Драйвер дисплея

Драйвера max7219 не достаточно, чтобы отображать нужные нам данные, он лишь предоставляет нам API микросхемы, абстрагирует нас от понятия SPI, когда и какую ножку нужно выставлять в высокий уровень и т.д. Напишем теперь драйвер дисплея `display.c` / `display.h`.

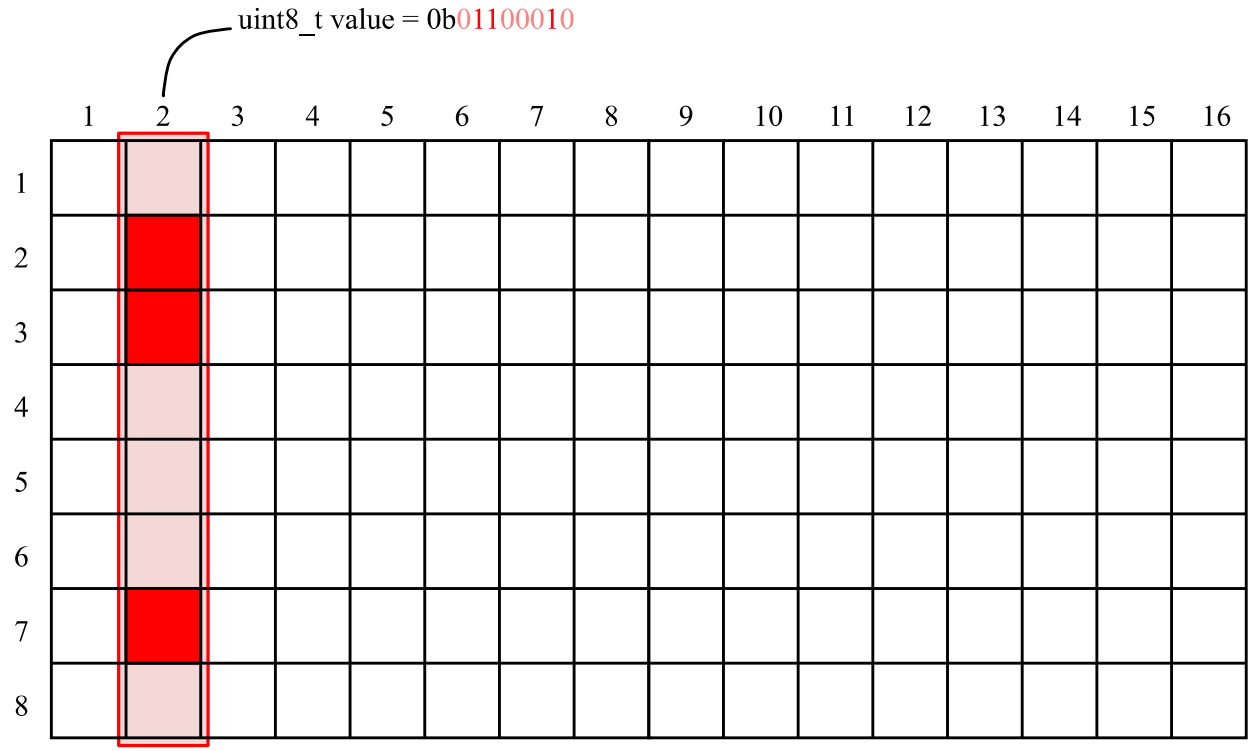
Давайте условимся, что то что отображается на дисплее и то, что мы храним в памяти МК не всегда совпадает. Оно и правильно: сначала записываем в буфер всё новое состояние дисплея, а уже после этого отправляем данные на него. То есть кроме понятных нам функций `display_int()` и `display_set_brightness()` потребуется функция обновления показаний дисплея `display_update()`. Все остальные функции должны работать с буфером.

Нам безусловно понадобиться функция очистки, `display_clear()`, а так же отрисовки некоторых примитивов: точки, чисел, линий, прямоугольников. Составим API.

```
#ifndef __DISPLAY_H__
#define __DISPLAY_H__

#include "stm32f10x.h"
#include "max7219.h"
#include "utils.h"

typedef enum {
    DOT_OFF = 0,
    DOT_ON  = !DOT_OFF,
} DOT_STATE_t;

void display_init(const uint8_t br);
void display_set_brightness(const uint8_t br);
void display_update(void);
void display_clear(void);
void display_clear_and_update(void);
void display_fill(void);
void display_draw_dot(const uint8_t x, const uint8_t y, const DOT_STATE_t state);
void display_draw_number(uint8_t x, uint8_t y, uint8_t num);
void display_draw_v_line(const uint8_t y,
                         const uint8_t x0,
                         const uint8_t x1,
                         const DOT_STATE_t state);
void display_draw_h_line(const uint8_t x,
                         const uint8_t y0,
                         const uint8_t y1,
                         const DOT_STATE_t state);
void display_draw_rect(const uint8_t x0,
                       const uint8_t y0,
                       const uint8_t x1,
                       const uint8_t y1,
                       const DOT_STATE_t state);
void display_animation_intro(void);
```

```
#endif /* __DISPLAY_H__ */
```

Здесь нужно обратить внимание на то, как данные хранятся и как они передаются.



Т.е. для буфера нам потребуется создать массив из `uint8_t` элементов.

```
#define MATRIX_X_SIZE       16
#define MATRIX_Y_SIZE       8

uint8_t display_buffer[MATRIX_X_SIZE] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // matrix 1: [0 -  7] cols
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // matrix 2: [8 - 15] cols
};
```

Из файла `main.c` модуль `max7219.c/.h` виден не будет, следовательно работать с микросхемой мы должны напрямую из драйвера дисплея.

```
void display_init(const uint8_t br) {
    max7219_init(br);
    display_animation_intro();
    display_clear_and_update();
}

void display_set_brightness(const uint8_t br) {
    max7219_set_brightness(br);
}

void display_update(void) {
    for (uint32_t col; col < MATRIX_Y_SIZE; col++)
        max7219_send(REG_DIGIT_0 + col,
            display_buffer[col                ],
```

```
            display_buffer[cols + MATRIX_Y_SIZE]);
}
```

Приступим к работе с буфером. Для очистки дисплея, очевидно, нужно просто обнулить все значения в буфере.

```
void display_clear(void) {
    for (uint32_t col = 0; col < MATRIX_Y_SIZE; col++)
        display_buffer[col] = 0x00;
}

void display_clear_and_update(void) {
    display_clear();
    display_update();
}
```

Если вдруг нам потребуется сделать обратную операцию очистки, то придётся воспользоваться функцией заполнения.

```
void display_fill(void) {
    display_draw_rect(0, MATRIX_X_SIZE, 0, MATRIX_Y_SIZE, DOT_ON);
}
```

Частный случай закрасить экран полностью — нарисовать прямоугольник во весь размер экрана. Не зачем дублировать функционал. Более того, частный случай прямоугольника в нашем случае, это линия.

```
void display_draw_v_line(const uint8_t y,
                         const uint8_t x0,
                         const uint8_t x1,
                         const DOT_STATE_t state) {
    display_draw_rect(x0, y, x1, y, state);
}

void display_draw_h_line(const uint8_t x,
                         const uint8_t y0,
                         const uint8_t y1,
                         const DOT_STATE_t state) {
    display_draw_rect(x, y0, x, y1, state);
}
```

Точка так же является частным случаем прямоугольника.

```
void display_draw_dot(const uint8_t x, const uint8_t y, const DOT_STATE_t state) {
    display_draw_rect(x, y, x, y, state);
}
```

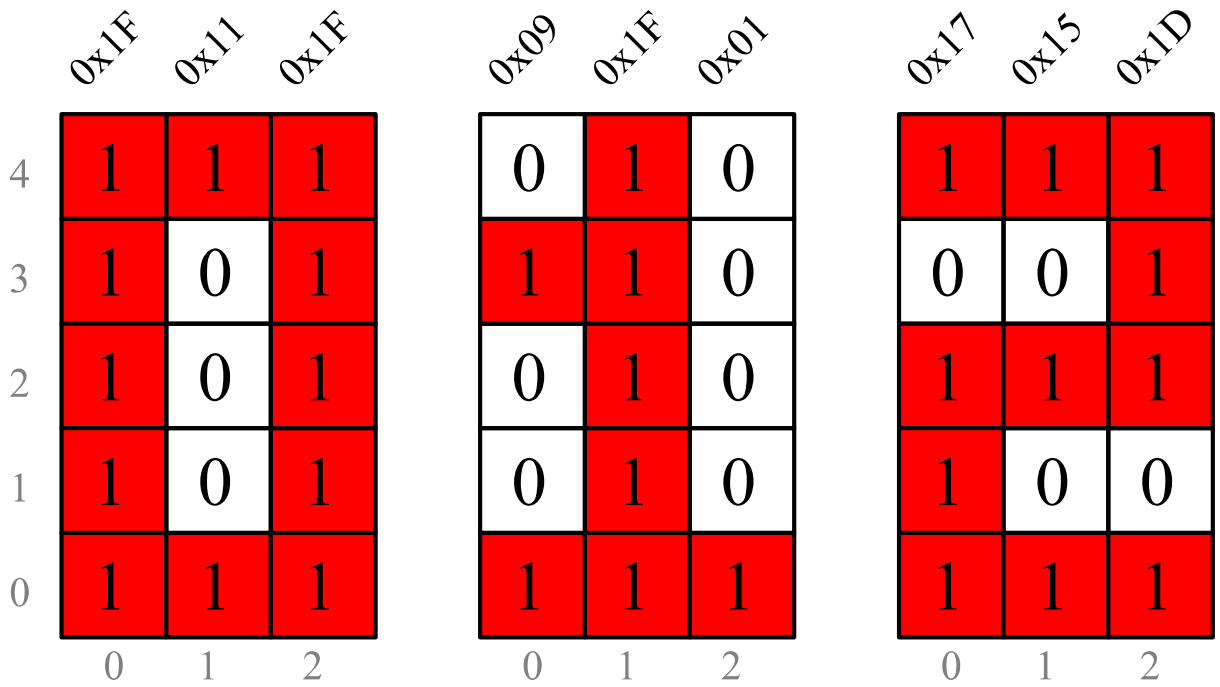
Таким образом большинство функций зависит только от одной единственной, `display_draw_rect()`. Составим её.

```
#define is_valid(x,y)       ((x < MATRIX_X_SIZE) && (y < MATRIX_Y_SIZE))

void display_draw_rect(const uint8_t x0, const uint8_t y0,
                       const uint8_t x1, const uint8_t y1,
                       const DOT_STATE_t state) {
```

```
    if ( (x0 > x1) || (y0 > y1) || !is_valid(x0,y0) || !is_valid(x1,y1) )
        return;

    uint16_t mask = 0;
    for (uint32_t i = y0; i < y1; i++)
        mask += (1 << i);

    for (uint32_t x = x0; x < (x0 + x1); x++) {
        display_buffer[x] = (state == DOT_ON) ?
                            display_buffer[x] | mask :
                            display_buffer[x] & ~mask;
    }
}
```

Так как отрисовка происходит по столбикам, то нам нужно вычислить маску (`mask`), в которой по сути просто стоят `1` на нужных нам позициях. Нужно зажечь первый светодиод в столбике? Записываем `1`. Нужно зажечь третий? Просто смещаем на 2 позиции `1` и получаем 8.

Для отображения чисел, однако, нам потребуется отдельная функция. По сути, нам нужно создать массив из трёх значений для каждой цифры (от 0 до 9).



Попробуйте самостоятельно найти значения для остальных чисел (3, 4, 5, 6, 7, 8, 9).

Составим функцию отображении цифр.

```
void display_draw_number(uint8_t x, uint8_t y, uint8_t num) {
    static const uint8_t numbers[][] = {
        { 0x1F, 0x11, 0x1F, }, // 0
        { 0x09, 0x1F, 0x01, }, // 1
        { 0x17, 0x15, 0x1D, }, // 2
        // ...
    };
    if ((number > 10) || ((x + 3) > 16) || ((y + 5) > 8))
```

```
        return;

    for (uint32_t col = x; col < (x + 3); col++) {
        // clear and fill
        display_buffer[col] &= ~(0x1F << x);
        display_buffer[col] |= (numbers[num][col] << x);
    }
}
```

Добавим функцию отображения времени.

```
void dislay_show_time(const uint8_t hh, const uint8_t mm) {
    display_draw_number( 0, 0, hh / 10);
    display_draw_number( 4, 0, hh % 10);
    display_draw_number( 9, 0, mm / 10);
    display_draw_number(13, 0, mm % 10);
    display_update();
}
```

И вызовим её:

```
int main(void) {
    mcu_init(); // --> display_init(5);

    while(1) {
        display_show_time(rtc_get_time().hh, rtc_get_time().mm);
    }
}
```

Что же касается функции `display_animation_intro()` — оставьте её пустой или закомментируйте её вызов в функции инициализации. Вернитесь к ней после следующей главы. К слову, какую анимацию сделать — дело ваше, в стоковой прошивке это зажигание светодиодов матриц по спирали. Попробуйте реализовать данный алгоритм самостоятельно, а если не получится подсмотрите решение на Rosetta Code — **Spiral matrix**.

Код можно найти на GitHub: **CMSIS**.

Назад | **Оглавление** | **Дальше**