I2C (в примере это адреса 9, 10 и 11). В коде setup() все дисплеи подготавливаются к работе и в каждый дисплей загружается один и тот же текст из строки «ticker». В этом примере нельзя использовать самостоятельную прокрутку функцией autoScroll(), так как она не гарантирует синхронности сдвига бегущих строк на всех дисплеях. По этому в коде loop() строки всех дисплеев постоянно сдвигаются на 1 шаг функцией scrollStep(). А если бегущая строка прокручена до последнего символа (что определяется условием оператора if), то все дисплеи сначала очищаются функцией clrScr() после чего их строки устанавливаются в начальные позиции сдвига функцией scrollPos() и все повторяется. Меняя значения параметров функций delay() данного скетча, можно менять скорость прокрутки, паузу между прокрутками, задержку в начале и конце прокрутки бегущей строки.

В папке examples библиотеки iarduino_I2C_Matrix_8x8 находятся несколько десятков примеров в т.ч.: вывод изображений из области памяти программ, поворот дисплея, изменение яркости, изменение частоты развертки, копирование изображения дисплея в монитор последовательного порта, замена изображений символов, изменение межсимвольного интервала, пошаговый сдвиг бегущей строки, изменение направления, и режима сдвига бегущей строки, и т.д.

## Описание функций библиотеки:

Библиотека iarduino_I2C_Matrix_8x8 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:

```
#include <Wire.h>                    // Подключаем библиотеку Wire. Для работы с Arduino, Piranha или Metro Leonardo, библиоте
#include <iarduino_I2C_Matrix_8x8.h> // Подключаем библиотеку iarduino_I2C_Matrix_8x8.
iarduino_I2C_Matrix_8x8 disp(9);     // Объявляем объект disp для работы с LED матрицей 8x8, указывая её адрес на шине I2C.
```

Если Вам не известен адрес светодиодной матрицы, то объект `disp` можно объявить без указания адреса (библиотека сама найдёт адрес светодиодной матрицы), но на шине I2C должна быть только одна светодиодная матрица (при этом допускается что на шине могут быть другие модули):

```
#include <Wire.h>                    // Подключаем библиотеку Wire. Для работы с Arduino, Piranha или Metro Leonardo, библиоте
```

```
#include <iarduino_I2C_Matrix_8x8.h> // Подключаем библиотеку iarduino_I2C_Matrix_8x8.
iarduino_I2C_Matrix_8x8 disp;        // Объявляем объект disp для работы с LED матрицей 8x8, адрес которой нам неизвестен.
```

### Функция begin();

- Назначение: Функция инициирует работу светодиодной матрицы.
- Синтаксис: **begin();**
- Параметры: Нет.
- Возвращаемые значения:
  - bool: **true** / **false** - флаг наличия светодиодной матрицы на шине I2C.
- Примечание:
  - Функцию необходимо вызвать до обращения к любым другим функциям библиотеки.
  - Функцию достаточно вызвать один раз в коде `setup` .
  - Функцию можно использовать для определения наличия светодиодной матрицы.
- Пример:

```
disp.begin(); // Инициируем работу со светодиодной матрицей.
```

- Пример:

```
if( disp.begin() ){ Serial.println("Модуль найден"   ); } // Инициируем работу с проверкой.
else               { Serial.println("Модуль не найден"); }
```

### Функция changeAddress();

- Назначение: Изменение адреса светодиодной матрицы на шине I2C.
- Синтаксис: **changeAddress(** АДРЕС **);**
- Параметры:
  - **АДРЕС** - Новый адрес для светодиодной матрицы от 1 до 126 (включительно).

- Возвращаемые значения:
  - bool: **true** / **false** - флаг успешного изменения адреса.
- Примечание:
  - Новый адрес светодиодной матрицы записывается в её энергонезависимую память, а значит будет сохранён и после выключения питания.
  - Функция ждёт сохранение адреса и проверяет появление устройства с новым адресом на шине I2C, что занимает некоторое время.
  - После смены адреса можно продолжать работать со светодиодной матрицей без объявления нового объекта.
- Пример:

```
disp.changeAddress( 10 ); // Меняем адрес светодиодной матрицы на 10.
```

- Пример:

```
if( disp.changeAddress(10) ){ Serial.println("Адрес изменён"   ); } // Меняем адрес с проверкой.
else                        { Serial.println("Адрес не изменён"); }
```

### Функция reset();

- Назначение: Перезагрузка светодиодной матрицы.
- Синтаксис **reset();**
- Параметры: Нет.
- Возвращаемые значения:
  - bool: **true** / **false** - флаг успешной перезагрузки светодиодной матрицы.
- Примечание:
  - Функция ждёт завершение перезагрузки, что может занять некоторое время.
  - Изображения символов, изображение матрицы, ширина символов, межсимвольный интервал, яркость, поворот, частота кадров, текст бегущей строки и т.д. будут установлены в значения по умолчанию (как при отключении и включении питания).
- Пример:

```
disp.reset(); // Перезагружаем светодиодную матрицу.
```

### Функция getAddress();

- Назначение: Получение текущего адреса светодиодной матрицы на шине I2C.
- Синтаксис: **getAddress();**
- Параметры: Нет.
- Возвращаемые значения:
    - uint8_t: **АДРЕС** на шине I2C.
- Примечание:
    - Функция может быть полезна когда адрес светодиодной матрицы Вам неизвестен, но был определён библиотекой.
- Пример:

```
uint8_t i = disp.getAddress(); // Получить текущий адрес светодиодной матрицы на шине I2C.
```

### Функция getVersion();

- Назначение: Получение версии прошивки модуля.
- Синтаксис: **getVersion();**
- Параметры: Нет.
- Возвращаемые значения:
    - uint8_t: **ВЕРСИЯ** прошивки модуля.
- Примечание:
    - Чем выше версия тем она поддерживает большее число анимации и имеет незначительные доработки, но все функции описанные на данной странице работают с любой версией прошивки.
- Пример:

```
uint8_t i = disp.getVersion(); // Получить версию прошивки модуля.
```

### Функция getCoding();

- Назначение: Получение текущей кодировки используемой библиотекой.
- Синтаксис: **getCoding();**
- Параметры: Нет.
- Возвращаемые значения:
  - uint8_t **КОДИРОВКА** представленная одним из значений:
    - **X8_TXT_CP866** - библиотека сейчас использует кодировку CP-866.
    - **X8_TXT_UTF8** - библиотека сейчас использует кодировку UTF-8.
    - **X8_TXT_WIN1251** - библиотека сейчас использует кодировку Windows-1251.
- Примечание:
  - Используемая библиотекой кодировка должна совпадать с кодировкой источника символов и текста выводимых на дисплей. Чаще всего текст для вывода берётся из скетча.
  - Если выводятся символы и текст только Латиницы, то кодировка не имеет значения.
- Пример:

```
switch( disp.getCoding() ){ // В зависимости от используемой кодировки далее выводится текст ...
    case X8_TXT_CP866:   Serial.println("CP-866"  ); break;
    case X8_TXT_UTF8:    Serial.println("UTF-8"   ); break;
    case X8_TXT_WIN1251: Serial.println("WIN-1251"); break;
}
```

### Функция setCoding();

- Назначение: Установка кодировки для библиотеки.
- Синтаксис: **setCoding(** КОДИРОВКА **);**
- Параметры:
  - uint8_t **КОДИРОВКА** представленная одним из значений:
    - **X8_TXT_CP866** - требуется использовать кодировку CP-866.
    - **X8_TXT_UTF8** - требуется использовать кодировку UTF-8.
    - **X8_TXT_WIN1251** - требуется использовать кодировку Windows-1251.
- Возвращаемые значения: Нет.

- Примечание:
  - Функция нужна только если на дисплей выводятся текст или символы Кириллицы.
  - Указываемая кодировка должна совпадать с кодировкой источника символов и текста выводимых на дисплей. Чаще всего текст для вывода берётся из скетча.
  - Если выводятся символы и текст только Латиницы, то кодировку можно не задавать.
  - Если Вы не знаете в какой кодировке написан скетч, то используйте функцию автоопределения кодировки скетча `codingDetect("п")`.
- Пример:

```
disp.setCoding( X8_TXT_UTF8 ); // Указать библиотеке использовать кодировку UTF-8.
```

### Функция codingDetect();

- Назначение: Автоопределение кодировки скетча для библиотеки.
- Синтаксис: **codingDetect(** "п" **);**
- Параметры:
  - **"п"** - символ "п" в двойных кавычках.
- Возвращаемые значения: Нет.
- Примечание:
  - Функция работает как `setCoding()` с точно указанной кодировкой скетча.
  - После того как кодировка определена, её можно узнать функцией `getCoding()`.
- Пример:

```
disp.codingDetect("п"); // Определить кодировку скетча и использовать её в библиотеке.
```

### Функция clrScr();

- Назначение: Очистка экрана светодиодной матрицы.
- Синтаксис: **clrScr(** [АНИМАЦИЯ] **);**
- Параметры:

    - uint8_t **АНИМАЦИЯ** очистки дисплея, представлена одним из значений:
        - **X8_RIPPLES** - анимация ряби (хаотичное выключение светодиодов).
        - **X8_DOWN** - анимация сверху вниз (построчное выключение светодиодов).
        - **X8_TOP** - анимация снизу вверх (построчное выключение светодиодов).
- Возвращаемые значения: Нет.
- Примечание:
    - Если функция вызвана без параметра, то экран очищается сразу.
    - Функция ждёт пока дисплей не будет очищен, что занимает некоторое время.
- Пример:

```
disp.clrScr( X8_TOP ); // Очистить эран дисплея построчно снизу вверх.
disp.clrScr();         // Очистить эран дисплея сразу.
```

### Функция fillScr();

- Назначение: Заливка экрана светодиодной матрицы.
- Синтаксис: **fillScr(** [АНИМАЦИЯ] **);**
- Параметры:
    - uint8_t **АНИМАЦИЯ** заливки дисплея, представлена одним из значений:
        - **X8_RIPPLES** - анимация ряби (хаотичное включение светодиодов).
        - **X8_DOWN** - анимация сверху вниз (построчное включение светодиодов).
        - **X8_TOP** - анимация снизу вверх (построчное включение светодиодов).
- Возвращаемые значения: Нет.
- Примечание:
    - Если функция вызвана без параметра, то экран заливается сразу.
    - Функция ждёт пока дисплей не будет залит, что занимает некоторое время.
- Пример:

```
disp.fillScr( X8_TOP ); // Залить эран дисплея построчно снизу вверх.
```

```
disp.fillScr();          // Залить эран дисплея сразу.
```

### Функция invScr();

- Назначение: Инверсия текущего изображения светодиодной матрицы.
- Синтаксис: **invScr();**
- Параметры: Нет.
- Возвращаемые значения: Нет.
- Примечание:
  - Функция включает все светодиоды которые были выключены и выключает те что были включены.
  - Функция инвертирует только текущее изображение светодиодной матрицы.
- Пример:

```
disp.invScr(); // Поменять включённые и выключенные светодиоды местами.
```

### Функция drawImage();

- Назначение: Вывод изображения на экран светодиодной матрицы.
- Синтаксис: **drawImage(** МАССИВ , [ ПАМЯТЬ и/или АНИМАЦИЯ ] **);**
- Параметры:
  - uint8_t **МАССИВ** из 8 элементов с данными выводимого изображения.
  - uint8_t **ПАМЯТЬ** в которой хранится массив изображения, представлена значением:
    - **X8_IMG_RAM** - массив хранится в ОЗУ (по умолчанию).
    - **X8_IMG_ROM** - массив хранится в ПЗУ (памяти программ).
  - uint8_t **АНИМАЦИЯ** появления изображения на экране светодиодной матрицы:
    - **X8_EMPTY_RIPPLES** - появление рябью из пустого фона.
    - **X8_FILLED_RIPPLES** - появление рябью из закрашенного фона.
    - **X8_EMPTY_DOWN** - появление сверху вниз из пустого фона.
    - **X8_FILLED_DOWN** - появление сверху вниз из закрашенного фона.
    - **X8_EMPTY_TOP** - появление снизу вверх из пустого фона.

        - **X8_FILLED_TOP** - появление снизу вверх из закрашенного фона.
- Возвращаемые значения: Нет.
- Примечание:
    - Если анимация не указана, то изображение появляется сразу.
    - Если тип памяти не указан, то считается что массив находится в ОЗУ.
    - Массив указывается первым параметром, а тип памяти и анимация могут быть указаны в любом порядке после массива.
    - Функция ждёт завершение анимации (если она указана), что занимает некоторое время.
- Пример:

```
byte myImage[8] = { 0b00111100,                      //    ####   Массив находится в ОЗУ.
                    0b01000010,                      //   #    #
                    0b10100101,                      //  # #  # #
                    0b10000001,                      //  #      #
                    0b10100101,                      //  # #  # #
                    0b10011001,                      //  #  ##  #
                    0b01000010,                      //   #    #
                    0b00111100 };                    //    ####
disp.drawImage(myImage, X8_EMPTY_TOP, X8_IMG_RAM); // Вывести изображение массива myImage, с анимацией появления снизу вверх и
disp.drawImage(myImage, X8_IMG_RAM, X8_EMPTY_TOP); // Тоже самое, только значения анимации и памяти поменялись местами.
disp.drawImage(myImage, X8_EMPTY_TOP);             // Тоже самое, так как тип памяти по умолчанию X8_IMG_RAM - массив находитс
disp.drawImage(myImage);                           // Вывести изображение массива myImage без анимации.
```

- Пример:

```
const byte myImage[8] PROGMEM = { 0b00111100,      //    ####   Массив находится в памяти программ.
                                  0b01000010,      //   #    #
                                  0b10100101,      //  # #  # #
                                  0b10000001,      //  #      #
                                  0b10100101,      //  # #  # #
                                  0b10011001,      //  #  ##  #
```

```
                                        0b01000010,      //   #    #
                                        0b00111100 };    //    ####
disp.drawImage(myImage, X8_IMG_ROM);                     // Вывести изображение массива myImage, без анимации, массив находится в па
```

### Функция getImage();

- Назначение: Получение изображения из экрана матрицы в массив.
- Синтаксис: **getImage(** МАССИВ **);**
- Параметры:
    - uint8_t **МАССИВ** - в который требуется сохранить изображение с экрана матрицы.
- Возвращаемые значения: Нет.
- Примечание:
    - Действия данной функции противоположны функции `drawImage()` .
- Пример:

```
byte myImage[8];          // Объявляем массив в который требуется сохранить изображение экрана светодиодной матрицы.
disp.getImage( myImage ); // Сохраняем изображение с экрана светодиодной матрицы в массив myImage (как screenshot).
```

### Функция print();

- Назначение: Вывод числа, символа или загрузка текста.
- Синтаксис: **print(** ДАННЫЕ [, АНИМАЦИЯ] **);**
- Параметры:
    - **ДАННЫЕ** - цифра, символ или текст.
    - uint8_t **АНИМАЦИЯ** появления данных на экране светодиодной матрицы:
        - **X8_EMPTY_RIPPLES** - появление рябью из пустого фона.
        - **X8_FILLED_RIPPLES** - появление рябью из закрашенного фона.
        - **X8_EMPTY_DOWN** - появление сверху вниз из пустого фона.
        - **X8_FILLED_DOWN** - появление сверху вниз из закрашенного фона
        - **X8_EMPTY_TOP** - появление снизу вверх из пустого фона

    - **X8_FILLED_TOP** - появление снизу вверх из закрашенного фона
- Возвращаемые значения: Нет.
- Примечание:
  - Если указан символ или цифра, то функция выведет её на экран светодиодной матрицы.
  - Если указан текст, то функция лишь сохранит его в модуль, а для вывода текста необходимо запустить бегущую строку функцией `autoScroll()` или сдвинуть строку функцией `scrollPos()` , или `scrollStep()` .
  - Функция ждёт завершение анимации (если она указана), что занимает некоторое время
- Пример:

```
disp.print( 5 );         // Вывести на экран цифру 5.
disp.print( char(65) ); // Вывести на экран символ 'A' (символ с кодом 65).
disp.print( 'F' );       // Вывести на экран символ 'F'.
disp.print( "F" );       // Вывести на экран символ "F".
disp.print( "Hello" );  // Сохранить текст для бегущей строки.
```

### Функция autoScroll();

- Назначение: Автопрокрутка бегущей строки.
- Синтаксис: **autoScroll(** СКОРОСТЬ [, ПАУЗА] **);**
- Параметры:
  - uint8_t **СКОРОСТЬ** прокрутки бегущей строки. Указывается числом от 1 (минимальная) до 255 (максимальная). Если указать 0 то бегущая строка остановится.
  - uint16_t **ПАУЗА** между прокрутками бегущих строк в миллисекундах. Указывается числом от 100 (0,1 сек) до 25500 (25,5 сек). Если указать 0 или не указывать паузу, то бегущая строка будет прокручена однократно (без повторов).
- Возвращаемые значения: Нет.
- Примечание:
  - От скорости прокрутки бегущей строки зависит задержка между сдвигом бегущей строки на 1 шаг, она равна (256 - СКОРОСТЬ) / 100 секунд.
  - Указанное значение паузы между прокрутками бегущих строк округляется в меньшую сторону до десятых долей секунд (пример, 230 мс = 0,2 сек).

- Пример:

```
disp.autoScroll( 200 );       // Прокрутить бегущую строку однократно со скоростью 200.
disp.autoScroll( 250, 1000 ); // Прокручивать бегущую строку постоянно со скоростью 250 и паузой между прокрутками бегущих стр
```

### Функция scrollPos();

- Назначение: Установка позиции с которой требуется вывести строку на экран.
- Синтаксис: **scrollPos(** ПОЗИЦИЯ **);**
- Параметры:
  - uint16_t **ПОЗИЦИЯ** с которой требуется вывести строку на экран. Указывается числом от 0 до 65535.
- Возвращаемые значения: Нет.
- Примечание:
  - Позиция это номер пикселя или символа строки от её начала или конца, зависит от значений указанных функциями `scrollDir()` и `scrollMod()` . По умолчанию позиция определяет номер пикселя от начала строки.
  - Если текст строки был загружен, но строка ещё не выведена на экран, то строка появится на экране начиная с указанной позиции без автопрокрутки.
  - Если строка выводилась автопрокруткой, то строка продолжит прокручиваться начиная с указанной позиции.
- Пример:

```
disp.scrollPos( 10 ); // Вывести на экран строку с позиции 10 (с позиции 10 шага сдвига строки).
```

### Функция scrollDir();

- Назначение: Установка направления сдвига бегущей строки.
- Синтаксис: **scrollDir(** ФЛАГ **);**
- Параметры:
  - bool **true / false** - флаг обратного направления сдвига бегущей строки.
- Возвращаемые значения: Нет.

- Примечание:
  - Если вызвать функцию с параметром true, то позиция будет отсчитываться от конца строки, а её сдвиг будет осуществляться от последнего символа к первому.
  - По умолчанию (если не вызывать функцию) используется прямое направление сдвига.
  - Указанное направление не сохраняется после отключения питания.
- Пример:

```
disp.scrollDir( false ); // Сдвигать строку в прямом   направлении (от первого    символа к последнему).
disp.scrollDir( true  ); // Сдвигать строку в обратном направлении (от последнего символа к первому   ).
```

### Функция scrollMod();

- Назначение: Установка режима сдвига бегущей строки.
- Синтаксис: **scrollMod(** ФЛАГ **);**
- Параметры:
  - bool **true / false** - флаг посимвольного сдвига бегущей строки.
- Возвращаемые значения: Нет.
- Примечание:
  - Если вызвать функцию с параметром true, то за один шаг строка будет сдвигаться не на 1 пиксель, а на один символ.
  - По умолчанию (если не вызывать функцию) используется попиксельный режим сдвига.
  - В режиме посимвольного сдвига на экране будет отображаться только один символ из заданной позиции строки (части рядом стоящих символов отображаться не будут).
  - Указанный режим не сохраняется после отключения питания.
- Пример:

```
disp.scrollMod( false ); // Использовать попиксельный режим сдвига бегущей строки (строка прокручивается влево или вправо).
disp.scrollMod( true  ); // Использовать посимвольный режим сдвига бегущей строки (все символы строки выводятся в одно и то же
```

### Функция scrollStep();

- Назначение: Сдвиг бегущей строки на один шаг.
- Синтаксис: **scrollStep();**
- Параметры: Нет.
- Возвращаемые значения: Нет.
- Примечание:
  - Каждый вызов данной функции увеличивает позицию сдвига строки на один шаг и выводит её на экран светодиодной матрицы, как `scrollPos( ПОЗИЦИЯ++ );` .
  - Позиция увеличивается до значения 65535 после чего произойдёт сброс.
- Пример:

```
disp.scrollStep(); // Увеличить позицию сдвига строки на 1 шаг и вывести её на экран светодиодной матрицы.
```

### Функция setTimeIdleFirst();

- Назначение: Время простоя на первом символе при автопрокрутке бегущей строки.
- Синтаксис: **setTimeIdleFirst(** ВРЕМЯ **);**
- Параметры:
  - uint16_t **ВРЕМЯ** простоя на первом символе в миллисекундах. Указывается числом от 0 до 2550 (2,55 сек).
- Возвращаемые значения: Нет.
- Примечание:
  - Указанное значение времени округляется в меньшую сторону до сотых долей секунд (пример, 1234 мс = 1,23 сек).
  - При автопрокрутке бегущей строки время простоя на первом и последнем символе отличается от задержки между остальными шагами прокрутки. По умолчанию первый и последний символ текста бегущей строки отображается по секунде. Данная функция позволяет изменить время отображения первого символа.
  - Указанное значение не сохраняется после отключения питания.
- Пример:

```
disp.setTimeIdleFirst( 1500 ); // Задержаться на первом символе текста в течении 1,5 сек.
```

### Функция setTimeIdleLast();

- Назначение: Время простоя на последнем символе при автопрокрутке бегущей строки.
- Синтаксис: **setTimeIdleLast(** ВРЕМЯ **);**
- Параметры:
    - uint16_t **ВРЕМЯ** простоя на последнем символе в миллисекундах. Указывается числом от 0 до 2550 (2,55 сек).
- Возвращаемые значения: Нет.
- Примечание:
    - Указанное значение времени округляется в меньшую сторону до сотых долей секунд (пример, 1234 мс = 1,23 сек).
    - При автопрокрутке бегущей строки время простоя на первом и последнем символе отличается от задержки между остальными шагами прокрутки. По умолчанию первый и последний символ текста бегущей строки отображается по секунде. Данная функция позволяет изменить время отображения последнего символа.
    - Указанное значение не сохраняется после отключения питания.
- Пример:

```
disp.setTimeIdleLast( 1500 ); // Задержаться на последнем символе текста в течении 1,5 сек.
```

### Функция getScroolLen();

- Назначение: Получение количества символов бегущей строки загруженных в модуль.
- Синтаксис: **getScroolLen();**
- Параметры: Нет.
- Возвращаемые значения:
    - uint16_t **КОЛИЧЕСТВО** символов в бегущей строке загруженной в модуль.
- Примечание:
    - Функция возвращает количество символов загруженных в матрицу, а не количество байт занимаемое бегущей строкой. Например если в модуль загружена строка "Привет", то функция вернёт 6 вне зависимости от кодировки скетча.
- Пример:

```
uint16_t i = disp.getScroolLen(); // Определить количество символов бегущей строки загруженных в модуль.
```

---

### Функция getScroolWidth();

- Назначение: Получение ширины бегущей строки в пикселях.
- Синтаксис: **getScroolWidth();**
- Параметры: Нет.
- Возвращаемые значения:
  - uint16_t **КОЛИЧЕСТВО** пикселей в бегущей строке загруженной в модуль.
- Примечание:
  - Функция может понадобиться если используется пошаговый сдвиг бегущей строки с помощью scrollStep().
- Пример:

```
uint16_t i = disp.getScroolWidth(); // Определить количество пикселей в которое умещается вся бегущая строка.
```

### Функция angle();

- Назначение: Изменение угла поворота экрана светодиодной матрицы.
- Синтаксис: **angle(** УГОЛ **);**
- Параметры:
  - uint16_t **УГОЛ** поворота дисплея в градусах. Допускаются значения 0, 90, 180, 270.
- Возвращаемые значения: Нет.
- Примечание:
  - Текущее изображение экрана, а так же вновь выводимые данные будут повернуты на указанный угол.
  - Указанный угол не сохраняется после отключения питания.
- Пример:

```
disp.angle( 90 ); // Повернуть экран на 90°.
```

### Функция fps();

- Назначение: Установка частоты обновления экрана светодиодной матрицы.

- Синтаксис: **fps(** ЧАСТОТА **);**
- Параметры:
  - uint8_t **ЧАСТОТА** обновления экрана в fps (кадры в секунду). Указывается числом от 0 до 255.
- Возвращаемые значения: Нет.
- Примечание:
  - По умолчанию используется частота 100 кадров в секунду.
  - При частоте ниже 50 кадров в секунду будет заметно мерцание изображения.
  - Изменение частоты обновления экрана может быть полезно при съемке показаний светодиодной матрицы на камеру.
  - Указанная частота не сохраняется после отключения питания.
- Пример:

```
disp.fps( 60 ); // Установить частоту обновления экрана в 60 кадров в секунду.
```

### Функция bright();

- Назначение: Установка яркости экрана светодиодной матрицы.
- Синтаксис: **bright(** ЯРКОСТЬ **);**
- Параметры:
  - uint8_t **ЯРКОСТЬ** свечения светодиодной матрицы. Указывается числом от 1 (минимальная) до 255 (максимальная). Если указать 0, то все светодиоды будут постоянно выключены.
- Возвращаемые значения: Нет.
- Примечание:
  - Установленная яркость не сохраняется после отключения питания.
  - По умолчанию используется яркость 192 (75%).
- Пример:

```
disp.bright( 255 ); // Установить максимальную яркость экрана светодиодной матрицы.
```

### Функция changeChar();

- Назначение: Изменение изображения символа.
- Синтаксис: **changeChar(** КОД **);**
- Параметры:
  - uint8_t **КОД** символа в таблице CP-866, изображение которого требуется изменить. Код символа указывается числом от 1 до 255.
- Возвращаемые значения:
- uint8_t ШИРИНА изображений символов в пикселях (колонках).
- Примечание:
  - В качестве нового изображения символа берётся то, что на момент вызова функции находится на экране светодиодной матрицы.
  - По умолчанию ширина символа равна 5 пикселям (колонкам), а изображение для символа берётся с отступом на 2 пикселя (колонки) от левого края экрана. Ширину символа и отступ можно изменить функциями `setCharWidth()` и `setCharIndent()`.
  - Функция ждёт пока изображение символа не сохранится, что занимает некоторое время.
  - Символ код которого был указан будет отображаться на экране по новому.
  - Все изменения сбрасываются после отключения питания.
- Пример:

```
// Создаём новое изображение для символа 'Б':      //   ________
byte myImage[8] = { 0b00000000,                    //  |        |
                    0b00111100,                    //  |  ####  |
                    0b00100000,                    //  |  #     |
                    0b00100000,                    //  |  #     |
                    0b00111100,                    //  |  ####  |
                    0b00100010,                    //  |  #   # |
                    0b00100010,                    //  |  #   # |
                    0b00111100 };                  //  |  ####  |
// Выводим изображение и сохраняем его для символа //   --------
disp.drawImage(myImage);                           // Выводим изображение на экран светодиодной матрицы.
disp.changeChar(129);                              // Меняем изображение символа 'Б' (в таблице CP-866 символ 'Б' имеет код 12
// Теперь символ 'Б' как в тексте бегущей строки, так и отдельно, будет отображаться так, как мы его нарисовали.
// Так можно изменить хоть все 255 символов в памяти модуля, но все изменения сбросятся при отключении питания.
```

### Функция setCharWidth();

- Назначение: Изменение ширины изображений всех символов.
- Синтаксис: **setCharWidth(** ШИРИНА **);**
- Параметры:
  - uint8_t **ШИРИНА** символов в пикселях (колонках). Указывается от 3 до 7 (включительно).
- Возвращаемые значения: Нет.
- Примечание:
  - По умолчанию все символы имеют ширину 5 пикселей (колонок).
  - После смены ширины символов их изображения исказятся (станут нечитаемыми), а значит изображения всех символов потребуется ввести заново функцией `changeChar()` .
  - Установленная ширина не сохраняется после отключения питания.
- Пример:

```
disp.setCharWidth( 7 ); // Установить ширину для всех символов в 7 пикселей (колонок).
```

### Функция getCharWidth();

- Назначение: Получение ширины символов.
- Синтаксис: **getCharWidth();**
- Параметры: Нет.
- Возвращаемые значения:
  - uint8_t **ШИРИНА** изображений символов в пикселях (колонках).
- Примечание:
  - По умолчанию все символы имеют ширину 5 пикселей (колонок).
  - Зная ширину символов, количество символов в тексте бегущей строки, межсимвольный интервал и скорость бегущей строки, можно приблизительно рассчитать время затрачиваемое на автопрокрутку всей бегущей строки.
- Пример:

```
uint i = disp.getCharWidth(); // Получить ширину изображений символов.
```

### Функция setCharInterval();

- Назначение: Изменение межсимвольного интервала.
- Синтаксис: **setCharInterval(** ИНТЕРВАЛ **);**
- Параметры:
  - uint8_t **ИНТЕРВАЛ** между символами в тексте бегущей строки. Указывается от 0 до 255.
- Возвращаемые значения: Нет.
- Примечание:
  - По умолчанию используется интервал между символами в 1 пиксель (колонку).
  - Увеличение интервала не влияет на изображения символов.
  - Установленный интервал не сохраняется после отключения питания.
- Пример:

```
disp.setCharInterval( 3 ); // Установить межсимвольный интервал в 3 пикселя (колонки).
```

### Функция getCharInterval();

- Назначение: Получение межсимвольного интервала.
- Синтаксис: **getCharInterval();**
- Параметры: Нет.
- Возвращаемые значения:
  - uint8_t **ИНТЕРВАЛ** между символами в тексте бегущей строки.
- Примечание:
  - По умолчанию используется интервал между символами в 1 пиксель (колонку).
  - Зная межсимвольный интервал, ширину символов, количество символов в тексте бегущей строки и скорость бегущей строки, можно приблизительно рассчитать время затрачиваемое на автопрокрутку всей бегущей строки.
- Пример:

```
uint i = disp.getCharInterval(); // Получить межсимвольный интервал.
```

### Функция setCharIndent();

- Назначение: Изменение отступа от левого края экрана до символа.
- Синтаксис: **setCharIndent(** ОТСТУП **);**
- Параметры:
  - uint8_t **ОТСТУП** от левого края экрана до символа в пикселях (колонках). Указывается от 0 до 5 (включительно).
- Возвращаемые значения: Нет.
- Примечание:
  - По умолчанию используется отступ в 2 пикселя (колонки) от левого края экрана.
  - Отступ используется для вывода одиночного символа, для вывода первого символа текста бегущей строки в позиции 0 и при чтении изображения с экрана светодиодной матрицы для замены им изображения символа.
  - Отступ + ширина символа не могу превышать ширину матрицы (8 пикселей).
  - Установленный отступ не сохраняется после отключения питания.
- Пример:

```
disp.setCharIndent( 1 ); // Установить отступ в 1 пиксель (колонку).
```

### Функция getCharIndent();

- Назначение: Получение отступа от левого края экрана до символа.
- Синтаксис: **getCharIndent();**
- Параметры: Нет.
- Возвращаемые значения:
  - uint8_t **ОТСТУП** от левого края экрана до символа в пикселях (колонках)
- Примечание:
  - По умолчанию используется отступ в 2 пикселя (колонки).
  - Зная отступ можно с какого места экрана будут отрисовываться символы.
- Пример:

```
uint i = disp.getCharIndent(); // Получить отступ от левого края экрана до символа.
```

## Применение:

Trema-модуль I2C LED матрица 8x8 можно использовать в любых проектах где требуется вывод данных на монохромный дисплей размером 8x8 пикселей. В качестве данных для вывода могут использоваться изображения, символы или бегущая строка.

Совместив несколько модулей можно выводить более крупные изображения, или пошагово выводить текст бегущей строки на модули составленные в одну линию.

## Ссылки:

- Библиотека iarduino_I2C_Matrix_8x8.
- Расширенные возможности библиотек iarduino для шины I2C.
- Wiki - Установка библиотек в Arduino IDE.