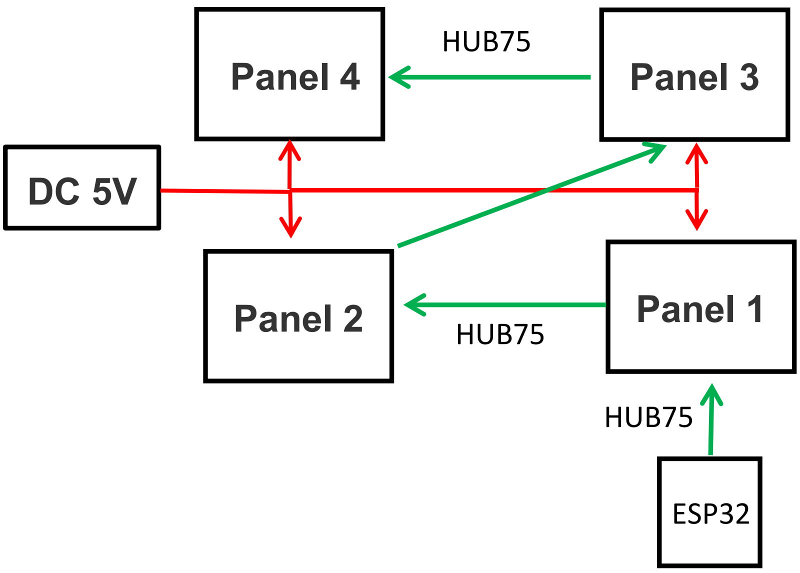
Resource Links：

https://drive.google.com/drive/folders/1IrSCvr2SVra0k1dCZQLyA0TeeSEUoPfw?usp=drive_link

Tutorial on displaying pictures on the ESP32 platform

Description: On the ESP32 platform, four 64X64 dot matrices are spliced together to display pictures. The wiring method is as follows:



Note:

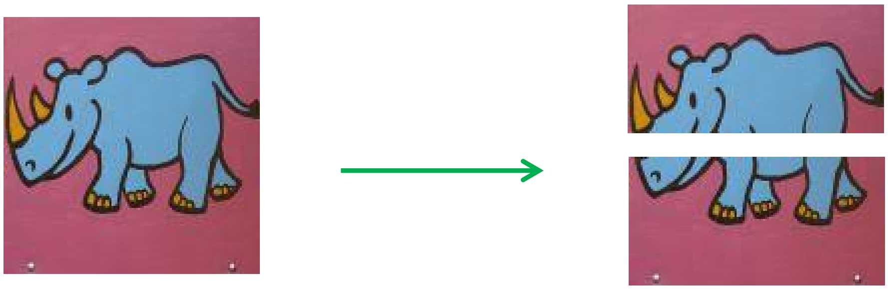
1. HUB75 is a 16P gray signal cable, and the power cable is a 4P red and black cable. One end of the HUB75 cable is connected to ESP32 (for specific pin definitions, please refer to the WIKI webpage content), and the other end is connected to the Sinnal Input Interface of Panel 1. One end of the second HUB75 signal cable is connected to the Sinnal Output Interface of Panel 1, and the other end is connected to the Sinnal Input Interface of Panel 2. The connection order is shown in the figure above. At the same time, a 4P red and black cable is required to power the two matrix panels.

2. It is recommended that the matrix panels be placed as shown in the figure above, so that it is consistent with the demo codes, so that the picture can be displayed normally.

1. Picture preparation

Prepare an RGB888 (bit depth 24 bits) and a resolution of 128x128 picture, which can be in jpg or bmp format, and then split the picture into two pictures with a resolution of 128x64 (users can find relevant software to split it themselves).

As shown in the figure below:



2. Use Image2Lcd to convert image data

Double-click the Image2Lcd.exe file to open the software, as shown in Figure 1 below

Figure 1

Click "Open" at "1" in Figure 1, and then select the picture prepared in step 1. After opening the picture, it will be shown in Figure 2 below. The red box in the figure is the main setting parameters. Select "C language array (.c)" for "Output data type", "Horizontal scan" for "Scan mode", "16-bit true color" for "Output grayscale", and fill in "128" and "64" for

"Maximum width and height" respectively. After setting, click the black triangle arrow at "1" in the red box.

Figure 2

Click the Save button in Figure 2 (see "2" in Figure 2) to save the modulo data, as shown in Figure 3. In the pop-up "Save as C file" window, enter the file name of the modulo data you want to save (see the red box), and then click "Save (S)". The saved 128x64_u.c file will be automatically opened, as shown in Figure 4:

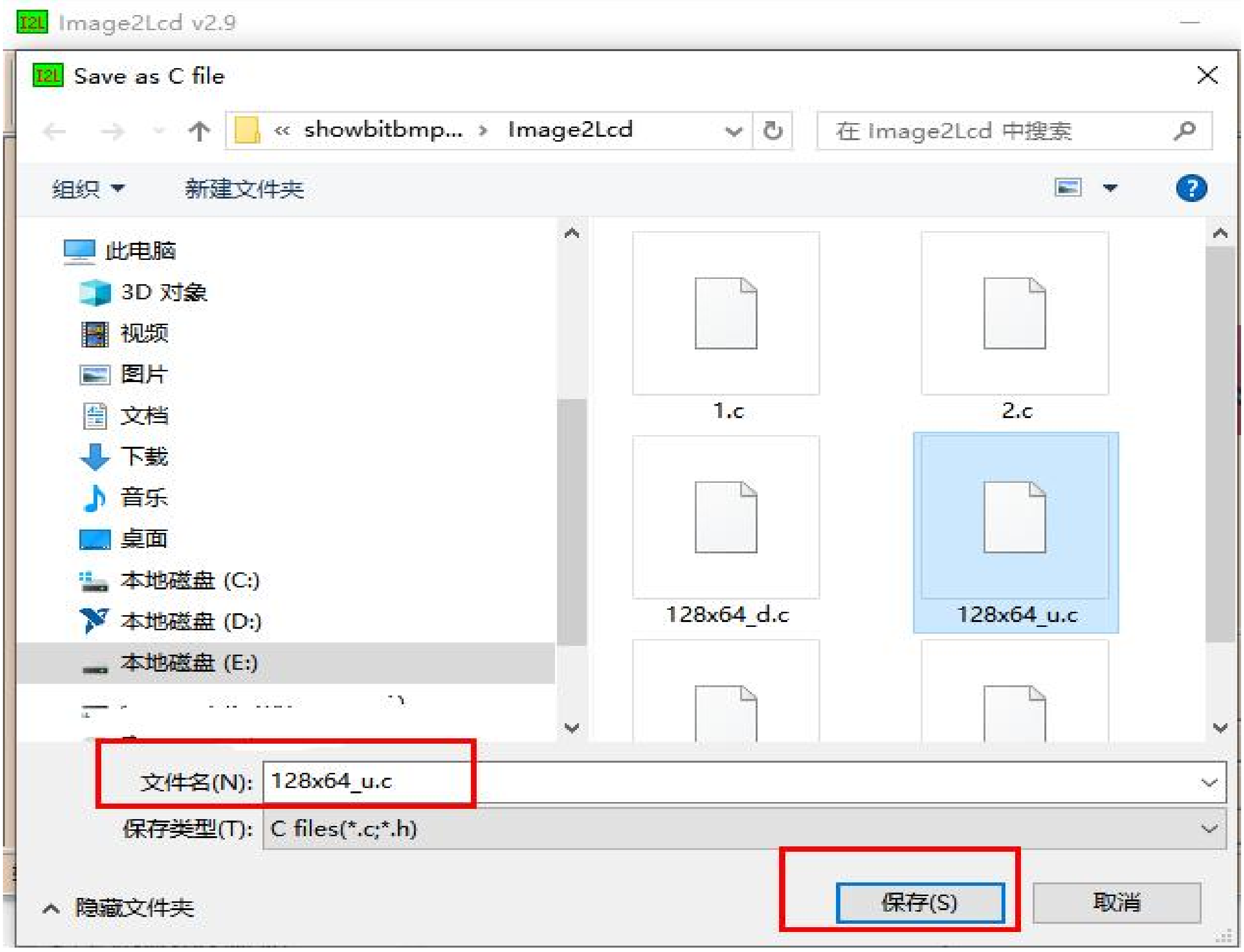
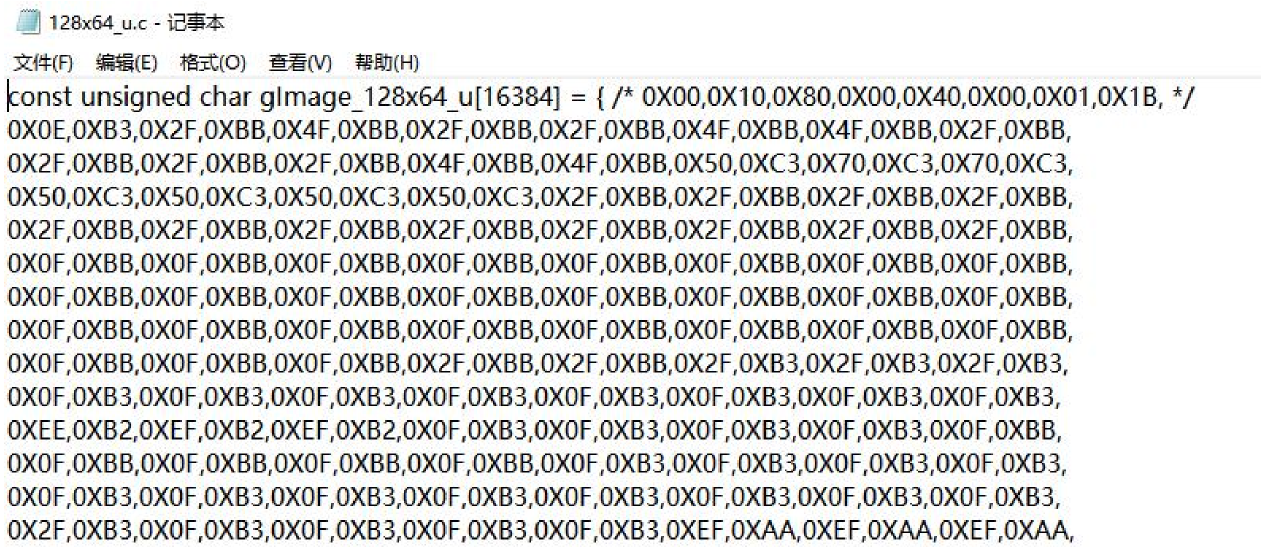


Figure 3


128x64_u.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
const unsigned char gImage_128x64_u[16384] = { /* 0X00,0X10,0X80,0X00,0X40,0X00,0X01,0X1B, */
0X0E,0XB3,0X2F,0XBB,0X4F,0XBB,0X2F,0XBB,0X2F,0XBB,0X4F,0XBB,0X4F,0XBB,0X2F,0XBB,
0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X4F,0XBB,0X4F,0XBB,0X50,0XC3,0X70,0XC3,0X70,0XC3,
0X50,0XC3,0X50,0XC3,0X50,0XC3,0X50,0XC3,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,
0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XB3,0X2F,0XB3,0X2F,0XB3,
0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,
0XEE,0XB2,0XEF,0XB2,0XEF,0XB2,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,
0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,
0X2F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0XEF,0XAA,0XEF,0XAA,0XEF,0XAA,
```

Figure 4

Use the same method to obtain the array gImage_128x64_d.c file of the lower half of the image.

3. Use of image data

Open the \ESP32\showbitbmp\showbitbmp.ino file, copy and paste the contents of the gImage_128x64_u.c and gImage_128x64_d.c files obtained above to the end of the bit_bmp.h file, and modify the type definition of the array, as shown in the red box in Figure 5, then reference the image data array at the end of the showbitbmp.ino file, as shown in the red box in Figure 6, and finally verify and upload it to the ESP32 mainboard.

```

const uint8_t PROGMEM gImage_128x64_u[] = { /* 0X00,0X10,0X80,0X00,0X40,0X00,0X01,0X1B, */
0X0E,0XB3,0X2F,0XBB,0X4F,0XBB,0X2F,0XBB,0X2F,0XBB,0X4F,0XBB,0X4F,0XBB,0X2F,0XBB,
0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X4F,0XBB,0X4F,0XBB,0X50,0XC3,0X70,0XC3,0X70,0XC3,
0X50,0XC3,0X50,0XC3,0X50,0XC3,0X50,0XC3,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,
0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X2F,0XBB,0X2F,0XBB,0X2F,0XB3,0X2F,0XB3,0X2F,0XB3,
0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,
0XEE,0XB2,0XEF,0XB2,0XEF,0XB2,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XBB,
0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XBB,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,
0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,0X0F,0XB3,
```

```

const uint8_t PROGMEM gImage_128x64_d[] = {/* 0X00,0X10,0X80,0X00,0X40,0X00,0X01,0X1B, */
0X51,0XC3,0X8D,0XA2,0XAA,0X92,0X02,0X49,0XE1,0X30,0X43,0X31,0X62,0X00,0XCA,0X21,
0X08,0X01,0X10,0X3B,0XF6,0X74,0X57,0X75,0X37,0X6D,0X79,0X6D,0X7B,0X6D,0X3B,0X65,
0X5B,0X65,0X5A,0X65,0X39,0X6D,0X18,0X75,0XF8,0X7C,0XF8,0X7C,0X39,0X75,0X3A,0X6D,
0X7C,0X65,0X1B,0X55,0X7C,0X5D,0X7B,0X65,0X39,0X6D,0X38,0X7D,0X37,0X8D,0X87,0X21,
0X05,0X21,0X2A,0X3A,0X37,0X85,0XD6,0X5C,0X5A,0X65,0X7A,0X65,0X96,0X54,0X37,0X85,
0XF0,0X7B,0X24,0X39,0XA1,0X38,0X23,0X49,0XE4,0X28,0X67,0X21,0X4F,0X4B,0X59,0X85,
```

Figure 5

```
uint8_t wheelval = 0;
void loop() {
    dma_display->fillScreen(myBLACK);
    dma_display->drawRGBBitmap(20,0,(uint16_t *)Pikachu1_64x64,64,64);
    delay(2000);
    dma_display->fillScreen(myBLACK);
    dma_display->drawRGBBitmap(84,0,(uint16_t *)Pikachu1_64x64,64,64);
    delay(2000);
    dma_display->fillScreen(myBLACK);
    dma_display->drawRGBBitmap(148,0,(uint16_t *)Pikachu1_64x64,64,64);
    delay(2000);
    dma_display->fillScreen(myBLACK);
    dma_display->drawRGBBitmap(0,0,(uint16_t *)gImage_128x64_u,128,64);
    dma_display->drawRGBBitmap(128,0,(uint16_t *)gImage_128x64_d,128,64);
    delay(2000);
}
```

Figure 6

As shown in Figure 7 below, line 11 of the showbitmap.ino file in the code has defined the use of four matrix panels for splicing, so this is defined as "#define PANEL_CHAIN 4"

```
6   #include "ESP32-HUB75-MatrixPanel-I2S-DMA.h"
7   #include "bit_bmp.h"
8
9   #define PANEL_RES_X 64      // Number of pixels wide of each INDIVIDUAL panel module.
10  #define PANEL_RES_Y 64      // Number of pixels tall of each INDIVIDUAL panel module.
11  #define PANEL_CHAIN 4       // Total number of panels chained one to another
12
13  //MatrixPanel_I2S_DMA dma_display;
14  MatrixPanel_I2S_DMA *dma_display = nullptr;
```

Figure 7