Arduino - 3208 Led matrix

Autor:
Data de publicació: 26-05-2016

3208 Red LED Dot Matrix Display Information Board?DE-DP13119?

.

https://cdn-shop.adafruit.com/datasheets/ht1632cv120.pdf

Brief Introduction

As the second member of the latest 3208 monocolor LED dot matrix display info board series, this HT1632C-based item integrates bright red LEDs built in each of 32*8 3mm dots. It can be used in many applications to display letters or digits. Users can build a driver board or choose the driver board (DE-DD21113) by Sure Electronics to light up this LED board and customize their own character libraries to display their favorite symbols and adjust the contrast ratio of this display for better visual effect.
Net Weight:100g /3.6oz

Features:

Size: 128.00 (L) × 60.00 (W) × 23.84 (H) ±0.2mm
HT1632C-based 32*8 LED dot-matrix info board
Light emitting diameter: 3mm
Display color: red
10-level PWM brightness control
Either one of the two 10-pin male sockets can be used to connect drive board, which powers the 32*8 LED, as well as send data and control signal in a SPI-like mode
Two +12V auxiliary power supply terminals provided
Up to 4 boards are recommended to be connected in series to display more characters

Specifications:

Parameter

Symbol

Value

Unit

Operating Voltage

Vin

12

V

Storage Temperature

Tstg

-20-80

?

Average Operating Temperature

Iavrg

0.20

A

Maximum Operating Temperature
(All LEDs on, 100% PWM duty cycle)

Imax

0.36

A

Applications:

Indoor and outdoor LED sign, elevator, advertising display, bus station and airport displays and other different display areas of equipment

Package Contents:

3208 Red LED 3mm Dot Matrix Display Information Board x 1
10-pin IDC cable ×1
demo board x1

3208 Single Color LED Dot Matrix Display Information Board Series:

Product Number

Product Name

DE-DP13111

3208 Green LED 3mm Dot Matrix Display Information Board

DE-DP13119

3208 Red LED 3mm Dot Matrix Display Information Board

DE-DP13211

3208 Green LED 5mm Dot Matrix Display Information Board

DE-DP13212

3208 Red LED 5mm Dot Matrix Display Information Board

Note:

The accessories might be a little different from what you've seen on webpage since the accessories you received might be from different batch. However, it won't affect your use. Sure Electronics assumes no liability for any ambiguity caused by the product photos.

Documents:

Download Center for Manual, Driver and Tutorial.

Keyword:

DE-DP13119

HT1632C

SureElectronics sells some low cost LED dot matrix displays. The 32x16 bicolor(red, green) uses the new HT1632C and I couldn't find any code for Arduino that works with these displays.

I put some schematic here using my Arduino Mega2560 a RTC DS1307 and two 3216 led matrix working together.

A video with this configuration running is available at YouTube. //www.youtube.com/watch?v=2ZPACHAZxWg

You can follow the development on this thread on the Arduino Forum.

A reusable library is available here, on GitHub.

Even though the reusable library mentioned above is very powerful, it was originally not made for supporting the 32x16 bicolor(red, green) board type. So, changes have been made to adapt it and are posted here in this link, on GitHub.

Page constructed by Wagner Sartori Junior with help of florinc, Tim Gilmore and westfw on forum.

BUGS:

In this code, it lasts more than 1 second to refresh(loop). We can use shadowram on plot function to compare before send that is faster I think.

TODO:

A reusable library
Draw char(an easier and clever way)
Scrolling
Blink

Code: ht1632c.h

HT1632 Arduino "Matrix Display" Library for the Sure 2416 and 0832

Introduction
The reasonably low cost Sure Electronics 2416 has the simple HT1632 controller. This panel has one defining benefit aside from cost, you can cascade up to 4 panels to create all kinds of effects.

If you're interested in purchasing a board and are in the UK check out [Mnet:Hardware]

Library
The DisplayMatrix library is designed to drive the RED Sure-Electronics 24x16 display (DE-DP016). This library makes it easy to cascade multiple displays so you can build really large panels for a reasonable price!

Installation
You'll need Arduino IDE 0017 or 0018 to use this Library.

Just unzip and drop the library into arduino-0018/libraries/MatrixDisplay. Then simple navigate to the various examples through the Arduino IDE.

Green displays Iván Lalaguna Alcaine has provided an simple but effective patch for green displays.

Latest
https://github.com/milesburton/HT1632 - 0832 Displays

[MatrixDisplay 2.01 for 0832 displays] - Matrix Display 2.01 for 0832 Displays

Archive
28/11/2010 - [MatrixDisplay 2.00 for 1624 displays] - Fully re-written with multi-device support.

20/04/2010 - [MatrixDisplay for 0832 displays] - Same as MatrixDisplay 200 but for the 08x32 display

20/01/2010 - [Sure2416 1.00] - The first official release*

I did not write the original code for this release, I purely put it in library form, re-factored a little and added examples. A full re-write is coming soon (including support for 0832 displays)

Change Log
2.01 (0832) - Martin Raynsford spotted a couple of mistakes. Failing to properly boundary check on XYToIndex and an "off-by-one" comparison mistakes within the getDispNum. Thanks Martin!

2.0 (1624) - A fully rewritten library with multi-display support

1.0 - Essentially a re-factor and collation of various methods and functions found on the web.

Contributors
If you want to contribute please send your changes to me ;) Related Work and Contact. There's plenty of code within the MatrixDisplay library which has been borrowed from around the web, thanks everyone for opening up your source!

Example
The following code has been taken from the "Simple" example in MatrixDisplay 2.00. It's designed to show you some of the basic functions of the library and how to use it.

view sourceprint?

```
001.#include "MatrixDisplay.h"

002.#include "DisplayToolbox.h"

003.

004.// Easy to use function

005.#define setMaster(dispNum, CSPin) initDisplay(dispNum,CSPin,true)

006.#define setSlave(dispNum, CSPin) initDisplay(dispNum,CSPin,false)

007.

008.// 4 = Number of displays

009.// Data = 10

010.// WR == 11

011.// False - we dont need a shadow buffer for this example. saves 50% memory!

012.

013.// Init Matrix

014.MatrixDisplay disp(4,11,10, false);

015.// Pass a copy of the display into the toolbox

016.DisplayToolbox toolbox(&disp);
```

```
017.

018.// Prepare boundaries

019.uint8_t X_MAX = 0;

020.uint8_t Y_MAX = 0;

021.

022.void setup() {

023.Serial.begin(9600);

024.

025.// Fetch bounds

026.X_MAX = disp.getDisplayCount() * (disp.getDisplayWidth()-1)+1;

027.Y_MAX = disp.getDisplayHeight();

028.

029.// Prepare displays

030.//  The first number represents how the buffer/display is stored in memory.  Could be useful for reorganising the displays or matching the physical  layout

031.// The number is a array index and is sequential from 0. You can't use 4-8. You must use the numbers 0-4

032.// The second number represents the "CS" pin (ie: CS1, CS2, CS3, CS4) this controls which panel is enabled at any one time.

033.disp.setMaster(0,4);

034.disp.setSlave(1,5);

035.disp.setSlave(2,6);

036.disp.setSlave(3,7);

037.}

038.

039.

040.void loop()

041.{

042./*

043.The MatrixDisplay library treats each display individually. As a result you can set

044.each display by it's seperate coordinates. For example:

045.
```

046.disp.setPixel(0, 5, 10, 1);

047.

048.This sets display 0 (which you defined above)

049.Coordinate x = 5

050.Coordinate Y = 10

051.Turn the LEDs on = 1 (0 for off)

052.

053.Alternatively you may wish to use the ToolBox. The toolbox assumes that each of your displays

054.are set out horizontally. Display 0 is the first through n. Using the toolbox you can access the display as

055.if it were on big virtual display. For example:

056.

057.toolbox.setPixel(5, 46, 1);

058.

059.Coordinate x = 5

060.Coordinate y = 46 (notice that's a virtual coordinate, larger than 23)

061.Turn the Leds on = 1 (0 for off)

062.

063.Once you have set the pixels you'd like on. You need to sync the displays (basically write the buffer to the device).

064.

065.disp.syncDisplays();

066.

067.Alternatively there's a few tricks you can use. If you sync the displays, the ENTIRE buffer is written out - that is hugely slow (comparatively).

068.It may more efficient to write each pixel as you go. For example:

069.

070.Just as shown above but you can add a "paint" argument. If you set this to true the library will write the pixel straight to the display.

071.You wont need to use disp.syncDisplays(); if you don't want to.

072.toolbox.setPixel(5, 46, 1, true);

073.*/

074.

```
075.// Write directly to the display

076.for(int y=0; y < Y_MAX; ++y)

077.{

078.for(int x = 0; x< X_MAX; ++x)

079.{

080.toolbox.setPixel(x, y, 1, true); // Lets write straight to the display.

081.}

082.}

083.

084.delay(2000); // Wait two seconds

085.

086.// Okay lets clear the buffer

087.disp.clear();

088.// ...and write the result to the displays

089.disp.syncDisplays();

090.

091.// Lets use syncDisplays now

092.// Write directly to the display

093.for(int y=0; y < Y_MAX; ++y)

094.{

095.for(int x = 0; x< X_MAX; ++x)

096.{

097.toolbox.setPixel(x, y, 1); // Notice we've discarded the "true". This means we're no longer writing to the display directly

098.}

099.}

100.

101.// Now we've written to the back buffer, lets write out the result to the display

102.disp.syncDisplays();

103.

104.// Now we're here. Why don't we try out another simple function. SetBrightness
```

```
105.for(int i=0; i<16; ++i) // The displays have 15 different brightness settings

106.{

107.// This will set the brightness for ALL displays

108.toolbox.setBrightness(i);

109.// Alternatively you could set them individually

110.// disp.setBrightness(displayNumber, i);

111.delay(200); // Let's wait a bit or you'll miss it!

112.}

113.

114.

115.

116.// Okay lets clear the buffer

117.disp.clear();

118.// ...and write the result to the displays

119.disp.syncDisplays();

120.

121.// We're all done. let's start looping

122.}
```

TODO

Improve the efficiency with a "dirty column" bit-field.

Release version with Green display and 0832 support (they have a different pixel mapping)

Troubleshooting

If you are using any MCU besides the AtMega328 you may need to change the "bitBlast" method of MatrixDisplay.cpp. Change the called function to digitalWrite which takes advantage of the inbuild port selection/write

Projects

Sure 2416 Displaying CPU Graph via Arduino

Sure 2416 Running Pong via Arduino

HT1632 for Arduino v2.0 (HT3208-16 github)

NOTE: This new version of the software changes the underlying data format - upgrading is easy and largely automated. See upgrade instructions.

This is a powerful library that allows an Arduino to interface with the popular Holtek HT1632C LED driver. It allows programmers to directly perform advanced drawing of images and text with minimal code.

Quick Start
Code
Here's some sample code to draw a blinking heart on the middle of a single screen:

```
#include <HT1632.h> // Include this before  or any font.
#include <images.h>

void setup () {
    HT1632.begin(pinCS1, pinWR, pinDATA);
    // Where pinCS1, pinWR and pinDATA are the numbers of the output pins
    // that are connected to the appropriate pins on the HT1632.
}

void loop () {
  // The definitions for IMG_HEART and its width and height are available in images.h.
  // This step only performs the drawing in internal memory.
  HT1632.drawImage(IMG_HEART, IMG_HEART_WIDTH,  IMG_HEART_HEIGHT, (OUT_SIZE -
IMG_HEART_WIDTH)/2, 0);

  HT1632.render(); // This updates the display on the screen.

  delay(1000);

  HT1632.clear(); // This zeroes out the internal memory.

  HT1632.render(); // This updates the screen display.

  delay(1000);
}
```

Before running the code, go to HT1632.h and check the USER OPTIONS section and follow the instructions to specify the type of board you are using.

Explanation
HT1632 is always initialized by calling the begin function, like so:

```
HT1632.begin(pinCS1 [, pinCS2 [, pinCS3 [, pinCS4]]], pinWR, pinDATA);
```

All pins are set to OUTPUT and memory is allocated and cleared automatically. The square brackets denote an optional argument.

The HT1632 class stores an internal copy of the state of each screen used. All drawing and writing functions operate on this internal memory, allowing you to perform complicated compositing. Once the internal memory is ready for display, the render() function will efficiently send the updated image to the screen.

Utilities
Image Drawing
This project includes an image-drawing utility, written in HTML5 (using the canvas tag) and JavaScript. It has been tested on Firefox 3.6.* on OSX.

The editor provides the data in a ready-to-paste format that allows for quick drawing of fonts and/or images. It can load previously drawn images as well (even those from v1 -- see upgrading instructions for more details).

It's use should be self-evident. You can find it in Utilities/image_drawing.html.

Font Creation
Fonts are defined as a series of images of equal height and variable width. (Note: Currently, fonts are only tested to at most one byte per column.) The bytes making up each character, from ASCII char 32 (space) onwards, are appended together without any byte alignment.

An array, FONT_8X4_END, encodes information necessary to extract the width and the offset of the character from the font array. The element at position i encodes the first index of character i + 1.

Generating FONT_NAME_NUM

This array can be generated automatically using Utilities/font_end_generate.html. Make sure your font has one character per line (with no trailing newline) and paste it into the tool.

If you want to skip character i, simply set FONT_NAME_NUM[i] = FONT_NAME_NUM[i - 1] or FONT_NAME_NUM[0] = 0 if i == 0. Using the font_end_generate.html tool, just leave a blank line.

Advanced Use
There are a few examples in Arduino/HT1632/examples/.

Bicolor Displays
This library natively supports Bicolor displays, using the selectChannel(n) function to switch to color channel n before rendering images. Calls to clear() and render() automatically operate on all channels.

An example is available in HT1632/examples/HT1632_Heart_Bicolor/.

NOTE: You need to call HT1632.setCLK(PIN_NUMBER_CLK); before calling begin(), where PIN_NUMBER_CLK is the Arduino pin connected to the CLK of your Bicolor board. Refer to the example.

NOTE: Make sure you have set the board type in HT1632.h. If you specify a wrong NUM_CHANNEL value, it won't work properly.

Brightness Control
The HT1632C comes with a 15-level PWM control option. You can control the brightness (from 1/16 to 16/16 of the duty cycle) of the current drawing target using the setBrightness(level) function, where level is a number from 1 to 16. level must never be zero!

If you want to simultaneously set multiple boards to the same brightness level, you can pass a bitmask as an optional second argument, like so: setBrightness(8, 0b0101). The rightmost bit is the first screen, while the fourth bit from the right corresponds to the fourth screen. In the above example, the first and third screen are set to half brightness, while the second and third remain unchanged.

Multiple HT1632s
This library supports up to 4 chips at a time (though technically more can be run concurrently). To take advantage of this, specify multiple CS pins in the initialization.

All rendering occurs on the first display by default. A scrolling text example is available in HT1632/examples/HT1632_Text_8X4_Multidisplay/.

Do note that all drawing happens to a single buffer. You need to clear() the contents of the buffer if drawing different graphics to different screens. To draw the same image to multiple screens, call renderTarget() and render() once per target.

Multiple HT1632s, each with multiple color channels, are supported natively.

Bugs & Features
Known Issues

Initialization doesn't automatically assign a single HT1632C as the RC Master - some unknown bug prevents this from working. As a result, multiple HT1632Cs only need the power and data pins connected, leaving the OSC and SYNC pins disconnected.

A single Arduino cannot support both a Mono-color and a Bi-color display.

Future Plans

Support for direct pixel access and primitive drawing.
Test support for fonts taller than 8px.
Allow FONT_8X4_END arrays to be either 2-byte ints or 1-byte uint8_ts