Драйвер MAX7219

Реализовать работу матричного индикатора на этапе проектирования можно было по-разному. Довольно часто для этого используют сдвиговый регистр, однако он не удовлетворил нас из-за своих параметров (в случае его использования микросхема работала бы за пределами своих возможностей, но об этом в самом конце курса). Наиболее простым способом оказалось использование специализированной микросхемы-драйвера MAX7219. Работает она по и интерфейсу SPI, т.е. для управления 64 светодиодами (матрица 8 * 8) требуется всего 3 ножки микроконтроллера. В устройстве матрицы две и подключены они по каскадной схеме, нам не нужна дополнительная ножка выбора микросхемы.

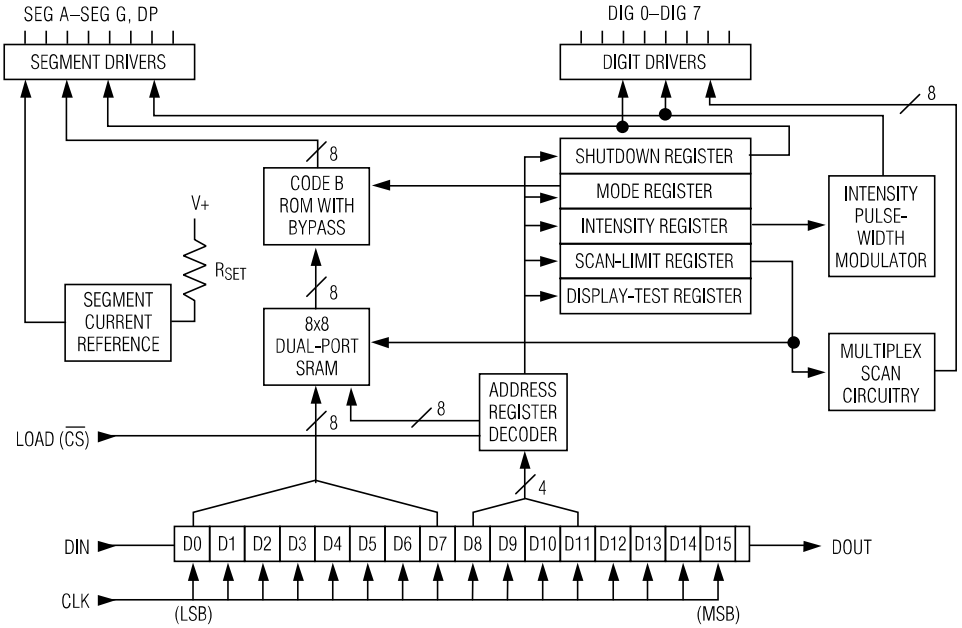
Как и в случае с датчиком температуры, нам необходимо изучить **документацию** для того чтобы понять, как работает устройство. Итак, постарайтесь найти все необходимые нам данные.

- Какова максимальная частота работы микросхемы?
- Каков размер посылки (в битах)?
- Какой бит старший?
- Как работает ножка выбора устройства CS (LOAD)?

Итак, если вы это проделали, то ваш ответ совпадет с тем, что изложено ниже:

▶ Ответ

Структурная схема поможет нам разобраться.



Структурная схема микросхемы max7219, изображение из документации

Пакет включает в себя «адрес» (в действительности это лишь указание декодеру, чтобы микросхема поняла, по какому адресу вы хотите записать данные) регистра, куда будут записываться данные, и непосредственно сами данные. Так, например, чтобы изменить яркость, вам необходимо отправить декодеру значение `0xXA`. Для начала составим перечисление всех «адресов» (согласно таблице 2 из документации).

```
// max7219.h
```

```
#define REG_NO_OP            0x00
#define REG_DIGIT_0          0x01
#define REG_DIGIT_1          0x02
#define REG_DIGIT_2          0x03
#define REG_DIGIT_3          0x04
#define REG_DIGIT_4          0x05
#define REG_DIGIT_5          0x06
#define REG_DIGIT_6          0x07
#define REG_DIGIT_7          0x08
#define REG_DECODE_MODE      0x09
#define REG_INTENSITY        0x0A
#define REG_SCAN_LIMIT       0x0B
#define REG_SHUTDOWN         0x0C
#define REG_DISPLAY_TEST     0x0F
```

> Если нам нужно передать данные только на вторую микросхему, то сначала мы загоняем данные для второй микросхемы и следом за ней отправляем пустой блок `REG_NO_OP`.

Ножки DIG0 - DIG7 микросхемы подключаются к катодам (столбцы) матрицы, а SEGA - SEGDP к анодам (строки). Следовательно данные, которые мы записываем в регистры `REG_DIGIT_0` - `REG_DIGIT_7` определяют состояние состояние столбцов. Данные могут быть интерпретированы по разному, в зависимости от выбранного режима (регистр `REG_DECODE_MODE`): в режиме «No decode» каждый бит соответствует определенному светодиоду, т.е. записав `0` — светодиод не горит, записав `1` — светодиод горит; в режиме «BCD code B» биты, записанные в регистре `REG_DIGIT_`, кодируют определенный символ, указанный в документации. Последний режим больше подходит для семисегментных индикаторов.

Яркость индикатора регулируется регулируется регистром `REG_INTENSITY`. Всего имеется 16 уровней, 0 соответствует самому тусклому свечению, а 15 самому яркому.

Регистр `REG_SCAN_LIMIT` отвечает за число обслуживаемых линий DIG. Чем больше линий, тем меньше частота обновления. Если используются все линии (наш случай), то частота составляет 800 Гц, если меньше, то 8 x 800/N, где N — количество задействованных линий.

Записав в регистр `REG_SHUTDOWN` «1», вы включаете «нормальный» режим работы. Перейдём к инициализации: включим тактирование порта и периферии; настроим ножки и SPI в соответствии с документацией; включим периферию; зададим начальную интенсивность индикатора и очистим его.

Составим заголовочный файл модуля max7219.

```
#ifndef __MAX7219_H__
#define __MAX7219_H__

#include "stm32f10x.h"

/* REGs */

#define MAX7219_CHIPS        2

void max7219_init(const uint8_t br);
void max7219_set_brightness(const uint8_t br);
void max7219_clear(void);
```

```
void max7219_send_to_all(const uint8_t reg, const uint8_t data);
void max7219_send_to_chip(const uint8_t reg, const uint8_t data, const uint8_t chip);
void max7219_send(const uint8_t reg, const uint8_t data_0, const uint8_t data_1);

void max7219_test(void);

#endif /* __MAX7219_H__ */
```

Начнём с функции инициализации. Микросхема подключена к модулю `SPI2`, ножкам PB5 (CS), PB13 (SCK) и PB15 (MOSI).

```
// max7219.c
void max7219_init(const uint8_t br) {
    RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;

    // SCK, MOSI
    GPIOB->CRH |= GPIO_CRH_MODE13;
    GPIOB->CRH &= ~GPIO_CRH_CNF13_0;
    GPIOB->CRH |=  GPIO_CRH_CNF13_1;

    GPIOB->CRH |= GPIO_CRH_MODE15;
    GPIOB->CRH &= ~GPIO_CRH_CNF15_0;
    GPIOB->CRH |=  GPIO_CRH_CNF15_1;

    // CS
    GPIOB->CRL |= GPIO_CRL_MODE5;
    GPIOB->CRL &= ~GPIO_CRL_CNF5;

    CS_DIACTIVATE();

    SPI2->CR1 |= SPI_CR1_MSTR;      // master mode
    SPI2->CR1 |= SPI_CR1_BIDIMODE;  // 1 line
    SPI2->CR1 |= SPI_CR1_BIDIOE;    // MOSI
    SPI2->CR1 &= ~SPI_CR1_BR;       // spi_sck = SystemCoreClock / 16 = 8 MHz
    SPI2->CR1 |= SPI_CR1_BR_1;
    SPI2->CR1 &= ~SPI_CR1_LSBFIRST; // MSB
    SPI2->CR1 |= SPI_CR1_DFF;       // 16 bit format
    SPI2->CR1 |= SPI_CR1_SSI;       // software CS
    SPI2->CR1 &= ~SPI_CR1_CPHA;
    SPI2->CR1 &= ~SPI_CR1_CPOL;

    SPI1->I2SCFGR &= ~SPI_I2SCFGR_I2SMOD;

    SPI2->CR1 |= SPI_CR1_SPE;

    // set mode
    max7219_send_to_all(REG_SHUTDOWN,    0x01); // restart
    max7219_send_to_all(REG_DECODE_MODE, 0x00); // use normal mode
    max7219_send_to_all(REG_SCAN_LIMIT,  0x07); // use all lines

    max7219_set_brightness(br);
```

```
    max7219_clear();
}
```

Следующая функция должна релизовывать задание яркости свечения.

```
void max7219_set_brightness(const uint8_t br) {
    max7219_send_to_all(REG_INTENSITY, br > 15 ? 15 : br);
}
```

И функция очистки дисплея.

```
void max7219_clear(void) {
    for (uint32_t i = 0; i < 8; i++) {
        max7219_send_to_all(REG_DIGIT_0 + i, 0x00);
    }
}
```

Далее для корректной работы нам требуется функция `max7219_send_to_all()`, реализуем и её.

```
#define CS_ACTIVATE()           (GPIOB->ODR &= ~GPIO_ODR_ODR5)
#define CS_DIACTIVATE()         (GPIOB->ODR |= GPIO_ODR_ODR5)

__attribute__((always_inline)) static inline void send_data(uint8_t reg, uint8_t data) {
    SPI2->DR = (uint16_t)( (reg << 8) | data );
    while ( (SPI2->SR & SPI_SR_BSY) == SPI_SR_BSY);
}

void max7219_send_to_all(const uint8_t reg, const uint8_t data) {
    CS_ACTIVATE();
    for (uint32_t i = 0; i < MAX7219_CHIPS; i++) {
        send_data(reg, data);
    }
    CS_DIACTIVATE();
}
```

Так как мы пишем драйвер, то нужно позаботиться об удобном его использовании. По этой причине нужно предоставить возможность отправки данных только на определённый чип.

```
#define EMPTY_DATA              0x00

void max7219_send_to_chip(const uint8_t reg, const uint8_t data, const uint8_t chip) {
    if (chip > MAX7219_CHIPS) {
        return;
    }

    CS_ACTIVATE();
    for (uint32_t i = 0; i < MAX7219_CHIPS; i++) {
        if ( (MAX7219_CHIPS - i - 1) == chip)
            send_data(reg, data);
        else
            send_data(REG_NO_OP, data);
    }
    CS_DIACTIVATE();
}
```

Стараясь сделать драйвер более или менее универсальным, напишем функцию, благодаря которой можно отправлять данные сразу на всю линию. Выхода два: использовать возможность модуля из стандартной библиотеки `<stdarg.h>` [1], но можно поступить по другому и предавать массив.

```
void max7219_send(const uint8_t reg, const uint8_t data_0, const uint8_t data_1) {
    CS_ACTIVATE();
    send_data(reg, data_0);
    send_data(reg, data_1);
    CS_DIACTIVATE();
}
```

И осталась последняя функция для того, что бы протестировать работу микросхемы. Поступим следующим образом:

```
void max7219_test(void) {
    max7219_send_to_all(REG_DIGIT_0, 0x01);
    max7219_send_to_all(REG_DIGIT_1, 0x02);
    max7219_send_to_all(REG_DIGIT_2, 0x04);
    max7219_send_to_all(REG_DIGIT_3, 0x08);
    max7219_send_to_all(REG_DIGIT_4, 0x0F);
    max7219_send_to_all(REG_DIGIT_5, 0x10);
    max7219_send_to_all(REG_DIGIT_6, 0x12);
    max7219_send_to_all(REG_DIGIT_7, 0x14);
}
```

Вызовите эту функцию до входа в цикл `while(1)` .

Код урока (и для других библиотек) можно найти на GitHub: **CMSIS** .

Назад | **Оглавление** | **Дальше**

---

1. https://learnc.info/c/vararg_functions.html ↩