Making an SPI Interface for Full Color Dot Matrix LED Panels

P.S.: The program has been redesigned and has become more powerful.
Please see here.

LED panel controller completed Please note that the program in this article is the older on e.

Recently, I found a full-color dot matrix LED panel selling pretty cheaply on Amazon.

Since the width is 64 pixels and the height is 32 pixels, a total of 2048 RGB LEDs are use d.
Still, 3299 yen is extremely cheap, isn't it? It is 1.61 yen per LED.
I searched for others and found a few.

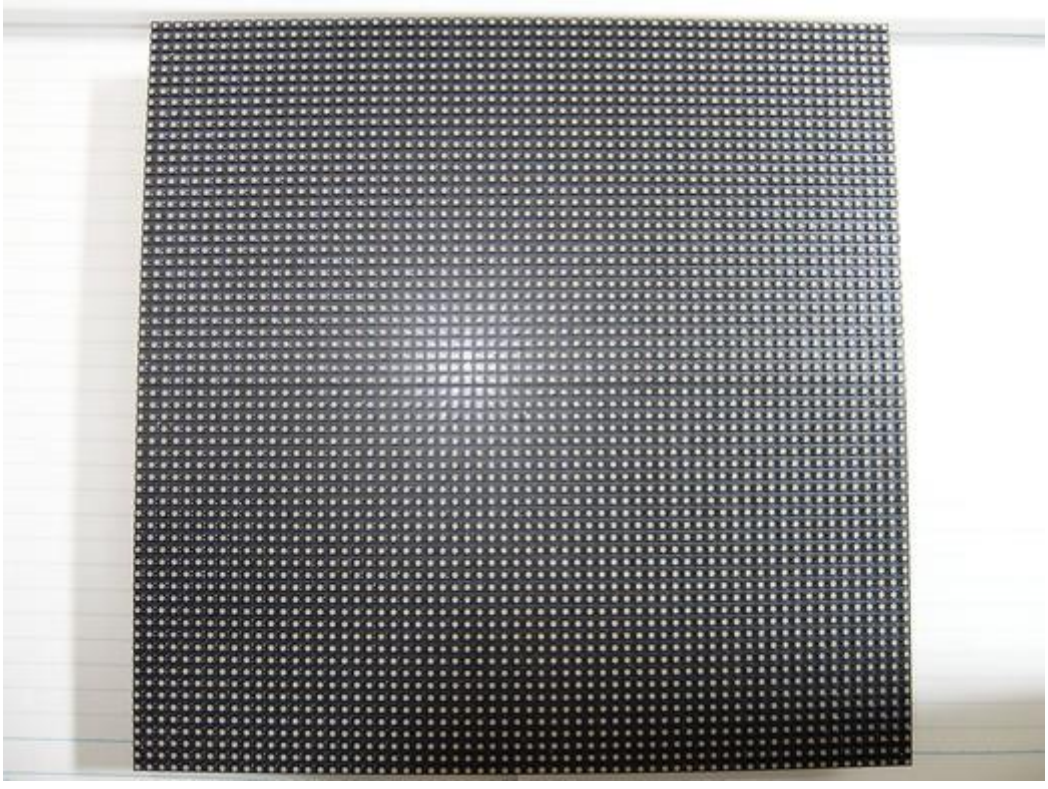
This is 0.81 yen for 1 LED, and
this one is 1.76 yen

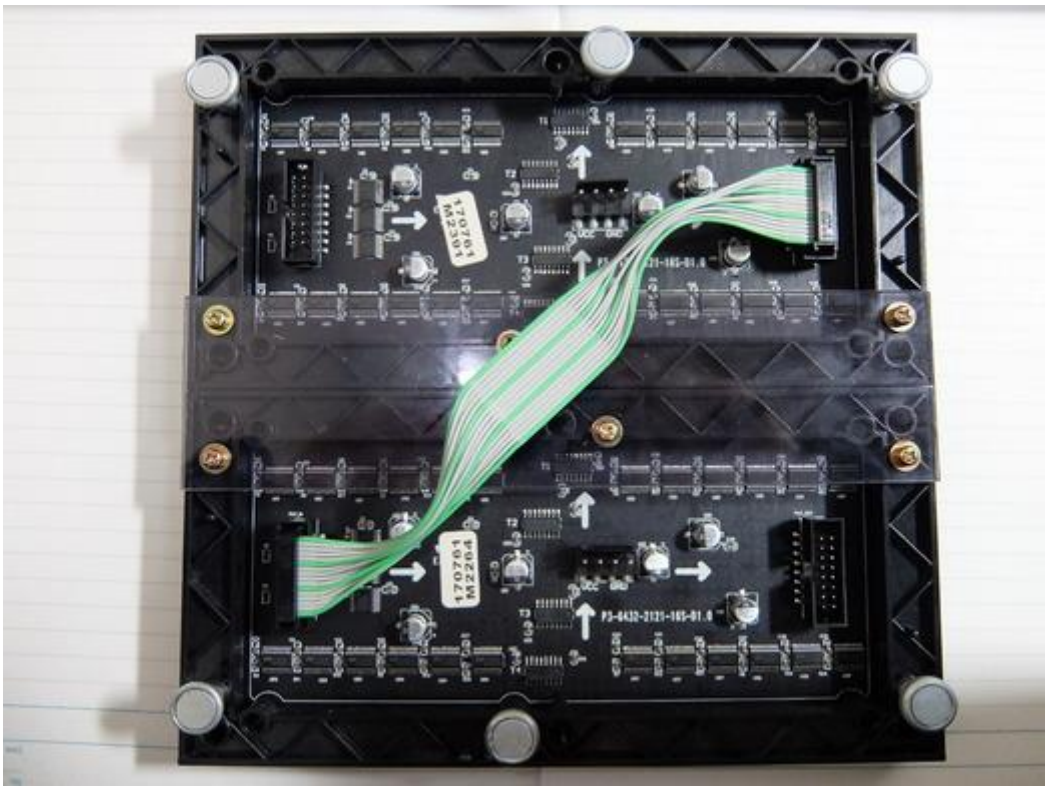
That's why I bought two 64*32 panels with a pitch of 3mm on impulse without even looki ng into it.
Later, I also purchased a 64*64 panel with a 3mm pitch, but this has not arrived yet.

The reason why full-color dot matrix LED panels are sold at such a low price is probably b ecause the demand for LED displays is expanding with the spread of digital signage.

I will post a link to Amazon for the time being, but there is a possibility that the price will change or it will not be handled in the future.
P3 RGB Pixel Panel HD Video Display 64×32 Dot Matrix SMD LED Display Module 192×96 mm
P3 SMD Indoor rgb 32S LED Module P3 64 For Indoor Full Color X64 Pixels LED Video Wal l LED Screen Electronic Scoreboard
P5 RGB Pixel Panel HD Video Display LED Display Module 320x160 1/16s 64×32

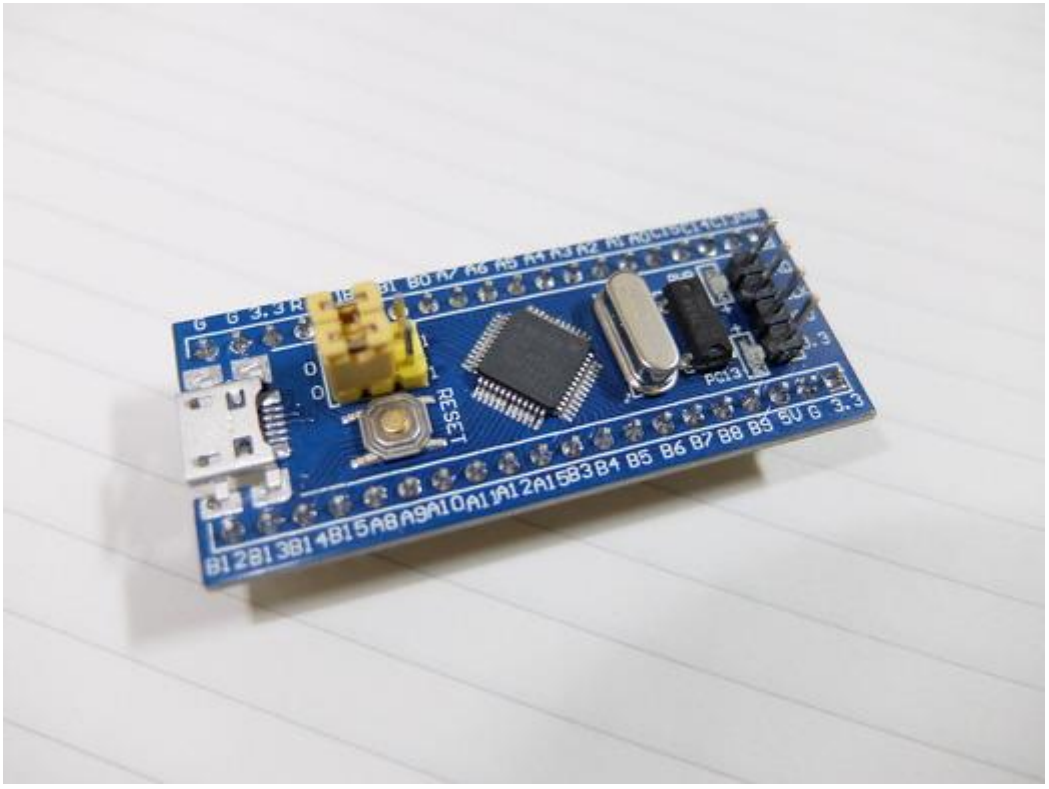
Here is the actual full-color dot matrix LED panel.
Two 64*32 panels are connected.



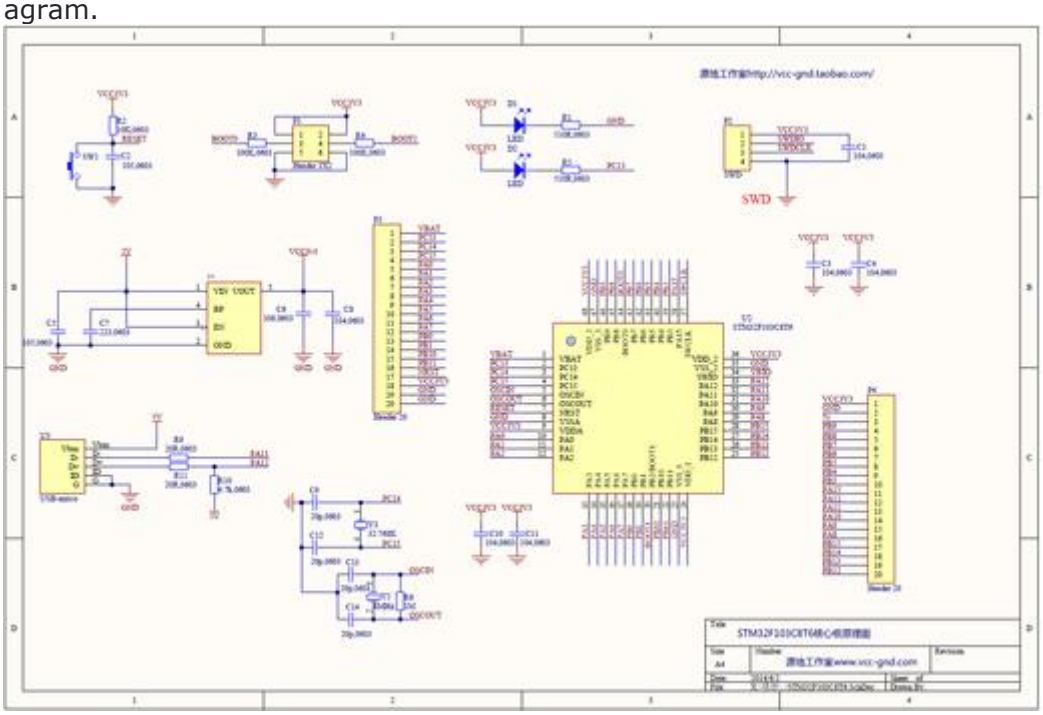


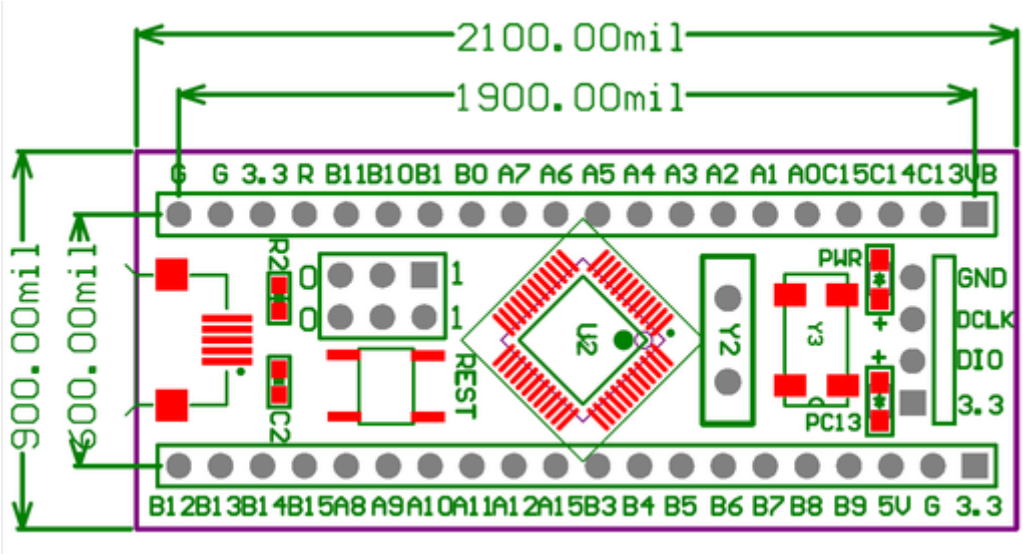
By the way, as for the display method of this full-color dot matrix LED panel, it seems tha t the signal standard called HUB75 is used.
There are several other easy-to-understand sites on the Internet about the HUB75, so I won't go into detail here, but in a nutshell, it's a simple shift register for dynamically light ing the LEDs on the panel.
Therefore, in order to continue displaying the panel of the HUB75, it is necessary to const antly send data, and since the RGB of the LED of each dot is basically only ON or OFF, it i s quite troublesome to perform gradation expression while performing other processing w ith the microcomputer.

Therefore, we created an interface that allows other microcontrollers to easily display full -color dot matrix LED panels with an SPI connection.
The microcontroller used is STM32F103C8T6. I chose this because the board called BluePi ll on which this microcomputer is located is sold on amazon for about 500 yen, and it is e asy to obtain and inexpensive.



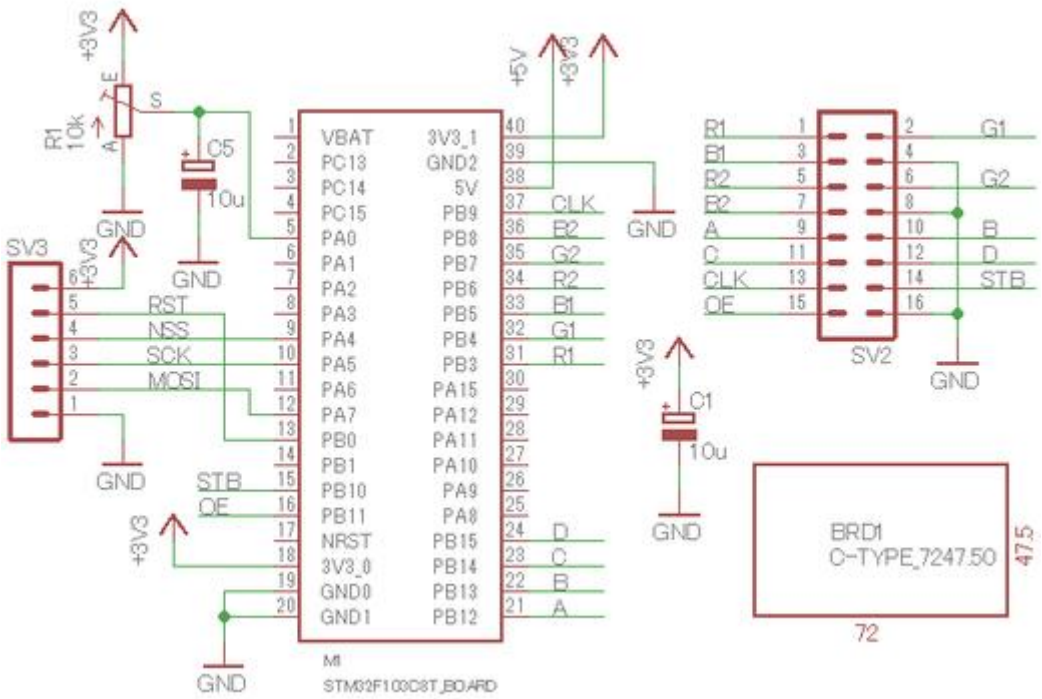
It may be useful for something, so I will also post a circuit diagram and a board outline di agram.





The target specification is to allow the
SPI transmitter to display the RGB luminance information for each pixel simply by transm itting it.
Supported resolutions are 64*32 or higher. Preferably 128*32.
The frame rate is 60 FPS or more.
The color depth is 4 bits or more.
Allow the brightness of the entire screen to be variable.
It was determined.
Actually, I was aiming for the same usability as a full-color OLED graphic display with an SPI connection, but I felt that it was impossible for me to realize it with a microcomputer, so I gave up early.
I wanted to send it separately into commands and data, but the SPI receiver didn't know how to know how many data was actually sent and when it was sent. Therefore, we have made it a specification that simply drips RGB4bit and it is displayed on the panel as it is.

First of all, the circuit diagram.



As you can see, it's quite simple. It is a circuit that is not interesting in terms of hardwar e.
Adjust the brightness of the panel with R1.
Connect the SV2 to a full-color dot matrix LED panel.
Connect the SPI to the SV3.
MOSI: Master Output Slave Input
SCK: Serial Clock
NSS: Slave Select
RST: Reset the DMA transfer of the SPI inside the microcontroller (transfer 1 frame of dat a after toggling this pin)
Since it is a receive-only slave, there is no MISO.

Next is the program.
My STM32 development environment uses STM32CubeMX and SW4STM32.
The development process is to first set up the microcontroller peripherals with STM32Cub eMX, generate code, read the project generated by SW4STM32, write a program, and wri te and debug with ST-LINK/V2.
The following program is an excerpt of what I wrote, omitting the automatically generate d part.
Since it is an uncode generated by a self-taught C language man, please leave it in the c omment section if you have any tips.

main.c stm32f1xx_it.c

I will explain in order from the top.

main.c stm32f1xx_it.c

Here you can change the resolution of the matrix panel. When changing the resolution, b
oth main.c and stm32f1xx_it.c need to be rewritten. (I don't know much about preproces
sors or header files.) MATRIXLED_X_COUNT
: Landscape resolution up to 128
MATRIXLED_Y_COUNT: Portrait resolution up to 32
MATRIXLED_COLOR_COUNT: Number of colors RGB so 3 colors There is no need
to mess with it here MATRIXLED_MAX_Y_ADDRES: Maximum address selection of ABCD i
n the matrix panel For example, if it is up to ABC, it is up to 8 ABCDE, but HUB75E is not
supported by this program
MATRIXLED_PWM_RESOLUTION: PWM resolution
ADC_DIVIDE_VALUE: The value obtained by dividing the ADC value of the variable resist
or for brightness adjustment If the value is lowered, the maximum brightness will be brig
hter, but if it is set too low, the operating time of TIM2, which determines the brightness
of the LED, will exceed the operating cycle of TIM3, which determines the frame rate, an
d the frame rate will decrease
TIM2_ARR_OFFSET_ VALUE: Minimum value of TIM2 It's better

not to mess with it too much main.c

fill_flame_buffer_color is a function that fills the buffer with the RGB value passed in the
argument.
fill_flame_buffer_random is a function that fills a buffer with random values. Each time y
ou call it, the random value changes.
fill_flame_buffer_testpattern is a function that fills the buffer with a test pattern. I often
use it to check contrast.

main.c

A buffer corresponding to the RGB pixels of the panel is reserved in RAM.
Pixel data received by SPI is directly DMA transferred here.
For example, if the MATRIXLED_PWM_RESOLUTION is 16 and you want the uppermost le
ft pixel to be displayed in white, you

can use .

main.c

TIM2 determines the ON time of the LED, and the ON time of the LED is changed by chan
ging the value of the automatic reload register of TIM2 according to the PWM stage.
It is essential to obtain a smooth brightness gradation expression. More on that later.

main.c

This is a loop that converts and outputs data from the buffer to the matrix panel.
Well, I've seen it. I don't think it's a very smart method, and I think there must be a fast

er and better way, but I have no choice but to do this because I have to convert from RG B data.
If you have plenty of RAM, you can convert it from RGB data to data that can be output t o GPIO as it is in advance and perform DMA transfer with a double-buffered feeling, but u nfortunately STM32F103C8T6 was impossible because the RAM was only 20 KB.
Also, if you had a flexible memory controller, you could have taken advantage of this, but STM32F103C8T6 don't have it.
At the moment, this is the most time-consuming part of the process, which is the bottlen eck in improving the frame rate.
Next, I think it's a PWM loop.

main.c

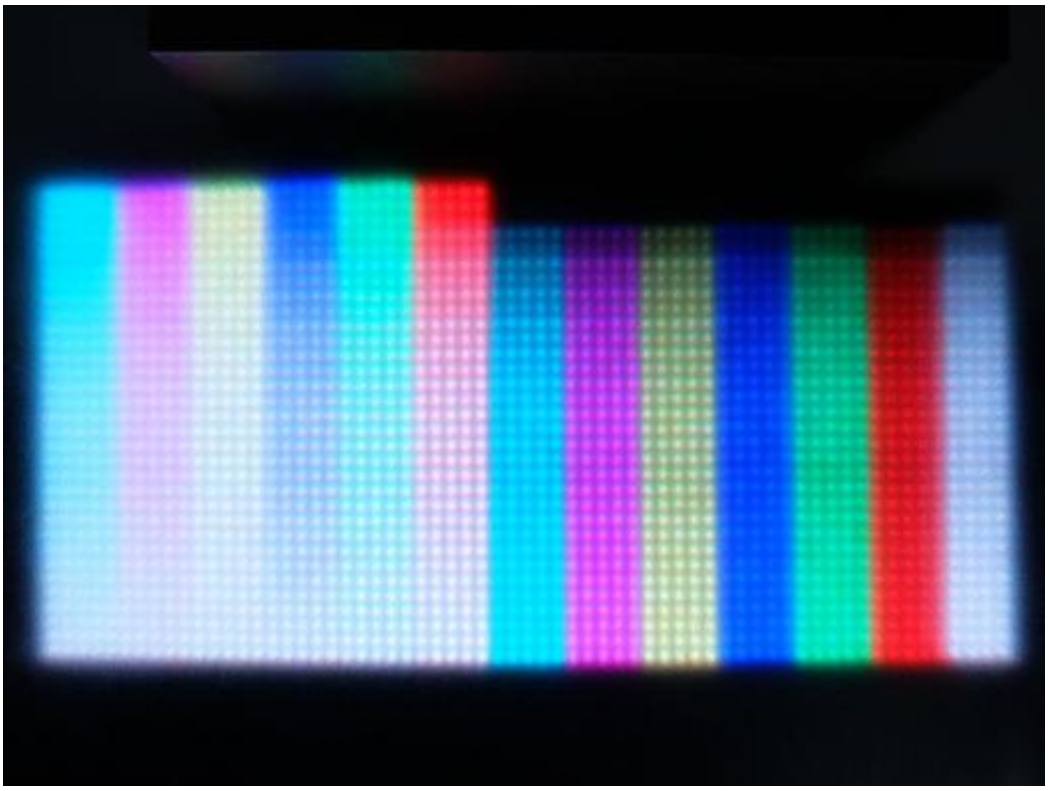
This is it, but the above comment is probably a problem that can be solved by using doub le buffering. However, the RAM capacity is insufficient, so I will leave it alone for now.
I think that the comment below was solved for the time being by setting the interrupt pri ority of TIM2 to the highest.

stm32f1xx_it.c

This is an interrupt handler that is called when an external interrupt occurs.
So, which pin is the external interrupt is the RST pin of the SV3 connector in the circuit di agram above.
RGB data is sent from the microcontroller on the SPI master side, but the microcontroller on the slave side does not know which is the head of the data and how many pieces of da ta it is, so the DMA transfer of SPI is stopped before the start of data transmission, and t he DMA transfer of SPI is started on the assumption that the data to be transmitted is tra nsmitted at once for one frame.

About brightness gradation expression in HUB75 Normally

, each pixel can only be turned on or off in HUB75, and in order to express this luminance gradation, The same frame must be displayed many times while changing the ON time of the LED.
However, simply changing the ON time of the LED in proportion to the brightness informa tion of the pixel does not seem to change the brightness smoothly at all.
It will look like this.



It suddenly becomes bright from a dark place, and it seems that there is almost no chang e in brightness after that.
This is because the sensitivity of the human eye to luminance changes logarithmically. What this means is that when the brightness is low, it seems to be much brighter even if the brightness increases slightly, but when the brightness is high, it seems that there is a lmost no change even if the brightness increases further. ~~In order to offset these charact eristics and make the brightness seem to change smoothly, the brightness must be chan ged exponentially.~~
~~However, it takes a lot of time and is difficult to have a microcomputer perform exponent ial calculations, so this time we decided to change it quadraticly.~~
~~If we want to change it quadraticly,~~
~~we want to make sure that the equation ON time = PWM value ^2~~
~~holds.~~
~~However, in this program, the ON time accumulates more and more, so the actual ON ti me = PWM current value ^ 2 + previous ON time~~
~~is actually~~
~~the case.~~
~~Here, the last ON time is~~
~~Last ON time = PWM previous value ^2 PWM previous value = PWM current value -1~~
~~Therefore,~~
~~the previous ON time = (PWM current value -1)^2~~
~~In other words, the ON time can be calculated~~
~~as ON time = PWM current value ^2 + previous ON time - (PWM current value -1)^2. If you put the equation nicely, it will~~

~~be ON time - Last ON time = 2 * PWM current value -1. In other words, it can be seen th at the value of the automatic reload register ARR of TIM2 should be 2 * PWM current valu e -1, but if this value becomes too small, the interrupt period of TIM2 will be too short an~~

~~d it will not work properly.~~
~~TIM2->ARR = 2*PWM current value + TIM2ARR minimum value~~
~~Leave it as is.~~
~~Also, in this case, we want to change the brightness of the panel with a variable resistor, so we give the luminance coefficient in the form of a coefficient of PWM current value 2 that is included.~~
~~THEN~~
~~TIM2->ARR = PWM current value * luminance factor + TIM2ARR minimum value~~
~~. When the PWM current value * luminance coefficient term is 0, the value of TIM2's automatic reload register ARR becomes constant at the TIM2ARR minimum value regardless of the PWM current value, so~~
~~TIM2->ARR = PWM current value * (luminance factor + 1) + TIM2ARR minimum value~~
~~. Then~~
~~TIM2->ARR = (uint16_t)(pwm_count * (display_brightness_temp + 1) + TIM2_ARR_OFFSET_VALUE);~~
~~It has become the same shape.~~
~~As a matter of fact, by calculating the ON time of TIM2 in this way, the smoothness of the gradation expression is greatly improved.~~
P.S.: It should be corrected by the power of 2.2, not the square. This is because the standard Windows gamma correction value is 2.2.

It is clear when you compare it with the previous image.
When displaying an image on a panel, the contrast changes considerably.

When

the ON time is changed in a first-order function When the
ON time is changed

I feel that it is still quite lacking because I substitute a quadratic function where it should be changed exponentially, but I can see that the contrast has improved considerably.

Well, if the resolution is up to 64*32 with this program, it will work fine at 60 FPS. However, if you connect two 64 * 32 panels in series and set the resolution to 128 * 32, the frame rate drops to about 42 FPS, and the flickering becomes noticeable even with th

e naked eye, which is quite severe.
Here, if you set the MATRIXLED_PWM_RESOLUTION to 8, you will get 60 FPS, although it sacrifices color depth.

This is what it looks like when the color depth is 4 bits.

On the other hand, this is the display when the color depth is 3 bits.

It is easy to understand if you compare the area around the hair, but the number of color s is reduced, and the color reproduction is worse.

In other words, do you take color depth and frame rate at the expense of resolution? Do you take
resolution and color depth at the expense of

frame rate?
There are three patterns of trade-offs. You don't want to sacrifice too much of any of the m if you can.

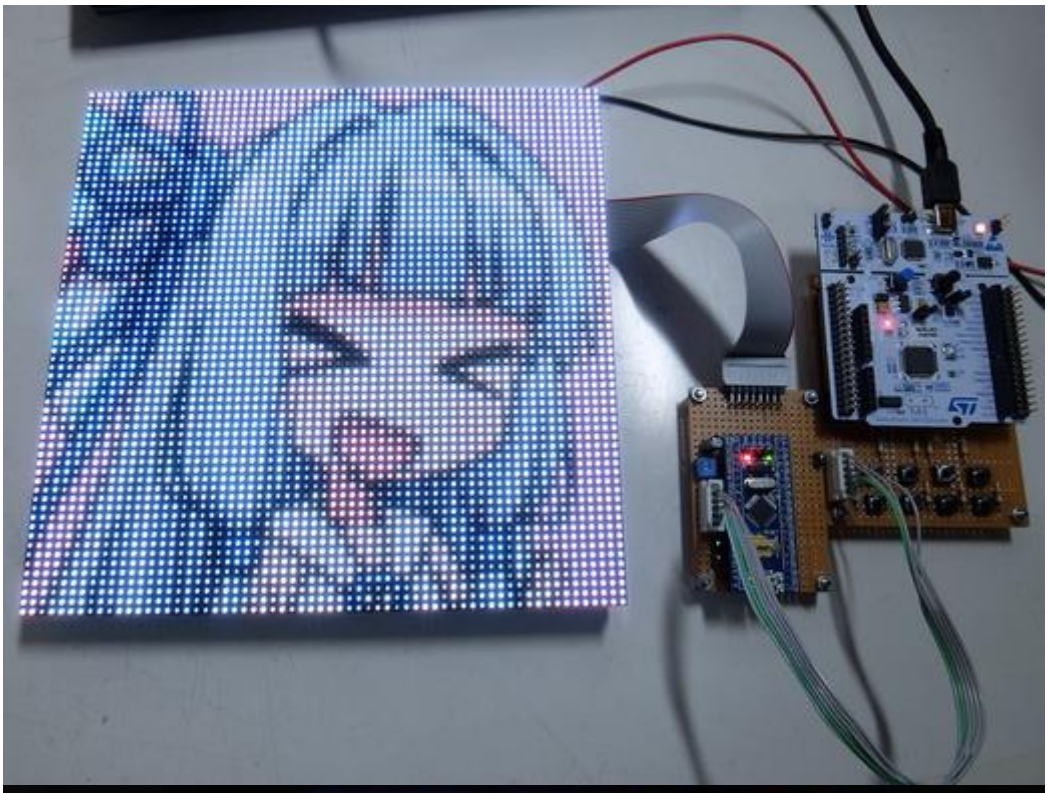
Therefore, I decided to adopt an interlaced method in order to make the flickering as inco nspicuous as possible in the direction of sacrificing the frame rate a little with a painstaki ng measure.
I won't explain interlacing in detail, but by skipping one horizontal scan and displaying it alternately, the field is updated at twice the frequency of the frame rate, so flickering is l ess noticeable even at the same frame rate.

The program excerpts only the parts that have changed.

main.c

The results are relatively good, and the flickering is no longer so noticeable to the naked eye.

With this, we have decided to take a break from creating an SPI interface for a full-color dot matrix LED panel.
It's quite regrettable that I couldn't get 60 FPS in progressive display at 128*32 resolutio n, but it seems that it's impossible at my current level.
Finally, I will fry the project file of the program I created this time.
Download, unzip and import to STM32CubeMX, SW4STM32, etc.
I'm hoping that someone will improve it perfectly.
SPI Interface Progressive Version Project File for HUB75 Full Color Dot Matrix LED Panel S PI
Interface Interlaced Project File for HUB75 Full Color Dot Matrix LED Panel



Postscript: About

R1	1	2	G1
B1	3	4	GND
R2	5	6	G2
B2	7	8	E
A	9	10	B
C	11	12	D
CLK	13	14	LAT
/OE	15	16	GND

First of all, about the role of the E pin of the line decoder, which was added as the chang e from HUB75 to HUB75E, the
vertical resolution is 64 pixels, the scan rate is 1/32, and the line decoderLet me explain assuming that the input pin of is an LED panel that meets the condition
that the 5 pins of ABCDE are present.

In the first place, the role of the line decoder input pin is to select the LED at the corresp onding position
by entering the position of the two lines next to the LED panel you want to display in bina ry numbers.

The relationship between the position of the first row to be selected and the position of th e second row to be selected is as follows.

Position of the second row to be selected = position of the first row to be selected + 1/(s can rate);
Here, from the condition that the above scan rate is 1/32, the position of the second row to
be selected = the position of the first row to be selected + 32;
It comes to.

In addition, the position of the first row to be selected can be expressed by the following equation from the state of the input pin of the line decoder.
However, it is assumed that 0b00000000 is assigned to the A, B, C, D, and E variables w hen the input pin of the line decoder is LOW, and 0b00000001 is assigned to the A, B, C, D, and E variables
, respectively.

Position of the LED in row 1 = (E<<4) | (D<<3) | (C<<2) | (B<<1) | (A<<0);

As you may have noticed here, if you want to display lines 15 and 31 of the LED panel w hen the vertical resolution of the LED panel is 32 pixels and the scan rate is 1/16,
15 = 0b00001111, and the number of input bits of the line decoder is 4 bits.
However, if you want to display
lines 31 and 63 of the LED panel when the vertical resolution of the LED panel is 64 pixel s and the scan rate is 1/32, 31 = 0b00011111 and you can see
that the number of input bits of the line decoder is 5 bits I think.
For this purpose, the E pin of the line decoder was added.

Here's what you need to change and what you need to do to make the program in the blo
g post compatible with the HUB75E.

Before

and after

the
change. If you rewrite it like Arduino,
it
will be .
In addition, since Arduino's digitalWrite is very slow, it seems that it will actually be nece
ssary to hit the register of the microcontroller directly.
(In the program before the change, the speed is increased by directly hitting the STM32 r
egister, and the readability is hell.)
I'm sorry for
my poor explanation, but the sentence is quite difficult to understand.
The following site explains in great detail how to control the HUB75, so if you take a look
at that site, you may be able to deepen your understanding of the HUB75 and HUB75E, i
ncluding the
internal aspects.
LED display display experiment using the DMA function of PIC32

```
/* 略 ------------------------------------------------------------------------*/

/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdlib.h>
/* USER CODE END Includes */

/* 略 ------------------------------------------------------------------------*/

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
#define MATRIXLED_R1(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, x))
#define MATRIXLED_G1(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, x))
#define MATRIXLED_B1(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, x))
#define MATRIXLED_R2(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, x))
#define MATRIXLED_G2(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, x))
#define MATRIXLED_B2(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, x))
#define MATRIXLED_ADDR_A(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, x))
#define MATRIXLED_ADDR_B(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, x))
```

```
#define MATRIXLED_ADDR_C(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, x))
#define MATRIXLED_ADDR_D(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, x))
#define MATRIXLED_CLOCK(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, x))
#define MATRIXLED_STROBE(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, x))
#define MATRIXLED_OUTENABLE(x) (HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, x))

#define ONBOARD_LED(x) (HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, x))

#define LED_RGB(x) {\\
        GPIOB->BRR = (0x3f << 3);\\
        GPIOB->BSRR = ((x & 0x3f) << 3);\\
}

#define LED_ADDR(x) {\\
        GPIOB->BRR = (0x0f << 12);\\
        GPIOB->BSRR = ((x & 0x0f) << 12);\\
}

#define LED_CLK_HIGH {\\
        GPIOB->BSRR = (1 << 9);\\
}

#define LED_CLK_LOW {\\
        GPIOB->BRR = (1 << 9);\\
}

#define LED_STB_HIGH {\\
        GPIOB->BSRR = (1 << 10);\\
}

#define LED_STB_LOW {\\
        GPIOB->BRR = (1 << 10);\\
}

#define LED_OE_HIGH {\\
        GPIOB->BSRR = (1 << 11);\\
}

#define LED_OE_LOW {\\
```

```
        GPIOB->BRR = (1 << 11);\\
}

#define ONBOARD_LED_TOGGLE {\\
        GPIOC->ODR ^= (1 << 13);\\
}

#define HIGH 1
#define LOW 0

#define COLOR_R 0
#define COLOR_G 1
#define COLOR_B 2

#define R1_SET 0x0001
#define G1_SET 0x0002
#define B1_SET 0x0004
#define R2_SET 0x0008
#define G2_SET 0x0010
#define B2_SET 0x0020

#define MATRIXLED_X_COUNT 64
#define MATRIXLED_Y_COUNT 32
#define MATRIXLED_COLOR_COUNT 3

#define MATRIXLED_MAX_Y_ADDRES 16

#define MATRIXLED_PWM_RESOLUTION 16

#define ADC_DIVIDE_VALUE 445
#define TIM2_ARR_OFFSET_VALUE 5

unsigned char
Flame_Buffer[MATRIXLED_Y_COUNT][MATRIXLED_X_COUNT][MATRIXLED_COLOR_COUNT];

volatile unsigned char TIM2_Updated_Flag, TIM3_Updated_Flag;

/* USER CODE END PV */

/* 略 ----------------------------------------------------------------*/
```

```
/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
void fill_flame_buffer_color(unsigned char, unsigned char, unsigned char);
void fill_flame_buffer_random(void);
void fill_flame_buffer_testpattern(void);

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  *
  * @retval None
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
        unsigned short matrixled_X_loop_count, matrixled_Y_loop_count;
        unsigned char pwm_count;
        unsigned short led_rgb_temp;

        unsigned short ADC_value;

        float display_brightness_temp;

  /* USER CODE END 1 */

/* 略 ------------------------------------------------------------------------*/

  /* USER CODE BEGIN 2 */
  fill_flame_buffer_testpattern();

  HAL_SPI_Receive_DMA(&hspi1, Flame_Buffer, (MATRIXLED_Y_COUNT *
MATRIXLED_X_COUNT * MATRIXLED_COLOR_COUNT));
  HAL_TIM_Base_Start_IT(&htim3);
```

```
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
        HAL_ADC_Start_DMA(&hadc1, &ADC_value, 1);
        display_brightness_temp = (float)ADC_value / ADC_DIVIDE_VALUE;
              for(pwm_count = 1;pwm_count <
MATRIXLED_PWM_RESOLUTION;pwm_count++){
                        for(matrixled_Y_loop_count = 0;matrixled_Y_loop_count <
MATRIXLED_MAX_Y_ADDRES;matrixled_Y_loop_count++){

                                if(matrixled_Y_loop_count == 1) TIM2->ARR =
(uint16_t)(pwm_count * (display_brightness_temp + 1) + TIM2_ARR_OFFSET_VALUE);
                                TIM2_Updated_Flag = RESET;
                                __HAL_TIM_ENABLE_IT(&htim2, TIM_IT_UPDATE);
                                __HAL_TIM_ENABLE(&htim2);
                                LED_OE_LOW;

                                for(matrixled_X_loop_count =
0;matrixled_X_loop_count < MATRIXLED_X_COUNT;matrixled_X_loop_count++){
                                        led_rgb_temp = 0;


        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_R] >=
pwm_count) led_rgb_temp |= R1_SET;

        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_G] >=
pwm_count) led_rgb_temp |= G1_SET;

        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_B] >=
pwm_count) led_rgb_temp |= B1_SET;

                                        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_R] >= pwm_count)
led_rgb_temp |= R2_SET;
```

```
                                        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_G] >= pwm_count)
led_rgb_temp |= G2_SET;
                                        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_B] >= pwm_count)
led_rgb_temp |= B2_SET;


                                        LED_RGB(led_rgb_temp);


                                        LED_CLK_HIGH;
                                        LED_CLK_LOW;
                                }
                                while(TIM2_Updated_Flag == RESET);

                                LED_ADDR(matrixled_Y_loop_count);
                                LED_STB_HIGH;
                                LED_STB_LOW;

                                //フレームバッファにSPIからDMA転送している最中に読み
だそうとすると一瞬フリーズする問題
                                //TIM2の割り込みがたまに他の割り込みとバッティングす
る問題
                        }
                }
                while(TIM3_Updated_Flag == RESET);
                TIM3_Updated_Flag = RESET;

                ONBOARD_LED_TOGGLE;
  }
  /* USER CODE END 3 */

}

/* 略 ----------------------------------------------------------------------*/

/* USER CODE BEGIN 4 */
void fill_flame_buffer_color(unsigned char red, unsigned char green, unsigned char
blue){
        unsigned char red_temp, green_temp, blue_temp;
        unsigned char fill_buffer_loop_x, fill_buffer_loop_y;
```

```
        if(red < MATRIXLED_PWM_RESOLUTION) red_temp = red;
        else red_temp = MATRIXLED_PWM_RESOLUTION - 1;

        if(green < MATRIXLED_PWM_RESOLUTION) green_temp = green;
        else green_temp = MATRIXLED_PWM_RESOLUTION - 1;

        if(blue < MATRIXLED_PWM_RESOLUTION) blue_temp = blue;
        else blue_temp = MATRIXLED_PWM_RESOLUTION - 1;

        for(fill_buffer_loop_y = 0;fill_buffer_loop_y <
MATRIXLED_Y_COUNT;fill_buffer_loop_y++){
                for(fill_buffer_loop_x = 0;fill_buffer_loop_x <
MATRIXLED_X_COUNT;fill_buffer_loop_x++){

        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_R] = red_temp;

        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_G] =
green_temp;

        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_B] = blue_temp;
                }
        }
}

void fill_flame_buffer_random(void){
        unsigned char fill_buffer_loop_x, fill_buffer_loop_y;

        srand(rand());

        for(fill_buffer_loop_y = 0;fill_buffer_loop_y <
MATRIXLED_Y_COUNT;fill_buffer_loop_y++){
                for(fill_buffer_loop_x = 0;fill_buffer_loop_x <
MATRIXLED_X_COUNT;fill_buffer_loop_x++){

        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_R] = rand() %
MATRIXLED_PWM_RESOLUTION;

        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_G] = rand() %
MATRIXLED_PWM_RESOLUTION;

        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_B] = rand() %
MATRIXLED_PWM_RESOLUTION;
                }
        }
```

```
}


void fill_flame_buffer_testpattern(void){
        unsigned char fill_buffer_loop_x, fill_buffer_loop_y;


        for(fill_buffer_loop_y = 0;fill_buffer_loop_y <
MATRIXLED_Y_COUNT;fill_buffer_loop_y++){
                for(fill_buffer_loop_x = 0;fill_buffer_loop_x <
(MATRIXLED_X_COUNT / 12);fill_buffer_loop_x++){


        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_R] =
fill_buffer_loop_y * (MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;


        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_G] =
(MATRIXLED_PWM_RESOLUTION - 1);


        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x][COLOR_B] =
(MATRIXLED_PWM_RESOLUTION - 1);


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12)][COLOR_R] = (MATRIXLED_PWM_RESOLUTION - 1);
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12)][COLOR_G] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12)][COLOR_B] = (MATRIXLED_PWM_RESOLUTION - 1);


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 2][COLOR_R] = (MATRIXLED_PWM_RESOLUTION - 1);
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 2][COLOR_G] = (MATRIXLED_PWM_RESOLUTION - 1);
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 2][COLOR_B] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 3][COLOR_R] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 3][COLOR_G] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 3][COLOR_B] = (MATRIXLED_PWM_RESOLUTION - 1);
```

```
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 4][COLOR_R] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 4][COLOR_G] = (MATRIXLED_PWM_RESOLUTION - 1);

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 4][COLOR_B] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 5][COLOR_R] = (MATRIXLED_PWM_RESOLUTION - 1);

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 5][COLOR_G] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 5][COLOR_B] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 6][COLOR_R] = 0;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 6][COLOR_G] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 6][COLOR_B] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 7][COLOR_R] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 7][COLOR_G] = 0;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 7][COLOR_B] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 8][COLOR_R] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 8][COLOR_G] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;

                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 8][COLOR_B] = 0;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 9][COLOR_R] = 0;
```

```
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 9][COLOR_G] = 0;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 9][COLOR_B] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 10][COLOR_R] = 0;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 10][COLOR_G] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 10][COLOR_B] = 0;


                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 11][COLOR_R] = fill_buffer_loop_y *
(MATRIXLED_PWM_RESOLUTION - 1) / MATRIXLED_Y_COUNT;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 11][COLOR_G] = 0;
                        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x +
(MATRIXLED_X_COUNT / 12) * 11][COLOR_B] = 0;


                        if(fill_buffer_loop_x < (MATRIXLED_X_COUNT % 12)){

        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x + (MATRIXLED_X_COUNT /
12) * 12][COLOR_R] = fill_buffer_loop_y * (MATRIXLED_PWM_RESOLUTION - 1) /
MATRIXLED_Y_COUNT;


        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x + (MATRIXLED_X_COUNT /
12) * 12][COLOR_G] = fill_buffer_loop_y * (MATRIXLED_PWM_RESOLUTION - 1) /
MATRIXLED_Y_COUNT;


        Flame_Buffer[fill_buffer_loop_y][fill_buffer_loop_x + (MATRIXLED_X_COUNT /
12) * 12][COLOR_B] = fill_buffer_loop_y * (MATRIXLED_PWM_RESOLUTION - 1) /
MATRIXLED_Y_COUNT;
                        }
                }
        }
}


/* USER CODE END 4 */


/* 略 ----------------------------------------------------------------------*/
/* 略 ----------------------------------------------------------------------*/
```

```
/* USER CODE BEGIN 0 */
#include <stdlib.h>

#define LED_OE_HIGH {\\
        GPIOB->BSRR = (1 << 11);\\
}

#define LED_OE_LOW {\\
        GPIOB->BRR = (1 << 11);\\
}

#define ONBOARD_LED_TOGGLE {\\
        GPIOC->ODR ^= (1 << 13);\\
}

#define MATRIXLED_X_COUNT 64
#define MATRIXLED_Y_COUNT 32
#define MATRIXLED_COLOR_COUNT 3

extern unsigned char
Flame_Buffer[MATRIXLED_Y_COUNT][MATRIXLED_X_COUNT][MATRIXLED_COLOR_COUNT];
extern unsigned char TIM2_Updated_Flag, TIM3_Updated_Flag;

/* USER CODE END 0 */

/* 略 ---------------------------------------------------------------------*/

/**
* @brief This function handles EXTI line0 interrupt.
*/
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
        HAL_SPI_DMAStop(&hspi1);
        HAL_SPI_Receive_DMA(&hspi1, Flame_Buffer, (MATRIXLED_Y_COUNT *
MATRIXLED_X_COUNT * MATRIXLED_COLOR_COUNT));

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */
```

```
  /* USER CODE END EXTI0_IRQn 1 */
}

/* 略 ----------------------------------------------------------------------*/

/**
* @brief This function handles TIM2 global interrupt.
*/
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */
        if(__HAL_TIM_GET_FLAG(&htim2, TIM_FLAG_UPDATE) != RESET){
                LED_OE_HIGH;
                TIM2_Updated_Flag = SET;
        }

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */
  if(TIM2_Updated_Flag == SET){
          HAL_TIM_Base_Stop_IT(&htim2);
  }

  /* USER CODE END TIM2_IRQn 1 */
}

/**
* @brief This function handles TIM3 global interrupt.
*/
void TIM3_IRQHandler(void)
{
  /* USER CODE BEGIN TIM3_IRQn 0 */
        if(__HAL_TIM_GET_FLAG(&htim3, TIM_FLAG_UPDATE) != RESET){
                TIM3_Updated_Flag = SET;
        }

  /* USER CODE END TIM3_IRQn 0 */
  HAL_TIM_IRQHandler(&htim3);
```

```
  /* USER CODE BEGIN TIM3_IRQn 1 */

  /* USER CODE END TIM3_IRQn 1 */
}

/**
* @brief This function handles SPI1 global interrupt.
*/
void SPI1_IRQHandler(void)
{
  /* USER CODE BEGIN SPI1_IRQn 0 */
        //SPI通信にてエラーが発生した時にここに飛ぶためエラー処理を行う必要がある
        ONBOARD_LED_TOGGLE;

  /* USER CODE END SPI1_IRQn 0 */
  HAL_SPI_IRQHandler(&hspi1);
  /* USER CODE BEGIN SPI1_IRQn 1 */

  /* USER CODE END SPI1_IRQn 1 */
}

/* 略 ------------------------------------------------------------------------*/
#define MATRIXLED_X_COUNT 64
#define MATRIXLED_Y_COUNT 32
#define MATRIXLED_COLOR_COUNT 3

#define MATRIXLED_MAX_Y_ADDRES 16

#define MATRIXLED_PWM_RESOLUTION 16

#define ADC_DIVIDE_VALUE 445
#define TIM2_ARR_OFFSET_VALUE 5
#define MATRIXLED_X_COUNT 64
#define MATRIXLED_Y_COUNT 32
#define MATRIXLED_COLOR_COUNT 3
void fill_flame_buffer_color(unsigned char, unsigned char, unsigned char);
void fill_flame_buffer_random(void);
void fill_flame_buffer_testpattern(void);
unsigned char
Flame_Buffer[MATRIXLED_Y_COUNT][MATRIXLED_X_COUNT][MATRIXLED_COLOR_COUNT];
```

```
Flame_Buffer[0][0][0] = 0x0f;
Flame_Buffer[0][0][1] = 0x0f;
Flame_Buffer[0][0][2] = 0x0f;
if(matrixled_Y_loop_count == 1) TIM2->ARR = (uint16_t)(pwm_count *
(display_brightness_temp + 1) + TIM2_ARR_OFFSET_VALUE);
for(matrixled_X_loop_count = 0;matrixled_X_loop_count <
MATRIXLED_X_COUNT;matrixled_X_loop_count++){
        led_rgb_temp = 0;

        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_R] >=
pwm_count) led_rgb_temp |= R1_SET;
        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_G] >=
pwm_count) led_rgb_temp |= G1_SET;
        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_B] >=
pwm_count) led_rgb_temp |= B1_SET;

        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_R] >= pwm_count)
led_rgb_temp |= R2_SET;
        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_G] >= pwm_count)
led_rgb_temp |= G2_SET;
        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_B] >= pwm_count)
led_rgb_temp |= B2_SET;

        LED_RGB(led_rgb_temp);

        LED_CLK_HIGH;
        LED_CLK_LOW;
}
//フレームバッファにSPIからDMA転送している最中に読みだそうとすると一瞬フリーズする問題
//TIM2の割り込みがたまに他の割り込みとバッティングする問題
/**
* @brief This function handles EXTI line0 interrupt.
*/
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
        HAL_SPI_DMAStop(&hspi1);
        HAL_SPI_Receive_DMA(&hspi1, Flame_Buffer, (MATRIXLED_Y_COUNT *
MATRIXLED_X_COUNT * MATRIXLED_COLOR_COUNT));
```

```
  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}
  /* USER CODE BEGIN WHILE */
  while (1)
  {

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
        HAL_ADC_Start_DMA(&hadc1, &ADC_value, 1);
        display_brightness_temp = (float)ADC_value / ADC_DIVIDE_VALUE;
              for(pwm_count = 1;pwm_count <
MATRIXLED_PWM_RESOLUTION;pwm_count++){
                      for(matrixled_Y_loop_count =
interlace_count;matrixled_Y_loop_count <
MATRIXLED_MAX_Y_ADDRES;matrixled_Y_loop_count += 2){

                                if((matrixled_Y_loop_count == 2) ||
(matrixled_Y_loop_count == 3)) TIM2->ARR = (uint16_t)(pwm_count *
(display_brightness_temp + 1) + TIM2_ARR_OFFSET_VALUE);
                                TIM2_Updated_Flag = RESET;
                                __HAL_TIM_ENABLE_IT(&htim2, TIM_IT_UPDATE);
                                __HAL_TIM_ENABLE(&htim2);
                                LED_OE_LOW;

                                for(matrixled_X_loop_count =
0;matrixled_X_loop_count < MATRIXLED_X_COUNT;matrixled_X_loop_count++){
                                        led_rgb_temp = 0;


        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_R] >=
pwm_count) led_rgb_temp |= R1_SET;

        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_G] >=
pwm_count) led_rgb_temp |= G1_SET;

        if(Flame_Buffer[matrixled_Y_loop_count][matrixled_X_loop_count][COLOR_B] >=
pwm_count) led_rgb_temp |= B1_SET;
```

```
                                        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_R] >= pwm_count)
led_rgb_temp |= R2_SET;
                                        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_G] >= pwm_count)
led_rgb_temp |= G2_SET;
                                        if(Flame_Buffer[matrixled_Y_loop_count +
MATRIXLED_MAX_Y_ADDRES][matrixled_X_loop_count][COLOR_B] >= pwm_count)
led_rgb_temp |= B2_SET;

                                        LED_RGB(led_rgb_temp);

                                        LED_CLK_HIGH;
                                        LED_CLK_LOW;
                                }
                                while(TIM2_Updated_Flag == RESET);

                                LED_ADDR(matrixled_Y_loop_count);
                                LED_STB_HIGH;
                                LED_STB_LOW;

                                //フレームバッファにSPIからDMA転送している最中に読み
だそうとすると一瞬フリーズする問題

                                //TIM2の割り込みがたまに他の割り込みとバッティングす
る問題
                        }
                }
                while(TIM3_Updated_Flag == RESET);
                TIM3_Updated_Flag = RESET;

                ONBOARD_LED_TOGGLE;

                if(interlace_count == 0) interlace_count++;
                else interlace_count = 0;
  }
  /* USER CODE END 3 */
#define LED_ADDR(x) {\\
        GPIOB->BRR = (0x0f << 12);\\
        GPIOB->BSRR = ((x & 0x0f) << 12);\\
}
#define LED_ADDR(x) {\\
```

```
        HAL_GPIO_WritePin(/*Aが接続されているポート*/, /*Aが接続されているピン*/,
(((x) >> 0) & 0x01));\\
        HAL_GPIO_WritePin(/*Bが接続されているポート*/, /*Bが接続されているピン*/,
(((x) >> 1) & 0x01));\\
        HAL_GPIO_WritePin(/*Cが接続されているポート*/, /*Cが接続されているピン*/,
(((x) >> 2) & 0x01));\\
        HAL_GPIO_WritePin(/*Dが接続されているポート*/, /*Dが接続されているピン*/,
(((x) >> 3) & 0x01));\\
        HAL_GPIO_WritePin(/*Eが接続されているポート*/, /*Eが接続されているピン*/,
(((x) >> 4) & 0x01));\\
}
#define LED_ADDR(x) {\\
        digitalWrite(/*Aが接続されているピン*/, (((x) >> 0) & 0x01));\\
        digitalWrite(/*Bが接続されているピン*/, (((x) >> 1) & 0x01));\\
        digitalWrite(/*Cが接続されているピン*/, (((x) >> 2) & 0x01));\\
        digitalWrite(/*Dが接続されているピン*/, (((x) >> 3) & 0x01));\\
        digitalWrite(/*Eが接続されているピン*/, (((x) >> 4) & 0x01));\\
}
```