# LEDMatrix\_22 Manual

Contents

[LEDMatrix\_22 Manual 1](#_Toc93690899)

[Overview 2](#_Toc93690900)

[Examples 3](#_Toc93690901)

[Getting Started – start out simple! 6](#_Toc93690902)

[New Features 8](#_Toc93690903)

[24 bit full color bitmap images (sprites) 8](#_Toc93690904)

[Creating a Lookup Table 8](#_Toc93690905)

[Irregular led arrays 10](#_Toc93690906)

[Irregular Array Example 10](#_Toc93690907)

[LEDMatrix Functions 13](#_Toc93690908)

[Files in the LEDMatrix\_22 Library 18](#_Toc93690909)

[Set up Steps 18](#_Toc93690910)

[Section 1 20](#_Toc93690911)

[Section 2 - tiles/blocks in the matrix panel 22](#_Toc93690912)

[Your Sketch and setup() 25](#_Toc93690913)

[Dr Oldies LED Extender 26](#_Toc93690914)

[Advanced Topics 29](#_Toc93690915)

[Using constant variables instead of #defines 30](#_Toc93690916)

[FastLED function list (partial) 31](#_Toc93690917)

[List of Functions 36](#_Toc93690918)

[Errors and Possible Fixes 39](#_Toc93690919)

[Multiple led stripss not than in a Matrix 39](#_Toc93690920)

[Multiplexing to increase matrix panel size 41](#_Toc93690921)

[Fast\_LEDS multiple “controllers” 41](#_Toc93690922)

[LEDMatrix\_22 multiple “controllers” 41](#_Toc93690923)

[Examples 43](#_Toc93690924)

[LEDMatrix\_22.ino 44](#_Toc93690925)

## Now works with ESP32, Teensy 3.5/3.6, Teensy 4.0/4.1 and other 32 bit (recommended) MCUs using FastLED 3.5.0+

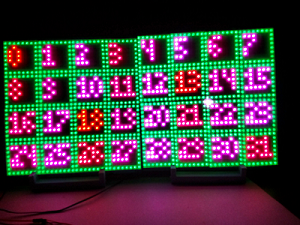
## Overview

*If you just want to jump directly into LEDMatrix\_22 set up for your LED project, go to Section “5. Set up using configuration\_22.h”*

The LEDMatrix\_22 library is a **medium weight**, two-dimensional graphics library for led matrices/arrays using FastLED. This is an update and enhancement of previous LEDMatrix libraries. In addition to drawing shapes, and text, this library expands image and animation handling to 24 bit color. Larger displays can be formed using tiles of led strips/panels - to build one big matrix. New and enhanced feature include:

* 2-wire leds: APA102, Adafruit’s DotStar, SK9822, HD1701, LPD8806, SM16716, WS2801.
* 1-wire Neomatrix led stripss and arrays (WS2812, etc.).
* 24 bit color image and basic sprite display (no edge detection).
* Block save/restore to restore backgrounds.
* Transparent sprite drawing. (No boundary detection)
* Text, image, and sprite rotation in 90 deg increments.
* Option to read a XYTable\_LookUp.h to replace slower, complex coordinate calculations.
* An Arduino sketch utility to create the lookup table is included.
  + The lookup table option allows for irregularly shaped led physical layouts.
* A report generator to confirm proper led array mapping configuration.
* Method names now use Adafruit\_GFX naming (for similar methods).

Figure 0. Panel of 1024 leds of 16 8x8 tiles

*Dr Oldies LED Extender shields*

* Along with a soon-to-be-release *Dr Oldies leds Extender shields*, 1-wire and 2-wire Led strips can be wired in multiple Banks to reduce Led strip length. The extender:
  + Eliminates led “sparkle” and flashing (a frustrating problem with 2-wire leds types) by reducing the number of leds needed in series. - With proper power supplies, and wiring, the Extender can support up to at least 256 leds per strip. That’s 64,000 leds! A 32 bit MCU such as the Teensy 4.0/4.1 is required.

Figure 0. Panel of 1024 leds of 16 8x8 tiles

* + Multiplexes controller wiring, reducing pin count 1-wire or 2-wire Led strips. Up to 16 Led strips/panels with only 8 wires! (4 for 2 DATA + 2 CLOCK pins, and up to 4 more “enable” pins to switch between Banks of Led strips.
  + Voltage step up from 3.3v to 5v.
  + Isolates the MCU from the led wiring.

### Examples

\*\*There are numerous examples in the examples folder.

1. Types of LED configurations

#### Types of led configurations

#### Overview

The following led configurations are under user control. All parameters are contained in Configuration\_21.h in the library folder. This config file is clearly organized with numerous comments. Supported led configurations include:

* A single matrix (all leds flow in 1 long string in any combination of rows and columns.
* led “tiles” in panels such as 4x4 or 8x8 arrays in various order.
* Multiple strings (up to 4 strings) as strings or in matrices. These can be part of a larger Panel.
* With an inexpensive hardware shield up to 16 strings can be combined into very large panels. These shield also isolate the MCU and step the data voltage up to 5 volts. With only 7 or 8 MCU pins!

##### Single Matrix

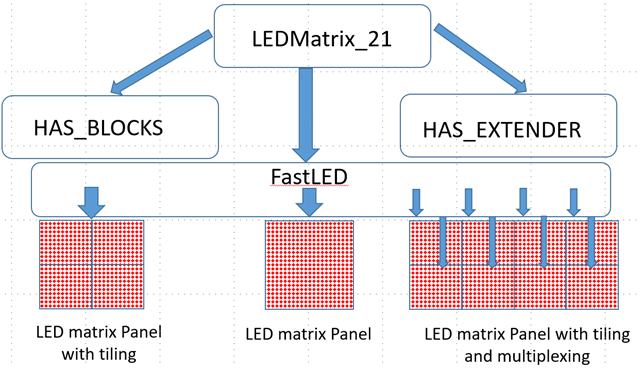
In this arrangement, one long led strips is cut into multiple rows to create an x,y array. As with previous LEDMatrix versions, the strips can be arranged into rows a zigzag, or left-2-right/right-2-left patterns. In this configuration you are limited to the length of the strip before sparkles/breakup occurs. SPI pins can be used to meet the demand of refreshing long data strings. This approach uses one FastLED Controller.

Figure 1 LEDMatrix modules for simple and tiles arrays as well as extending to large arrays and higher data/frame rates.

##### Tile Matrix

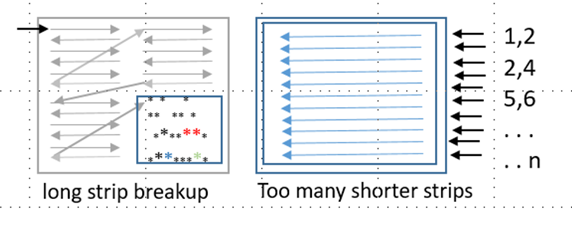
Tiling (also called blocks) organizes the one long led strips into smaller tiles. Popular 8x8 led panels are an example of this. The 8x8 panels can be strung together. LEDMatrix\_22 can handle any arrangement of these in normal or zigzag leds in tiles, and normal or zigzag tiles in the larger matrix panel. This approach uses one FastLED Controller.

##### FastLED Multiple-controllers (*new*)

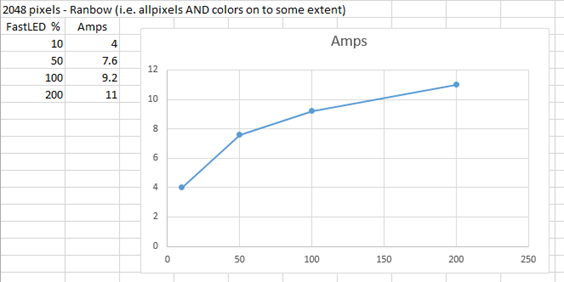
LEDMatrix\_22 uses FastLED’s multiple Controllers to simultaneously support up to 4 separate led stripss. This method requires a lot of MCU pins. 8 for 2-wire leds, and 4 for 1-wire leds.

##### Multiplexing with Multiple-controllers (*new*)

For further expansion LEDMatrix\_22 supports a small hardware shield PCB (”LED Extender”) and uses FastLED’s multiple Controllers to drive shorter strips or tiles rather than one long strip. For example, to drive a 32x32 matrix of 1024 leds with one long string, data send rates, brightness, and frames-per-second (fps) to avoid color breakdown as to be unusable.

Now, using the Dr. Oldies LED Extender (more detail in the Extender section) we can break up the 32x32 matrix and insert a second DATA/CLOCK line for leds 512-1024. No more sparkle even with higher data/brightness/fps rates. By breaking the your design into still more but shorter data lines (for example 256, 128, or 64 leds segments) allows even high data rates.

#### Using multiple led stripss together but not as a matrix or panel?

By configuring LEDMatrix as 1 (long) row single, or up to 4 led strings can be used. With the LED Extender shields to control up to 16 led stripss with 8 controller pins (the same as above). This is done by “Banking” led strings across up to 4 Extender shields, each supporting 4 led stripss. These Banks of leds can be turned on and off under Sketch control.

##### Compatibility with SmartMatrix and its various library combinations

LEDMatrix\_22 is designed as a medium weight library, with just enough features to support your project. Beyond FastLED (and the led types FastLED supports), this library is not intended to be combined or layered with other libraries.

#### Limitations in this library version

SmartMatrix has transitioned into a software + hardware solution primarily for HUB75 led panels. LEDMatrix\_22 currently does not support SmartMatrix or HUB75 hardware.

## Getting Started – start out simple!

Configuring LEDMatrix\_22 for a simple project with an array that is one long led strips (even if it is assembled in a number of row to make a matrix. Set the led type, the number of leds horizontal and vertical, and the data/clock pins at the top of Configuration\_22.h. Don’t forget to set HAS\_BLOCKS and HAS\_EXTENDER as false. Configuring LEDMatrix\_22 for a project using 4 x 4 or 8 x 8 commercials led panels, multiple strips, or irregular shaped layout can be a bit confusing. Make sure to read through this manual or the wiki before you start. I added many comments in the headers and the example code that should help.

1. Other libraries, Documentation and Credits

#### Other libraries and Documentation

The previous version of LEDMatrix include descriptions of the basic graphics draw routines (circle, square, triangle, etc.) these are all still supported in LEDMatrix\_22, with a select few renamed to reflect Adafruit\_GFX naming conventions.

LEDMatrix by VikingGod [Jürgen Skrotzky] at: <https://github.com/Jorgen-VikingGod/LEDMatrix> with additional descriptions here: <https://jorgen-vikinggod.github.io/LEDMatrix>

There is also a wiki for an even earlier version of LEDMatrix by Aaron Liddiment at: <https://github.com/AaronLiddiment/LEDMatrix/wiki>

FastLED by Garcia is a well liked interface for all the leds supported by LEDMatrix\_22: FastLED Documentation. While this is for version 3.1, it is the most complete description of FastLED is here: <http://FastLED.io/docs/3.1/>

There is also a FastLED Wiki here: <https://github.com/FastLED/FastLED/wiki>

How to use Excel to Animate leds! Arduino + WS2812 leds by Kevin Darrah: <https://www.youtube.com/watch?v=A_S3LAUQHwU>

Sprites – here are a few sprites to play with: <https://spritedatabase.net/download>

New Features

## New Features

### 24 bit full color bitmap images (sprites)

Draw an image at the specified(x, y) position from the 24 bit color bitmap (in CRGB::color).

There are several versions:

drawBitmap24(x, y, bitmapName, w, h, bg)

The bitmap must be PROGMEM memory in this call and drawing the bitmap’s colors. A background color of 0x000000 (black) = transparent, leaving the current color, otherwise color of bg. This is fastest.

A more flexible version allow you to use a zigzag pattern or not, set background of Black = transparent, and recall from PROGMEM or not

drawBitmap24(x, y, bitmapName, w, h, progMem, zigzag, bg)

Draw a bitmap image at the specified(x, y) position from the (bitmap must be PROGMEM memory) using the bitmap’s foreground colors. A bg color of 0x000000 (black) = transparent, otherwise color of bg. if all rows are left to right, zigzag = 0. If odd rows are reversed, zigzag = 1; this is the slower of the two.

The array structure for 24 bit images is 8 bits per color as 0xrrGGbb. So the value size is const long.

const long PROGMEM DigDug01[] = {  
0x000000,  
.  
.  
0xffffff  
};

An easy way to store the size of the bitmap is to include the size within the file as in the clip below. Since each bitmap is “#included” in the sketch, the #defines will be available when needed.

#define DIGDUG01\_W 16  
#define DIGDUG01\_H 16  
#define DIGDUG01\_Z false //zigzag  
#define DIGDUG01\_P true //progmem

### Creating a Lookup Table

A lookup table is an array indexing the leds in the strip by the x and y position. This is much faster than calculating the physical layout in code top-down, zigzag, or other directives. While this takes a bit of effort, you only need to do this once for each project. Use the table in this library as an example. It is for a 32x32 led matrix made up of 16 8x8 “cells.” Each cell is 8x8 with a zigzag pattern, and the cells are arrays in 4 rows from left to right (not zigzag).

There are a number of ways to create your table. • Use an excel spreadsheet listing each pixel’s strip number in an x,y cell order. You can export this as a CSV comma delimited file. Now rename it to “XYTable\_LookUp.h” and place it in the library folder. • Check out the FastLED XY Map Generator - web based generator by Garrett Mace (macetech.com), at: https://macetech.github.io/FastLED-XY-Map-Generator/ as a great example and array generator for simple matrices

##### XYTable\_LookUp Code header file

\*\* The XYTable\_LookUp generator is in it’s own folder  XYTable\_LookUpThis will create lookup tables in the format shown in the example below\*\*

This Arduino sketch will create a lookup table for led projects instead of writing and using mapping code. It uses LEDMatrix definitions (ex: HORIZONTAL\_ZIGZAG\_MATRIX) to define the led mapping.

The led mapping apps I have found all have shortcoming on the size or layout of the matrix. Especially for blocks or cells within the matrix like the popular 8x8 blocks. This sketch includes:

#### Up to 32k leds

Small to very large matrices – laid out in any direction with or w/o zigzag Matrix can be made of blocks (cells) of any size that of any size– laid out in any direction with or w/o zigzag in the block and block layout within the matrix. Produces a report on the Serial Terminal of the specified configuration and the resulting mapping array. Simply cut and paste into your header file. Arduino code is in small single purpose functions that are easy to modify

The look up table is only 3 lines of code added to the method mXY in the inLEDMatrix\_22.h file. This intercepts the coordinate x, y lookup request to return the table entry. It leaves the rest of the code intact rather than replacing it at compile time. The XYTable\_LookUp.h table looks like this for an 8x8 led array with every even line zigzagging:

/\* XYTable\_yx.h  
This table is laid out in X=horizontal in each row and Y=vertical rows.  
 Addressing is: XYTable[y][x] NOTx,y  
\*/  
const uint16\_t PROGMEM XYTable[][8] = {  
  
0,1,2,3,4,5,6,7,  
15,14,13,12,11,10,9,8,  
16,17,18,19,20,21,22,23,   
…  
63,62,61,60,59,58,57,56  
};

**CAUTION:** While the table numbers are in x, y (x = across the row, and y = down the rows). The code handles this, but if you access the Table[][] directly, addressing is: XYTable[y][x] NOTx,y.

#### Other Look up Table apps

How to use Excel to Animate leds! Arduino + WS2812 leds by Kevin Darrah: <https://www.youtube.com/watch?v=A_S3LAUQHwU>

### Irregular led arrays

Another advantage of using the XYTable look up is mapping irregular led arrays. Set THE PIXEL INDEX IN THE TABLE for the x, y coordinates (pixels) that are not physically present to a value larger than the number physical pixels.

**All library functions use drawPixel() for update the display matrix** **drawPixel() will test for this “not available” (i.e. out of bounds) pixel and ignore it**

Check out the FastLED XY Map Generator - web based generator by Garrett Mace (macetech.com), at: <https://macetech.github.io/FastLED-XY-Map-Generator>. This is a great example and array generator for simple irregular matrices.

### Irregular Array Example

For this table: Let’s say you are making a face mask with leds all over the mask, but no leds for the eyes, nose, and mouth openings. Also assume the longest row and column is 16x16 but there are missing leds at various spots in the matrix.

The LookUp table would look like this: Table size is 16x16, or VIRTUAL 256 elements (0-255) Physical number of leds = 102 (0-101) with unused leds skipped. So: 1. x, y must still work for the VIRTUAL size ex: 256 2. Fill and show functions must use only the Physical number of leds ex: 102

So: Make NUM\_LEDS = 256, and WIDTH and HEIGHT = 16. Any missing led in the Lookup Table are set to any number > 255 (or any number larger than the last actual leds).

**How it Works:** In this case, when your Sketch draws to x,y [0][0] there is no led to display the color. The array index is 256 (past the end of the led stripss) so no color is stored. This repeats until x, y [6][0], which is your 1st real leds in the led strips. The color is stored.

/\* XYTable\_yx.h  
  
This table is laid out in X=horizontal in each row and Y=vertical rows.  
 Addressing is: XYTable[y][x] NOTx,y  
  
\*/  
const uint16\_t PROGMEM XYTable[][32] = {  
  
 256, 256, 256, 256, 256, 256, 0, 1, 2, 3, 256, 256, 256, 256, 256, 256,  
 256, 256, 256, 256, 256, 9, 8, 7, 6, 5, 4, 256, 256, 256, 256, 256,  
 256, 256, 256, 256, 10, 11, 12, 13, 14, 15, 16, 17, 256, 256, 256, 256,  
 256, 256, 247, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 256, 256, 265,  
 256, 256, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 256, 256,  
 256, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 256,  
 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,  
 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70,  
 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101,  
 117, 116, 115, 114, 113, 112, 111, 110, 109, 108, 107, 106, 105, 104, 103, 102,  
 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133,  
 149, 148, 147, 146, 145, 144, 143, 142, 141, 140, 139, 138, 137, 136, 135, 134,  
 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,  
 181, 180, 179, 178, 177, 176, 175, 174, 173, 172, 171, 170, 169, 168, 167, 166,  
 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197,  
 213, 212, 211, 210, 209, 208, 207, 206, 205, 204, 203, 202, 201, 200, 199, 198  
 };

#### DEBUGGING REPORT - sent to the serial terminal

LEDMatrix includes a reporter to generate a detaileds description of your configuration. This is ideal for debugging your LEDMatrix configuration.

An example of this is in the **LEDMatrix.ino** example Sketch.   ——————————– Sample Report ——————————–

Port open ====== Reporting Enabled======

Use this report to check that all your definitions are correct (if your panel is not working correctly it is likely a parameter is incorrect)

======= Report ======  
  
NUM\_LEDS = 1024 leds total in all strings  
MATRIX\_WIDTH = 32 leds across entire matrix panel  
MATRIX\_HEIGHT = 32 leds up/down entire matrix panel  
Direction 1st row of matrix panel leds = LEFT\_2\_RIGHT / TOP\_DOWN  
HAS\_BLOCKS = true  
 LEDS\_IN\_TILE = HORIZONTAL\_ZIGZAG\_MATRIX (flow of leds inside each tile/block)  
 MATRIX\_TILE\_WIDTH = 8  
 MATRIX\_TILE\_HEIGHT = 8  
 MATRIX\_TILE\_H = 4  
 MATRIX\_TILE\_V = 4  
 TILES\_IN\_MATRIX = HORIZONTAL\_BLOCKS (flow of tiles/blocks thru the matrix panel)  
  
Your panel directions are: LEFT\_2\_RIGHT / TOP\_DOWN  
>> The following tables are L/R T/B. Visually flip as needed!  
  
Map of leds in your tiles  
 0 1 2 3 4 5 6 7  
 8 9 10 11 12 13 14 15  
 16 17 18 19 20 21 22 23  
 24 25 26 27 28 29 30 31  
 32 33 34 35 36 37 38 39  
 40 41 42 43 44 45 46 47  
 48 49 50 51 52 53 54 55  
 56 57 58 59 60 61 62 63  
  
Map of tiles in your matrix panel  
 0 1 2 3  
 4 5 6 7  
 8 9 10 11  
 12 13 14 15  
  
Strips Report  
 NUM\_STRIPS = 16  
 LEDS\_PER\_STRIP = 64  
  
Banks Report  
 NUM\_BANKS = 4  
 LEDS\_PER\_BANK = 256  
   
 Bank Enable Pins = 5, 6, 7, 8  
 Bank Data Pins (Data/Clock) = 1/2, 3/4   
  
 strip Data pin Clock pin StripStart StripEnd  
Bank = 0  
 0 1 2 960 1023  
 2 3 2 896 959  
 4 1 4 832 895  
 6 3 4 768 831  
Bank = 1  
 4 1 2 704 767  
 6 3 2 640 703  
 8 1 4 576 639  
 10 3 4 512 575  
Bank = 2  
 8 1 2 448 511  
 10 3 2 384 447  
 12 1 4 320 383  
 14 3 4 256

1. LEDMatrix Functions

## LEDMatrix Functions

**The previous version of LEDMatrix include descriptions of the basic graphics draw routines (circle, square, triangle, etc.)**

These are all still supported in LEDMatrix\_22, with a select few renamed to reflect Adafruit\_GFX naming conventions. See: Other libraries and Documentation above

##### Several Important Changes in LEDMatrix\_22 from FastLED

To handle the new multiplexing Banking features of this library several new functions replace previous FastLED functions. These actually streamline your code, reducing the mixing of LEDMatrix’s leds.xxx() and FastLED.xxx() function calls.

All functions listed are prefaced with as the default with “leds.” as in leds.LEDshow() format. Of course, you can create the led class with any name you want. In this case the example function call will be: [your class name].LEDshow()

\*\*ALWAYS USE LEDMatrix FUNCTIONS IN PLACE OF FastLED FUNCTIONS\*

| LEDMatrix\_22 | FastLED | Comments |
| --- | --- | --- |
| *leds.addleds()* | FastLED.addleds() | This call REPLACES the FastLED call in setup(). NO PARAMETERS ARE NEEDED. This function initializes the controllers for the LED Extender Shields. |
| *leds.LEDshow()* | FastLED.show() | Refreshes display w/ and w/o Extender banking |
| *leds.LEDshow(brightness)* |  | Display with a local brightness |
| *leds.SetBrightness(brightness)* | No global brightness for multiple controllers | Sets global brightness w and w/o banking |
| *leds.fillScreen(color)* | FastLED.fillScreen(color) | Fills the display w/ and w/o banking |

#### NEW Functions in LEDMatrix\_22

##### General Functions

| New to LEDMatrix\_22 | Comments |
| --- | --- |
| *clear()* | Clears to black AND displays in 1 step |
| *showColor(color)* |  |
| *setBrightness(uint8\_t bght)* |  |
| *fadeAll(uvalue)* | This was in the Cylon() example in FastLED. It is a useful graphics feature and has been added. |
| *drawPixel(x, y, color)* | Draw pixel in previous libraries did not properly rotate the drawing functions. |

##### Graphics General Functions

| New to LEDMatrix\_22 | Comments |
| --- | --- |
| *CRGB getPixel(x, y);* | Return pixel color in CRGB format |
| *drawTriangle(x0, y0, x1, y1, x2, y2, color);* | Color triangle Filled rectangle with rounded corners |
| *drawFilledTriangle(x0, y0, x1, y1, x2, y2, color);* | Filled color triangle |
| *drawRoundRect(x, y, w, h, r, color);* | Rectangle with rounded corners |
| \* drawFillRoundRect(x, y, w, h, r, color);\* | Filled rectangle with rounded corners |

#### Display “Block save” and “Block restore” Functions

**24 bit full-color (CRGB) Block functions are new. These are blocks of the display that can be saved and restored. For example, save a square area of drawn background, display a sprite or other figure, then restore the bitmap background.**

| New to LEDMatrix\_22 | Comments |
| --- | --- |
| *boolean blockInit(blockNum, w, h)* | Create space for a block of CRGB memory as #n with width h, and height h. Return error if memory not created. |
| *boolean blockStore(blockNum,  x1,  y1)* | Save a block of CRGB memory as #n (with width h, and height h). Return error if save failed. |
| *blockRestore(blockNum)* | Display block #n at its original location. |
| *blockRestore(blockNum, x1,  y1)* | Display block #n at its new location x1, y1. |
| *freeBlock(blockNum)* | Free up memory block #n for reuse. |

##### 24 bit color Bitmap Functions (CRGB color = 3 bytes RGB)

**In addition to previous single color bitmap functions, 24 bit color (CRGB) bitmap functions are. Bitmaps are block of the display that can be saved and restored. For example, save a square area of drawn background, display a sprite or other figure, then restore the bitmap background.**

| New to LEDMatrix\_22 | Comments |
| --- | --- |
| *drawBitmap24(x, y, bitmap, w, h,  bg)* | Display, at x, y (top left), of width w and height h, matching background color bg |
| *drawBitmap24(x, y, bitmap, w, h,  progMem,  zigzag,  bg)* | Display bitmap #n, at x, y (top left) , of width w and height h, in PROGMEM matching background color bg |
|  |  |

##### Loading Bitmaps

Bitmaps can be added to a Sketch in 2 ways 1) Adding the array in the Sketch. The format is listed here. Define the size, and zigzag format for you code here also.

PROGMEM option is false.  
  
#define BOMBJACK01\_W 16  
#define BOMBJACK01\_H 16  
#define BOMBJACK01\_Z true //zigzag  
#define BOMBJACK01\_P false //progmem  
  
const long BombJack01[] = {  
0x0099ff, 0xffffff, 0x000000, …..  
….  
…….0x0099ff, 0xffffff, 0x000000  
};

1. Using #include bitmapName.ext to load the bitmap into PROGMEM directly. The file format is similar to that above. See the example bitmaps for more details.

#include "DigDug02.c"  
  
I recommend you use this format and add this code to bitmap files to remove occasional redefinition warnings:  
   
// Create the array of retro arcade characters and store it in Flash memory  
//24bit color  
//>>>>>>>>>ALL forward direction  
   
#ifndef DIGDUG01  
#define DIGDUG01  
   
#define DIGDUG01\_W 16  
#define DIGDUG01\_H 16  
#define DIGDUG01\_Z false //zigzag  
#define DIGDUG01\_P true //progmem  
   
#ifdef \_\_AVR\_\_  
 #include <avr/io.h>  
 #include <avr/pgmspace.h>  
#elif defined(ESP8266)  
 #include <pgmspace.h>  
#else  
 #ifdef PROGMEM //remove redef warning  
 #undef PROGMEM  
 #endif  
 #define PROGMEM  
#endif  
   
<bitmap array>  
#endif //DIGDUG01

##### 24 bit color Sprite Functions (CRGB color = 3 bytes RGB)

**Sprites are smaller 24 bit color (3 byte RGB) bitmaps. Any number of pre-drawn sprites can be saved in code or loaded into PROGMEM space and displayed in sequence to produce actions or sequences.**

| New to LEDMatrix\_22 | Comments |
| --- | --- |
| *spriteInit( spriteNum, w, h, bitmapName,  progMem,  zigzag);* | Prepare the sprite #n with the name “bitmap.” Indicate where stored, and if normal or zigzag rows. |
| *drawSprite( spriteNum, x, y,  bg);* | Display sprite #n, at x, y (top left), matching background color bg |
| *eraseSprite( spriteNum, x1, y1,  bg)* | Erase sprite #n from x, y, setting background to bg. |

##### LEDShow variations

In addition to LEDShow(), you can change the brightness. If the Extender is used, you have control over when to display each Bank or Banks of up to 4 led stripss in each Bank. See the Extender section for more details.

| New to LEDMatrix\_22 | Comments |
| --- | --- |
| *LEDShow();* | Refresh entire matrix, including all banks and strips |
| *LEDShow(gBrightness)* | Show with a new brightness |
|  |  |
| **Display individual Banks:** |  |
| *LEDShow(Bank,  gBrightness);* | Show/refresh/display a individual bank (i.e. segmented portions of the matrix panel and the attached led stripss). |
| *LEDShow( Bank1,  Bank2,  gBrightness);* | Any two Banks |
| *LEDShow( Bank1,  Bank2,  Bank3,  gBrightness);* | Any 3 Banks |

1. Set up using configuration\_22.h

### Files in the LEDMatrix\_22 Library

* LEDMatrix.cpp, LEDMatrix.h are the actual library. You Sketch must include this line at the top:

#include < LEDMatrix.h>

* The file *configuration\_22.h* (located in the library folder) defines the parameters of the led matrix panel, and optionally Blocks and Extender.
  + Since *configuration\_22.h* is in the library folder, **not the sketch folder**, it is available to all your sketches.
* The files *gfxfont.h* and *glcdfont.c* are default text fonts.
  + These can be included when needed for text in configuration\_22.h using: *#define ENABLE\_FONTS true //true/false*
* The folder example\_configuration files contain several configuration header file with and without Blocks and Extender.
* The folder example **XYTable\_LookUp** files contains several lookup table of both normal and zigzag types.
  + Your lookup customized table can be Enabled in LEDMatrix\_22.h using: *#define XYTable\_LookUp*
* The folder report\_Generator is the code to print the current matrix configuration to the serial monitor
  + Enable reporting in your sketch by enabling: *#define RUN\_REPORT*

## Set up Steps

Before setting the definitions in the configuration\_22.h file, let’s review how large led matrices are laid out. This is the most confusing part of using LEDMatrix. Look at the figure below as you decide on your led panel layout.

The most popular leds are now serial leds i.e. wiring goes from led #1 then to led #2, and so on, like Christmas light strings. This compares to older “RGB” leds that worked using a wiring matrix of anode and cathode wires. LEDMatrix\_22 support both 1-wire and 2-wire led stripss. 1-wire uses 1 data lead and timing cycles to transfer data to the led string. 2-wire led stripss use a data and a clock to transfer data. They both have advantages and disadvantages that are not discussed here.

So a led matrix panel is made up one long led strips. This may include a simple 8x8 array with 64 leds or a 32x32 led matrix with a whopping 1024 leds. The led strips can be string out in one continuous zigzag pattern (even rows one direction, odd the opposite), either horizontally or vertically. Alternatively, the string can be cut into equal lengths and every row running in the same direction or zigzag. Finally, the first led of the string can be in any of the 4 corners of the panel.

**LEDMatrix can be configured for any of these configurations.**

> *For “simple” layouts, you will use only Section 1 of the* configuration \_21.h file*.*

> *For tiles array use Section 1 and Section 2.*

> *If using the hardware Extender, use all 3 sections.*

##### Tiles a.k.a. Blocks - Understanding led strip flow, zigzagging, blocks and tiles

Purchased long led stripss are difficult to lay out and glue to a surface. Then often don’t look very professional. An alternative if to buy smaller 8x8 led panels. For example, Dotstar or NEOpixel panels from Adafruit. Placing these panels next to each other into a larger 16x16, 8x32 or other layout works great. A second approach is to buy or make your own panels, and add discrete leds. Services like jlcpcb.com will do both at a reasonable price. Previous versions of LEDMatrix called these tiles “blocks” in configuration definitions so I will use tiles and blocks interchangeably.

At 1st look, all the definitions seem confusing, but in LEDMatrix\_22 these are separated into three groups, with detaileds explanations of each parameter. Here are the definition with code comments removed for clarity.

**The configuration header file is broken into 3 sections:**

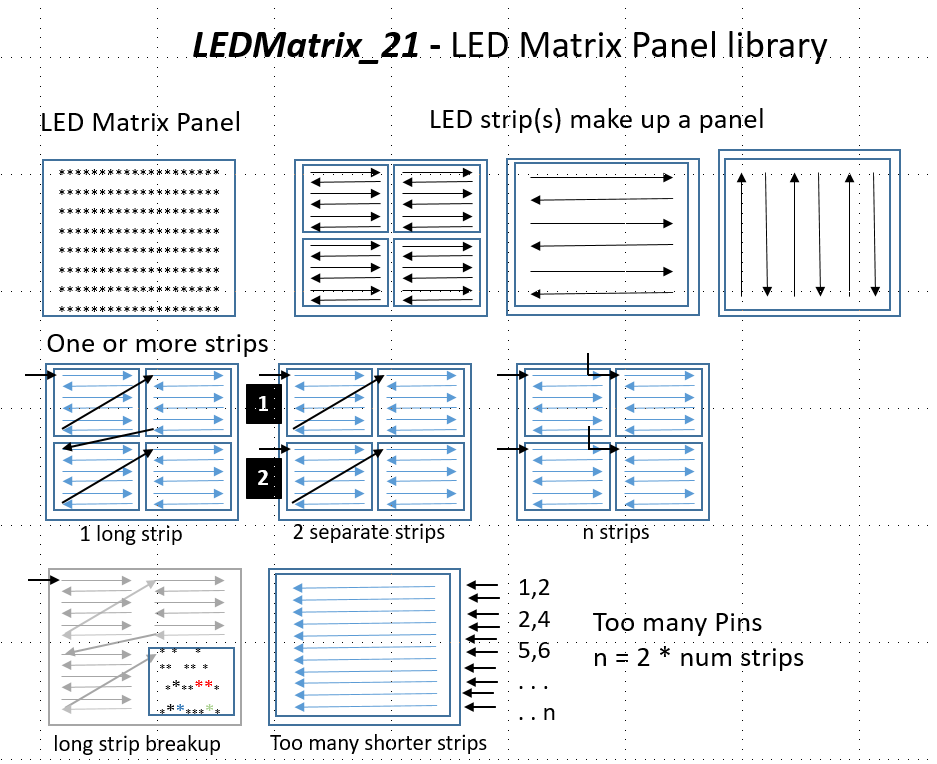
**Section 1: Required matrix array definitions**

**Section 2: #define HAS\_BLOCKS**

**Section 3: #define HAS\_EXTENDER (for both 1-wire and 2-wire leds)**

Setting the following defines to true/false will enable/disable these sections:

#define HAS\_BLOCKS true/false //Section #2  
#define HAS\_EXTENDER true/false //Section #3



1. Section 1 matrix Panel Configuration

### Section 1

If XYTable\_LookUp is defined below, use an external table named XYTable\_LookUp.h, in the library folder to map the leds in XYTable[y][x] instead of calculating with mXY(x,y). The table is stored in PROGMEM.

#define XYTable\_LookUp  
#ifdef XYTable\_LookUp  
 #include "XYTable\_LookUp.h"  
#endif

Teensy 4.0 and 4.1 are the preferred MCUs, but other 32 bit MCUs are fine. UNO and other 8 bit MCUs are too slow.

//#define FastLED\_TEENSY3 //no teensy4 Enabled for DATA\_RATE\_MHZ()  
#define FastLED\_TEENSY4 //defined for DATA\_RATE\_MHZ() and FAST\_SPI in fastSPI\_ARM\_MXRT1062.h

======================== set up physical led type, number ========================= |

|  |  |
| --- | --- |
| Definitions | Descriptions |
| #define COLOR\_ORDER BGR | Find these parameters in FastLED Documentation |
| #define CHIPSET APA102 | see FastLED docs or examples for list |
| #define CLOCK\_PIN\_REQUIRED  true/false | MUST MATCH led type. LEDMatrix cannot detect FastLED’s internal led type. |
| #define CORRECTION TypicalSMD5050 | set Correction type: *TypicalSMD5050, TypicalLEDStrip, Typical8mmPixel, TypicalPixelStrinp* |
| #define SPI\_MHZ 72 | WORKS EVEN IF SPI PINS NOT USED. *Find these parameters in FastLED Documentation* |
| #define refresh\_fps 60 | set Max Refresh Rate set in setup() after size set |
| #define BRIGHTNESS 10 | peed, fps, and brightness together to limit the quality of the leds display |

===== set up physical led arrangement in overall matrix then blocks within the matrix =====

Set the overall Panel size in number of leds (POSITIVE VALUES ONLY). Previous LEDMatrix versions use a negative value for reserved (right to left) and (bottom to top). Use HORIZ\_DIR and VERT\_DIR below to do this.

What direction does the FIRST row of leds in the matrix panel go?

| Definition | Descriptions |
| --- | --- |
| #define MATRIX\_WIDTH 32 | former LEDMatrix use negative value for reversed (right to left) |
| #define MATRIX\_HEIGHT 32 | former LEDMatrix use negative value for reversed (bottom to top) |
| #define MATRIX\_TYPE | if this is a simple matrix (no tiles/blocks), then define the flow of the led strip(s), otherwise ignore *HORIZONTAL\_MATRIX, VERTICAL\_MATRIX* |

|  |  |
| --- | --- |
| Definitions | Descriptions |
| #define HORIZ\_DIR LEFT\_2\_RIGHT | *LEFT\_2\_RIGHT, RIGHT\_2\_LEFT* |
| #define VERT\_DIR TOP\_DOWN | *BOTTOM\_UP, TOP\_DOWN* |

//the total number of leds in your display calculated  
#define NUM\_LEDS MATRIX\_WIDTH \* MATRIX\_HEIGHT

1. Section 2 tiles blocks in the matrix panel

### Section 2 - tiles/blocks in the matrix panel

If you’re led matrix is a simple strip of leds (running in any direction), you can ignore this section and the header definitions. Set HAS\_BLOCKS and HAS\_EXTENDER false, or use an example configuration\_22.h without this section.

Is this matrix made up of block/cells of leds? If NO, ignore these

| Definitions | Descriptions |
| --- | --- |
| #define HAS\_BLOCKS true | Set true is has tiles, otherwise false |
| #define MATRIX\_TILE\_WIDTH 8 | width of EACH MATRIX “cell” (not total display) |
| #define MATRIX\_TILE\_HEIGHT 8 | height of each matrix “cell” |
| #define MATRIX\_TILE\_H 4 | number of matrices arranged horizontally (positive value only) |
| #define MATRIX\_TILE\_V 4 | number of matrices arranged vertically (positive value only) |
| #define LEDS\_IN\_TILE HORIZONTAL\_ZIGZAG\_MATRIX | *HORIZONTAL\_MATRIX, VERTICAL\_MATRIX, RIZONTAL\_ZIGZAG\_MATRIX, VERTICAL\_ZIGZAG\_MATRIX* |
| #define TILES\_IN\_MATRIX HORIZONTAL\_BLOCKS | *HORIZONTAL\_BLOCKS, VERTICAL\_BLOCKS, HORIZONTAL\_ZIGZAG\_MATRIX, VERTICAL\_ZIGZAG\_MATRIX* |

What direction does the FIRST row of leds in the tile go?

Definitions Descriptions

#define LEDS\_HORIZ\_DIR RIGHT\_2\_LEFT *LEFT\_2\_RIGHT, RIGHT\_2\_LEFT*

#define LEDS\_VERT\_DIR TOP\_DOWN *BOTTOM\_UP, TOP\_DOWN*

======== end of Tiles/Blocks =======

1. Section 3 setup number of extenders and LED “strips” in each bank

Section #3 of the configuration\_22.h file applies to using multiple led strips and the Dr Oldies LED Extender. This will be detailed later. For example configurations not using the Extender, Section #3 has been left out for simplicity.

If you are NOT using the LED Extender Shields, but want to use up to 4 separate led strips, set HAS\_EXTENDER to true, set the Banks = 1, and the NUM\_STRIPS to your strips. Be sure to assign the DATA or DATA/CLOCK pins correctly. Teensy boards limit the useable pins for 1-wire led strips.

For 1-wire leds, it appears that only some Teensy pins will work at DATA lines.

// Usable pins:  
// Teensy LC: 1, 4, 5, 24  
// Teensy 3.2: 1, 5, 8, 10, 31 (overclock to 120 MHz for pin 8)  
// Teensy 3.5: 1, 5, 8, 10, 26, 32, 33, 48  
// Teensy 3.6: 1, 5, 8, 10, 26, 32, 33  
// Teensy 4.0: 1, 8, 14, 17, 20, 24, 29, 39  
// Teensy 4.1: 1, 8, 14, 17, 20, 24, 29, 35, 47, 53

More details are here: https://github.com/PaulStoffregen/WS2812Serial

#### This section sets up 1 to 4 Dr Oldies LED Extender Shields for up to 16 leds segments

2-wire leds are limited in the number of leds that can be addressed on each strip. Too many leds and they begin to blink and flash erratically. Reducing the send rate, fps, or brightness helps only a little. These Dr Oldies extender boards and Teensy shields increase the number of led stripss that can be used with a reduced pin count. Each extender uses 4 (2 data/2 clock pins) to address 4 led stripss. Up to 4 extenders can be address with the SAME 4 PINS, plus 1 “enable” pin for each of the 4 extender boards - 16 strips with only 8 pins! This dramatically increases the total number of addressable leds!

| Definitions | Descriptions |
| --- | --- |
| #define HAS\_EXTENDER true | true/false multiple led strip support or Dr Oldies LED Extender shields. If not, ignore this section. |
| #define NUM\_BANKS 4 | 1 to 4 extender “banks” |
| #define STRIPS\_PER\_BANK 4 | 1 or more but 4 strips per Bank is the most efficient use of the hardware |

//total number of strips used  
#define NUM\_STRIPS STRIPS\_PER\_BANK \* NUM\_BANKS

———Choose DATA and CLOCK pins in the bank (all banks use the same pins) ———

The same data/clock pins are used for all Banks, and made active by the BANK\_PIN above.

//All 4 are required regardless of 2, 3, or 4 physical strips per Bank.

if CLOCK\_PIN\_REQUIRED // 2-wire pin selection   
 //Select your DATA/CLOCK pins - if using the Extender shield pin selections are limited  
 //depends on how Teensy is rotated on the Extender board   
 #define DATA\_1 1 //Teensy with Extender only 1 or 14   
 #define CLOCK\_1 2 //Teensy with Extender only 2 or 15  
 #define DATA\_2 3 //Teensy with Extender only 13 or 16  
 #define CLOCK\_2 4 //Teensy with Extender only 14 or 17  
  
#else //1-wire DATA only. Teensy pins are limited to just a few  
 //if other MCU change as desired  
 #define DATA\_1 1 //Teensy 4x 1, Teensy 3.5/3.6 1  
 #define DATA\_2 8 //Teensy 4x 8, Teensy 3.5/3.6 8  
 #define DATA\_3 17 //Teensy 4x 17, Teensy 3.5/3.6 10  
 #define DATA\_4 20 //Teensy 4x 20, Teensy 3.5/3.6 26  
#endif

—————- Choose pins to enable each bank ——————-

Define as many as the number of Banks.

\*\*NOTE: Any of the Teensy boards can be positioned in 2 ways on the shield.

Depending on what Teensy pins are used for other purposes, the extender board allows you to rotate the Teensy MCU 180 degrees. Normal placement of the Teensy on the extender board will use for Bank control pins 5,6,7,8 and for data/clock pins 1,2,3,4. By rotating the Teensy board, you can use for Bank control pins 18,19,20,12 and for data/clock pins 14,15,16,17.  
 Alternate pins (18-21) when the Teensy is rotated on the Extender board

Alignment 1. Align the Teensy GROUND pin (next to the 0 pin) with the LARGE “G” on the shield. Alignment 2. Align the Teensy GROUND pin with the SMALL “G” on the shield.

#define BANK\_PIN\_0 3 //3   
 #define BANK\_PIN\_1 4 //4  
 #define BANK\_PIN\_2 5 //5  
 #define BANK\_PIN\_3 6 //6

======================== end of user definitions ===========================

Final note: These parameters slice and dice the FastLED array up into a Bank sizes then into strips in each Bank Number of leds PER BANK - FastLED only “sees” 1 Bank of led stripss, and thinks its the SAME strips even when we are switching banks.

1. Your Sketch and setup()

## Your Sketch and setup()

After configuring your hardware, next create the leds memory arrays in your sketch. At the top of you sketch before setup() add the following. First include the LEDMatrix\_22 library. Next create the matrix panel array in memory. Unfortunately, the calling format when using tiles/Blocks or the Extender configuration is different than without. Choose the call that applies to your project.

**Allow the #if statement to choose the calling format or delete the un-needed code**

At the top of your sketch include LEDMatrix\_22.h. Do not include FastLED.h.

#include <LEDMatrix\_22.h> //includes FastLED.h

BEFORE setup() add the appropriate call for you hardware configuration:

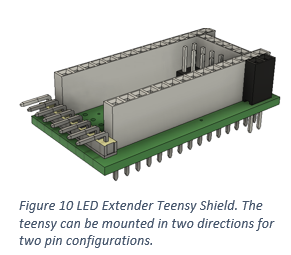
//------------------- create the total matrix panel array -------------------  
#if HAS\_EXTENDER || HAS\_BLOCKS   
 cLEDMatrix<MATRIX\_TILE\_WIDTH, MATRIX\_TILE\_HEIGHT, LEDS\_IN\_TILE, MATRIX\_TILE\_H\_DIR,  
 MATRIX\_TILE\_V\_DIR, TILES\_IN\_MATRIX> leds;  
#else  
 cLEDMatrix<MATRIX\_WIDTH\_DIR, MATRIX\_HEIGHT\_DIR, MATRIX\_TYPE> leds;  
#endif

Inside setup() call leds.addleds().

Setup(){  
 leds.addleds() //sets up the memory and drivers for all the internal functions. No parameters are needed.  
}

A. Dr Oldies LED Extender

## Dr Oldies LED Extender

This Teensy shield comes in several configurations: 1) stackable and “rotatable” shield for Teensy 3.5 to 4.1 (4.0 or 4.1 recommended), and 2) stand-alone PCB for other processor boards. Each design above also has a: 3) 1-wire leds and 4) 2-wire leds pin configuration. 32 bit MCUs are recommended for non-shield versions. 8 bit MCUs don’t have the speed or memory for large led matrix applications.

**Stacking:** Boards come with long-tale stacking header pins to pass-thru all pins including pins not used by The LED Extender.

In setup() CALL leds.addleds() function to initialize the FastLED Banks controllers. c Setup(){ leds.addleds(); } <Figure 10 LED Extender Teensy Shield. The teensy can be mounted in two directions for two pin configurations.>

##### Configurations: for ordering:

* LED Extender for 1-wire led stripss
  + Shield\_1W - Shield for Teensy 3.5 – 4.1 (4.0 or 4.1 recommended)
  + NON-shield\_1W - Non-shield version recommend for 32 bit MCUs
* LED Extender for 2-wire wire led stripss
  + Shield\_2W - Shield for Teensy 3.5 – 4.1 (4.0 or 4.1 recommended)
  + NON-shield\_2W - Non-shield version recommend for 32 bit MCUs

##### Array Memory Use

Instead of one large led array, to use strips and Banks we must use these Controllers - one per led strips BUT for my Banks, the “output” array is only part of the full led array. For a 1024 led array with 4 Banks, each is only 256 leds in 4 strips. Further, for 4 strips in each bank we now have 64 leds in each strip and 4 controllers as below. Finally, each BANK is using the same 4 pins and the SAME 256 led array. memcpy copies the 256 led portion of the fill array into this 256 led output array, once for each Bank.

#### Section 3a: #define HAS\_EXTENDER

If you are not using the multiple led strips or LED Extender Shields, ignore this section and the header definitions. Set HAS\_EXTENDER false, or use an example configuration\_22.h without this section.

#### Section 3b: LED Extender pin definitions in the Extender section

The led Shield can be connected to the Teensy (any 3.2 to 4.1) in two directions. If Teensy pins are needed for other operations or controls the Extender interferes with, hopefully, rotating the shield will free up the needed pins. Here is a list of the pin combinations. See the LED Extender Shield documentation for more details.

#### Understanding the configuration header sections in combination

When you add or remove the tiles/Blocks and the Extender functionality, the initialization calls for LEDMatrix/FastLED changes. To avoid compile errors, use the following comments.

#define HAS\_BLOCKS false   
#define HAS\_EXTENDER false

|  | **Configuration\_22.h settings** |
| --- | --- |
|  | Simple led array with one continuous led strips. |
|  | No tiling or Extender Bank routines available. |
|  | MATRIX\_TILE\_HEIGHT, MATRIX\_TILE\_WIDTH and related variables are not defined |
| **Initialize array with:** | *cLEDMatrix<MATRIX\_WIDTH\_DIR, MATRIX\_HEIGHT\_DIR, MATRIX\_TYPE> leds;* |

#define HAS\_BLOCKS true   
#define HAS\_EXTENDER false

|  | **Configuration\_22.h settings** |
| --- | --- |
|  | Complex arrays with Blocks/tiling sub-configurations. |
|  | Array is made up of one continuous led strips. |
| **Initialize array with:** | *cLEDMatrix<MATRIX\_TILE\_WIDTH, MATRIX\_TILE\_HEIGHT, LEDS\_IN\_TILE,  MATRIX\_TILE\_H\_DIR, MATRIX\_TILE\_V\_DIR, TILES\_IN\_MATRIX> leds;* |

#define HAS\_BLOCKS false   
#define HAS\_EXTENDER true

|  | **Configuration\_22.h settings** |
| --- | --- |
|  | Simple led array without tiles and zigzaging. |
|  | led array can be cut into multiple led stripss (maximum 16). Using the LED Extender to connect strips |
| **Initialize array with:** | *cLEDMatrix<MATRIX\_WIDTH\_DIR, MATRIX\_HEIGHT\_DIR,*  *MATRIX\_TYPE> leds;* |
| **Add addleds() to setup():** | *leds. addleds();* |

#define HAS\_BLOCKS true  
#define HAS\_EXTENDER true

|  | Configuration\_22.h setting |
| --- | --- |
|  | Complex arrays with Blocks/tiling sub-configurations. |
|  | led array can be cut into multiple led stripss (maximum 16). Using the LED Extender to connect strips. |
| **Initialize array with:** | cLEDMatrix<MATRIX\_TILE\_WIDTH, MATRIX\_TILE\_HEIGHT, LEDS\_IN\_TILE, MATRIX\_TILE\_H\_DIR, MATRIX\_TILE\_V\_DIR, TILES\_IN\_MATRIX> leds; |

B. Advanced Topics

### Advanced Topics

Access Directly to the m\_LEDS[] array in the FastLED library One great feature of FastLED is that you have direct access to m\_LEDS[] array of the pixels. However, all LEDMatrix libraries make this array private – so no access. This library makes m\_LEDS[] public (but it’s still in the class cLEDMatrixBase), so the call must be as follows:

leds.m\_LEDS[n] = CRGB::Red; //where n is the index into the led strips.  
or  
leds.m\_LEDS[leds.mXY(x,y)] = CRGB::Red;

#### Memory mapping in LEDMatrix\_22

Below is a figure showing how the arrays are laid out. leds[] is the CRGB (color) sized array that contains the entire led matrix panel array. It is a 1-dimenion array, that is access through LEDMatrix functions in x,y coordinates. LEDMatrix’s leds[] array and FastLED’s m\_leds[] are equivalent. These are created in the cMatrixController() call in your sketch.

For multiplexing/banking, a new array, e\_leds[] is created in the .addleds(function call in setup(). This creates an array the size of the length of the longest led strips times the number of strips in a bank. So, up to for additional FastLED controllers are created. LEDMatrix maps array segments of leds[] into e\_leds[] at each call to the show function.

**NOTE: The addleds() call REPLACES the FastLED.addleds call. All initialization is performed by addleds()**

leds. addleds(); // init params for Extender functions

Here is how the Extender Banking is segmented from the full led matrix array:

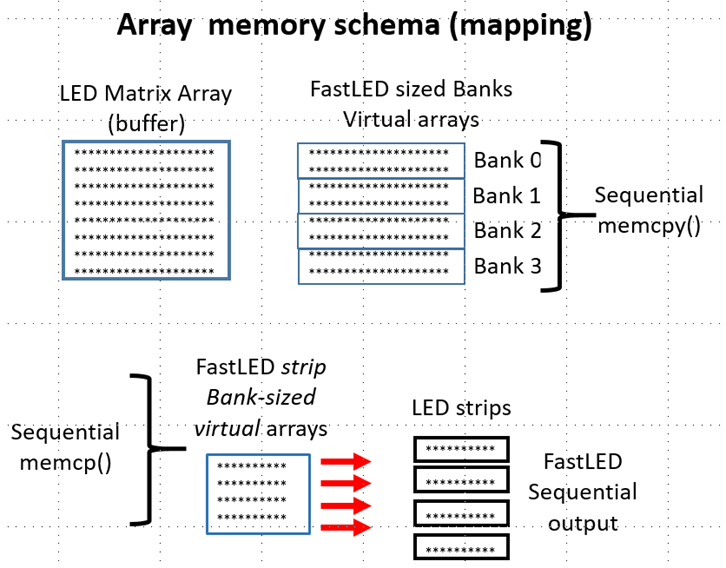


Figure 9 Array memory Schema (mapping)

C. Using const variables instead of #defines

### Using constant variables instead of #defines

##### #define compiler errors

Previous LEDMatrix versions and FastLED used (required) #defined constants like #define NUM\_LEDS to configure the led matrix panel. In LEDMatrix\_22 the configuration definitions for HAS\_BLOCKS and HAS\_EXTENDER are optional. In fact, they may not even be defined if these sections of the configuration\_22.h file are deleted. This can cause compiler errors. For example when MATRIX\_TILE\_HEIGHT is not defined because HAS\_BLOCKS is false.

##### Side stepping this issue

Using constant variables (see the table below) instead of #defines eliminates complier errors if defines are not instantiated (for example when MATRIX\_TILE\_HEIGHT is not defined because HAS\_BLOCKS is false). LEDMatrix\_22 declares a set of constant variable with similar names. These default to zero if their corresponding HAS\_BLOCKS and/or HAS\_EXTENDER are not defined or set to false.

For example: If HAS\_BLOCKS is not defined or HAS\_BLOCKS = false, MATRIX\_TILE\_HEIGHT is not defined and will cause a compiler error in present ANYWHERE in your sketch. In this case use leds.tileHeight in your code instead of MATRIX\_TILE\_HEIGHT. ”c.” is the class name you used to configure the library. In the examples this is leds. So the variable is leds.tileHeight. The variable leds.tileHeight = 0 by default if Blocks are not Enabled. No complier errors will occur and loops over leds.tileHeight rather than MATRIX\_TILE\_HEIGHT are skipped.

You can test for these cases and handle the condition as needed.

##### FastLED #defined constants – more flexible programming

FastLED requires most matrix parameters to be #defined rather than declared. This is primarily for speed. This can make programming more complicated when you want to use these parameters in you code. LEDMatrix\_22 creates GLOBAL CONSTANT VARIABLES for each of these matrix defines. With the popularity of 32 bit MCUs like Teensy, using constant variables are not as critical to speed loss as 8 bit MCUs like Arduino UNO. While you cannot change the values of these constants, I recommend your use these rather than the defined tokens unless speed is an issue. Here are the constant variable alias for each defined one.

List of all definitions describing the matrix panel

**USING VARIABLES INSTEAD OF DEFINITIONS DURING CODING REMOVES undefined ERRORS and confusing #if….#endif brackets**

“c.” is the user defined class such as “leds.”

| Variable | #defines | Settings |
| --- | --- | --- |
| **Matrix Panel** |  |  |
| c.matrixWidth | MATRIX\_WIDTH | former LEDMatrix use negative value for rev |
| c.matriHeight | MATRIX\_HEIGHT | former LEDMatrix use negative value for rev |
| c.matrixType | MATRIX\_TYPE | HORIZONTAL\_MATRIX, VERTICAL\_MATRIX, |
|  |  | HORIZONTAL\_ZIGZAG\_MATRIX, VERTICAL\_ZIGZAG\_M |
|  |  | \*\*what direction does the FIRST row of leds flow? |
| c.ledsHrorizDir | LEDS\_HORIZ\_DIR | LEFT\_2\_RIGHT, RIGHT\_2\_LEFT |
| c.ledsVertDir | LEDS\_VERT\_DIR | BOTTOM.UP, TOP\_DOWN |
|  |  |  |
| **TilesBlocks** |  |  |
| c.tileWidth | MATRIX\_TILE\_WIDTH | width of EACH MATRIX “cell” (not total across the entire panel |
| c.tileHeight | MATRIX\_TILE\_HEIGHT | height of each matrix “cell” |
| c.tilesPerRow | MATRIX\_TILE\_H | number of matrices arranged horizontally |
| c.tilesPerCol | MATRIX\_TILE\_V | number of matrices arranged vertically |
| c.tileledsFlow | LEDS\_IN\_TILE | HORIZONTAL\_MATRIX, VERTICAL\_MATRIX, |
|  |  | HORIZONTAL\_ZIGZAG\_MATRIX, VERTICAL\_ZIGZAG\_MATRIX |
| c.tileFlow | TILES\_IN\_MATRIX | HORIZONTAL\_BLOCKS, VERTICAL\_BLOCKS, |
|  |  | HORIZONTAL\_ZIGZAG\_BLOCKS, VERTICAL\_ZIGZAG\_BLOCKS |
| c.tileledsHorizDir | LEDS\_HORIZ\_DIR | LEFT\_2\_RIGHT, RIGHT\_2\_LEFT |
| c.tileledsVertDir | LEDS\_VERT\_DIR | BOTTOM\_UP, TOP\_DOWN |
|  |  |  |
| **Extender** |  |  |
| c.numBanks | NUM\_BANKS | 1 to 4 extender “banks” |
| c.stripsPerBank | STRIPS\_PER\_BANK | 1 or more but 4 strips per Bank is the most |
| c.ledsPerBank | LEDS\_PER\_BANK | equally split the total number of leds across |
| c.ledsPerStrip | LEDS\_PER\_STRIP | number of leds in each continuous string |

D. FastLED function list (partial)

### FastLED function list (partial)

These FastLED are relevant to the LEDMatrix\_22 library and can add additional les control to your sketches.

| **FastLED.h** |  |
| --- | --- |
| void setBrightness(uint8\_t scale) | Set the global brightness scaling @param scale a 0-255 value for how much to scale all leds before writing them out |
| uint8\_t getBrightness() | Get the current global brightness setting @returns the current global brightness value |
| inline void setMaxPowerInVoltsAndMilliamps(uint8\_t volts, uint32\_t milliamps) | Set the maximum power to be used, given in volts and milliamps. @param volts - how many volts the leds are being driven at (usually 5) @param milliamps - the maximum milliamps of power draw you want |
| inline void setMaxPowerInMilliWatts(uint32\_t milliwatts) | Set the maximum power to be used, given in milliwatts @param milliwatts - the max power draw desired, in milliwatts |
| void show(uint8\_t scale); | Update all our controllers with the current led colors, using the passed in brightness @param scale temporarily override the scale |
| void show() { show(m\_Scale); } | Update all our controllers with the current led colors |
| void clear(bool writeData = false); | clear the leds, wiping the local array of data, optionally black out the leds as well @param writeData whether or not to write out to the leds as well |
| void clearData(); | clear out the local data array |
| void showColor(const struct CRGB & color, uint8\_t scale); | Set all leds on all controllers to the given color/scale @param color what color to set the leds to @param scale what brightness scale to show at |
| void showColor(const struct CRGB & color) | Set all leds on all controllers to the given color @param color what color to set the leds to |
| void delay(unsigned long ms); | Delay for the given number of milliseconds. Provided to allow the library to be used on platforms that don’t have a delay function (to allow code to be more portable). Note: this will call show constantly to drive the dithering engine (and will call show at least once). @param ms the number of milliseconds to pause for |
| void setTemperature(const struct CRGB & temp); | Set a global color temperature. Sets the color temperature for all added led stripss, overriding whatever previous color temperature those controllers may have had @param temp A CRGB structure describing the color temperature |
| void setCorrection(const struct CRGB & correction); | Set a global color correction. Sets the color correction for all added led stripss, overriding whatever previous color correction those controllers may have had. @param correction A CRGB structure describing the color correction. |
| void setDither(uint8\_t ditherMode = BINARY\_DITHER); | Set the dithering mode. Sets the dithering mode for all added led stripss, overriding whatever previous dithering option those controllers may have had. @param ditherMode - what type of dithering to use, either BINARY\_DITHER or DISABLE\_DITHER |
| void setMaxRefreshRate(uint16\_t refresh, bool constrain=false); | Set the maximum refresh rate. This is global for all leds. Attempts to call show faster than this rate will simply wait. Note that the refresh rate defaults to the slowest refresh rate of all the leds added through addleds. If you wish to set/override this rate, be sure to call setMaxRefreshRate *after* adding all of your leds. @param refresh - maximum refresh rate in hz @param constrain - constrain refresh rate to the slowest speed yet set |
| void countFPS(int nFrames=25); | Get the number of frames/second being written out @returns the most recently computed FPS value |
| uint16\_t getFPS() | Get how many controllers have been registered @returns the number of controllers (strips) that have been added with addleds |
| int count(); | Get a reference to a registered controller @returns a reference to the Nth controller |
| CLEDSController & operator[\\Synology3\Electronics\git repo 2020\markdown documents\LEDMatrix\_22\int x](file:///\\Synology3\Electronics\git%20repo%202020\markdown%20documents\LEDMatrix_22\int%20x); | Get the number of leds in the first controller @returns the number of leds in the first controller |
| int size() | Get a pointer to led data for the first controller @returns pointer to the CRGB buffer for the first controller |
| CRGB \*leds() |  |
| Controller.h |  |
| CLEDSController() : m\_Data(NULL), m\_ColorCorrection(UncorrectedColor), m\_ColorTemperature(UncorrectedTemperature), m\_DitherMode(BINARY\_DITHER), m\_nleds(0) | create an leds controller object, add it to the chain of controllers |
| virtual void init() = 0; | initialize the led controller |
| virtual void clearleds(int nleds) | Clear out/zero out the given number of leds. |
| void show(const struct CRGB \*data, int nleds, uint8\_t brightness) | show function w/integer brightness, will scale for color correction and temperature |
| void showColor(const struct CRGB &data, int nleds, uint8\_t brightness) | show function w/integer brightness, will scale for color correction and temperature |
| void showleds(uint8\_t brightness=255 | show function using the “attached to this controller” leds data |
| void showColor(const struct CRGB & data, uint8\_t brightness=255) | show the given color on the led strips |
| static CLEDSController \*head() { return m\_pHead; } | get the first led controller in the chain of controllers |
| CLEDSController \*next() { return m\_pNext; } | get the next controller in the chain after this one. will return NULL at the end of the chain |
| CLEDSController & setleds(CRGB \*data, int nleds) | set the default array of leds to be used by this controller |
| void clearledsData() | zero out the led data managed by this controller |
| virtual int size() { return m\_nleds; } | How many leds does this controller manage? |
| CRGB\* leds() { return m\_Data; } | Pointer to the CRGB array for this controller |
| CRGB &operator[\\Synology3\Electronics\git repo 2020\markdown documents\LEDMatrix\_22\int x](file:///\\Synology3\Electronics\git%20repo%202020\markdown%20documents\LEDMatrix_22\int%20x) { return m\_Data[x]; } | Reference to the n’th item in the controller |

E. List of Functions

## List of Functions

**All Arrays are Public**

In LEDMatrix\_22 the led array struct CRGB leds[] should be used for addressing color memory. However, both the FastLED array m\_LEDS[] and the LED Extender strip array are pubic for advanced programming.

struct CRGB m\_LEDS; //LEDMatrix\_22 moved to public from protected

struct CRGB e\_LEDS; //Extender output array. Sized to longest led strip

cLEDMatrixBase();

virtual uint32\_t mXY(uint16\_t x, uint16\_t y)=0;

void SetLEDSArray(struct CRGB pLEDS); // Only used with externally defined led arrays

struct CRGB operator[\\Synology3\Electronics\git repo 2020\markdown documents\int n](file:///\\Synology3\Electronics\git%20repo%202020\markdown%20documents\int%20n);

struct CRGB &operator()(int16\_t x, int16\_t y);

struct CRGB &operator()(int16\_t i);

int Size() { return(m\_WH); } int Width() { return(m\_Width); } int Height() { return(m\_Height); }

void HorizontalMirror(bool FullHeight = true);

void VerticalMirror();

void QuadrantMirror();

void QuadrantRotateMirror();

void TriangleTopMirror(bool FullHeight = true);

void TriangleBottomMirror(bool FullHeight = true);

void QuadrantTopTriangleMirror();

void QuadrantBottomTriangleMirror(); void drawPixel(int16\_t x, int16\_t y, CRGB Col);

void drawLine(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, CRGB Col);

void drawRectangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, CRGB Col);

void drawCircle(int16\_t xc, int16\_t yc, uint16\_t r, CRGB Col);

void drawFilledRectangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, CRGB Col);

void drawFilledCircle(int16\_t xc, int16\_t yc, uint16\_t r, CRGB Col);

================LEDMatrix\_22 additions ============================  
  
CRGB getPixel(int16\_t x, int16\_t y);  
void fadeAll(uint16\_t value);  
void fillScreen(CRGB color);  
void drawTriangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, int16\_t x2, int16\_t y2, CRGB color);  
void drawFastVLine(int16\_t x, int16\_t y, int16\_t h, CRGB color);  
void drawFastHLine(int16\_t x, int16\_t y, int16\_t w, CRGB color);  
void drawFilledTriangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, int16\_t x2, int16\_t y2, CRGB color);  
void drawCircleHelper(int16\_t x0, int16\_t y0, int16\_t r, uint8\_t cornername, CRGB color);  
void drawFillCircleHelper(int16\_t x0, int16\_t y0, int16\_t r, uint8\_t cornername, int16\_t delta, CRGB color);  
void drawRoundRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h, int16\_t r, CRGB color);  
void drawFillRoundRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h, int16\_t r, CRGB color);  
void clear();  
void showColor(CRGB color);  
void setBrightness(uint8\_t bght);

———————–text—————————————–

void setCursor(int16\_t x, int16\_t y);  
void setTextColor(CRGB c);  
void setTextColor(CRGB c, CRGB bg);  
void setTextSize(uint8\_t s);  
void setTextWrap(boolean w);  
void setRotation(uint8\_t r);  
void cp437(boolean x=true);  
void setFont(const GFXfont \*f = NULL);  
int16\_t getCursorX(void) const; // get current cursor position (get rotation safe maximum values)  
int16\_t getCursorY(void) const;  
uint8\_t getRotation(void) const;  
void getTextBounds(char \*string, int16\_t x, int16\_t y, int16\_t \*x1, int16\_t \*y1, uint16\_t \*w, uint16\_t \*h);  
void getTextBounds(const \_\_FlashStringHelper \*s, int16\_t x, int16\_t y, int16\_t \*x1, int16\_t \*y1, uint16\_t \*w, uint16\_t \*h);  
void drawChar(int16\_t x, int16\_t y, unsigned char c, CRGB color, CRGB bg, uint8\_t size);  
void write(char);  
void print(char text[]);  
void invertDisplay();  
void invertSquare(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1);

—————————–bitmaps ———————————-

void drawBitmap(int16\_t x, int16\_t y, const uint8\_t \* bitmapName, int16\_t w, int16\_t h, CRGB color);  
void drawBitmap(int16\_t x, int16\_t y, const uint8\_t \* bitmapName, int16\_t w, int16\_t h, CRGB color, CRGB bg);  
void drawBitmap(int16\_t x, int16\_t y, uint8\_t\* bitmapName, int16\_t w, int16\_t h, CRGB color);  
void drawBitmap(int16\_t x, int16\_t y, uint8\_t\* bitmapName, int16\_t w, int16\_t h, CRGB color, CRGB bg);  
void drawXBitmap(int16\_t x, int16\_t y, const uint8\_t\* bitmapName, int16\_t w, int16\_t h, CRGB color);

=======LEDMatrix\_22 additions - 24 bit full color bitmaps ==============

void \_bitmapZigzag(int16\_t x, int16\_t y, uint8\_t i, uint8\_t j, int16\_t w, boolean zigzag, CRGB col);   
void drawBitmap24(int16\_t x, int16\_t y, const long\* bitmapName, int16\_t w, int16\_t h, boolean progMem, boolean zigzag, CRGB bg);   
void drawBitmap24(int16\_t x, int16\_t y, const long\* bitmapName, int16\_t w, int16\_t h, CRGB bg);

========LEDMatrix\_22 additions - 24 bit full color sprites =============

void spriteInit(uint8\_t spriteNum, int16\_t w, int16\_t h, const long\* bitmapName, boolean progMem, boolean zigzag); void drawSprite(uint8\_t spriteNum, int16\_t x, int16\_t y, CRGB bg); void eraseSprite(uint8\_t spriteNum, int16\_t x1, int16\_t y1, CRGB bg);

======LEDMatrix\_22 additions - 24 bit full color screen block save-restore ======

boolean blockInit(uint8\_t blockNum, uint8\_t w, uint8\_t h);  
boolean blockStore(uint8\_t blockNum, int16\_t x1, int16\_t y1);  
void blockRestore(uint8\_t blockNum, int16\_t x1, int16\_t y1);  
void blockRestore(uint8\_t blockNum);  
void freeBlock(uint8\_t blockNum);

==============LEDMatrix\_22 EXTENDER hardware ======================

void LEDShow();  
void LEDShow(uint8\_t gBrightness);  
void LEDShow(uint8\_t Bank, uint8\_t gBrightness);  
void LEDShow(uint8\_t Bank1, uint8\_t Bank2, uint8\_t gBrightness);  
void LEDShow(uint8\_t Bank1, uint8\_t Bank2, uint8\_t Bank3, uint8\_t gBrightness);  
void addleds();  
void defineBanks();

F. Errors and Possible Fixes

### Errors and Possible Fixes

##### Sparkles and pattern breakup (Dotstar and APA102)

Data transmission for 2-wire leds fail if strips are “too long.” Depending on your setup, this may be after 255, 144, or only 50 leds. Possible solutions:

**First – Be sure your GROUND is connected to the MCU, led stripss, and the power supply solidly!**

* The best solution is to use my LED Extender shields to break down the array into multiple sets.
* Use heavier data/clock wires (many Dupont jumpers wires are only 26 awg.)
* Use heavier power lines and “inject” addition power along the strips. Dimming at far end of strip
* Use heavier power lines and “inject” addition power along the strips.

##### Redefined PROGMEM Error

This occurs as more fonts or bitmaps are included into LEDMatrix\_21. For example glcdfont.c does this.

#ifdef \_\_AVR\_\_  
 #include <avr/io.h>  
 #include <avr/pgmspace.h>  
#elif defined(ESP8266)  
 #include <pgmspace.h>  
#else  
 #ifdef PROGMEM //remove redefinition warning  
 #undef PROGMEM  
 #endif  
 #define PROGMEM  
#endif

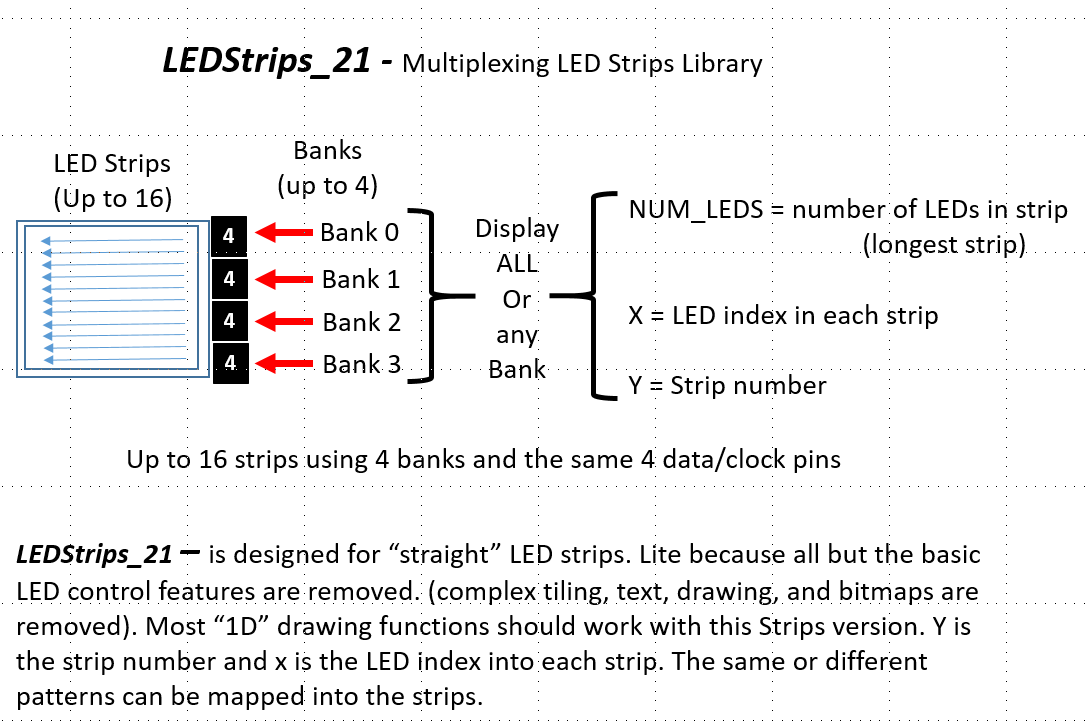
##### Report\_Generator compile errors

This .cpp may cause compile errors in Visual Studio or VS Code when a new sln or proj is created. Simply REMOVE the .cpp from the editor’s file explorer list (without deleting). It’s in Source Files.

G. Multiple LED strips not than in a Matrix

### Multiple led stripss not than in a Matrix

LEDStrips\_22 is a separate library designed for led stripss not in a matrix panel. This is a “lite” version because the matrix functions have been removes {circle, triangles, bitmaps, etc.). New functions to control individual banks and strips have been added.



H. Multiplexing to increase matrix panel size

## Multiplexing to increase matrix panel size

Several parameters limit the length of a 2-wire led strips, and thus the size of your matrix before sparkling and random pixilation occur. These include: - Length of the strip (obviously) - Quality of the leds - voltage drop along the 5 volt wires - wires too thin - power supply too small - one of more poor connections - low voltage of the data and clock lines - Data transfer speed - fps (frames per second) refresh rate - led brightness

### Fast\_LEDS multiple “controllers”

With FastLED you can break long strips into a number of shorter strips by creating multiple “controllers.” [More details here](https://github.com/FastLED/FastLED/wiki/Multiple-Controller-Examples)

### LEDMatrix\_22 multiple “controllers”

LEDMatrix\_22 combines one of these FastLED multiple controller schema with a hardware “Extender” board to expand the number of led stripss with a minimum of MCU pins. This section explains the implementation. In the figure below, look at Bank 0. Using 1 to 4 FastLED controllers, we can send data and clock signals to to up to 4 corresponding led stripss. This uses 2 data and 2 clock pins rather than 4 pairs (saving 4 pins). Without getting into the details yet, LEDMatrix\_22 multiplexes the data and clock lines to that only one pair if data/clock pins synchronize with 1 led strips at a time.

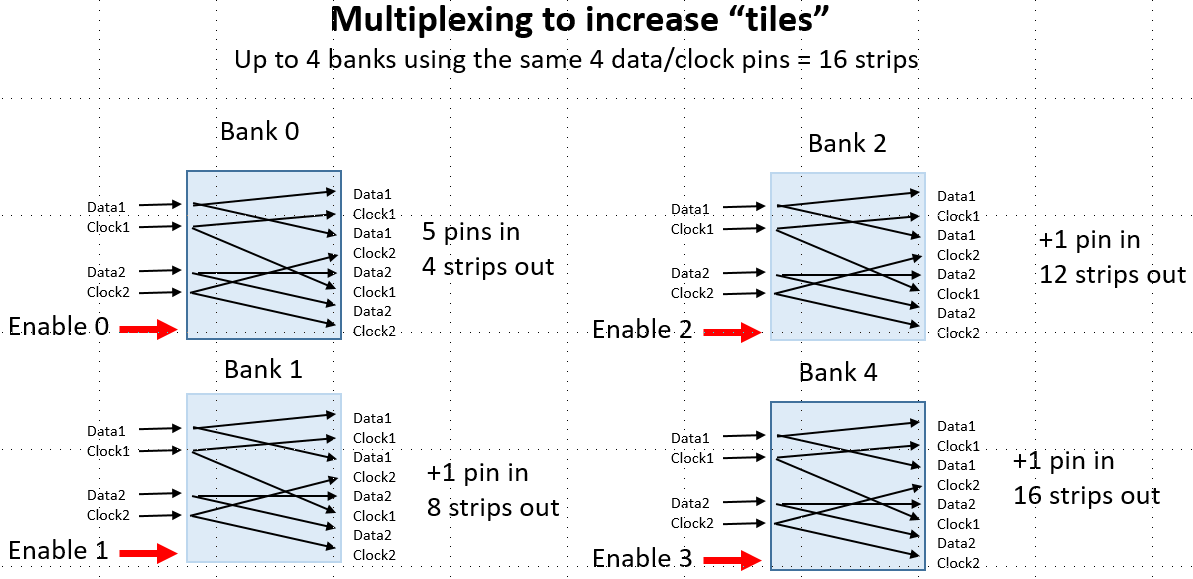
This is a reduction of 8 pins to 4 pins.

Adding a 2-input AND gate chip, to the extender board allows multiple boards to be “stacked,” into multiple Banks. This is done by utilizing the chip’s enable pin to enable on Bank at a time. So with 2 data, 2 clock, and any number of enable pins, led stripss can be broken down into shorter lengths - reducing or eliminating led breakdown.

Number of MCU pins versus strips supported using the Extender

| # data pins | # clock pins | # enable pins | led stripss | pins saved |
| --- | --- | --- | --- | --- |
| 2 | 2 | 0 | 4 strips | 4 |
| 2 | 2 | 2 | 8 strips | 10 |
| 2 | 2 | 3 | 12 strips | 17 |
| 2 | 2 | 4 | 16 strips | 28 |

See the Section Dr Oldies LED Extender for more details. | | | — |



I. Example Configurations and Sketches

## Examples

In the examples folder there are 3 types of examples:

• example XYTable\_LookUp files

• XYTable\_LookUp16x16

• XYTable\_LookUp32x32

• example\_configuration files

• 1x144 led strip • 8x8\_LEDS\_zigzag\_Panel

• 16x16\_2x2 tiles\_zigzag\_NO\_extender

• 32x32\_4x4\_tiles\_zigzag\_4x4\_extender

• 32x32\_4x4\_tiles\_zigzag\_NO\_extender

• examples (Sketches)

• bitmaps\_and\_sprite\_tests

• bitmap\_test

• bitmap24\_test

• block\_copy

• sprites\_test

• Cylon\_22

• Cylon\_strip

• Extender test examples

• flowingRainbow\_22

• fontTest

• introduction

• LEDMatrix\_22

• stepThruBasicFunctions

• Table\_Mark\_Estes\_LEDMatrix\_22

• test\_configuration\_1st

• testBrightness2Failure

#### fontTest.ino

//fontTest.ino will run on any size panel

You can use Adafruit\_GFX default and custom fonts with LEDMatrix\_22. You can dowbload these fonts with the Adafruit library or ither web sites.

Using custom fonts can be confusing. This example shows how to include and use these fonts. One point to note is that AF default fonts x,y is in the upper left of the character. Custom foints x,y is in the lower left. Wrapping does not appear to work with all custom fonts.

cFTest(); //This test creates a structure array to load an use multiple custom fonts.  
  
//fontTest(TomThumb, text); //simple font test with a small font  
//textTest(); //These two use the default font  
//printTest();

#### Introduction.ino

RUN LEDMatrix\_22.ino 1ST TO TEST YOUR led matriX PANEL HARDWARE AND SOFTWARE CONFIGURATION

This sketch introduces you to the numerous functions of this library. Here are the demo functions. These should work regardless of the size of your panel, but some may look odd if the panel is smaller than 16x16. For 24 bit bitmapping and sprites see the sketches specific to those library functions. Don’t try these until you have the basics down.

Scottish\_Flag  
canvasTest  
textTest  
invertTest  
printTest  
StepThru  
Cylon  
flowingRainbow

### LEDMatrix\_22.ino

Menu driven mapping tests for LEDMatrix\_22 library

This sketch allows you to test the mapping of your matrix panel. These menu functions are available through the serial terminal. The general report is printed first so you can review your settings.

If your setting do not include tiling or the extender hardware, these choices will be disabled.

MOST PROBLEMS ARE CAUSED BY IMPROPER SETTINGS OF THE MATRIX PANEL, TILES/BLOCKS. OR EXTENDER SETTINGS

Note: TT\_numbers\_Progmem.h is a small “Tom Thumb” character set file with only numbers to display

#### MENU tiling and bank sequences.

\*Sketch functionality\*  
Locate and exercise your matrix panel, Extender Banks and Strips.  
This is an interactive menu on serial monitor to select functions:  
 If no blocks, menu stops after 'd'   
 If no Extender, menu stops after 'e'   
  
a. We will report your configuration.  
b. Draw triangles in starting corner and each corner of matrix  
IF HAS\_BLOCKS  
c. Draw arrows in starting corners and centers of tiles  
d. Light up each block/tile AND print the number of the Block in sequence  
IF HAS\_EXTENDER  
e. Light up each Bank and number in sequence  
f. Light up each strips  
g. Display this MENU

#### testBrightness2Failure

APA102, DotStar and similar leds, are sensitive to receving data and passing it along a long strip of leds. The maximum limit is a combination of Brightness, speed (spi\_Mhz) and frames per second (fps) for the best balance. Use this sketch to find that balance.

Count the number of brightness levels until your matrix panel leds breakup. Adjust the speed (Mhz) and frames per second (fps) for the best balance.

#### Table\_Mark\_Estes\_22

Mark Estes’ Table code modified (yet again) to work with the LEDMatrix\_22 library. Steps through 113 patterns, and modifies the patterns on following passes. With this library should be scalable to any size led matrix panel. However, not all patterns will be centered in both x and y for non-square panels.

#### Mods for LEDMatrix\_22

Edited the short list to run through more interesting patterns. Just a few global changes needed for example change class to leds. Fixed bugs in #8 and #9 that caused lockup

Table.h Split out variable and functions definitions to this header for clarity

#### Pattern control

Via the Arduino serial port, turn off line endings, and use ‘n’ and ‘p’ to change patterns or send ‘80’ to jump to pattern 80 directly.

#### Table\_Mark\_Estes (previous version from marc merlin)

Mark Estes’ Table code modified to work with just a matrix

Define BESTPATTERNS (Table.h) Lets you see some of my favorite patterns given that the whole list is otherwise 113 patterns long :)

#### Video examples (from marc merlin)

* 64x64 with audio: https://photos.app.goo.gl/qLS14Ad6UzCng3Q23
* More 64x64 with audio: https://photos.app.goo.gl/syEPi7O97hgsWKb53
* My 32x24 panel compilation of my favorites patterns: https://www.youtube.com/watch?v=SSllLL5SGCg