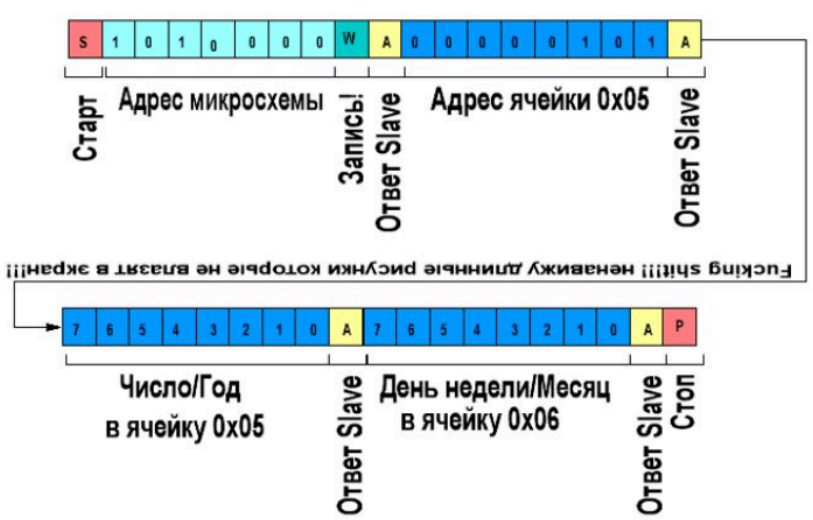
TM1637

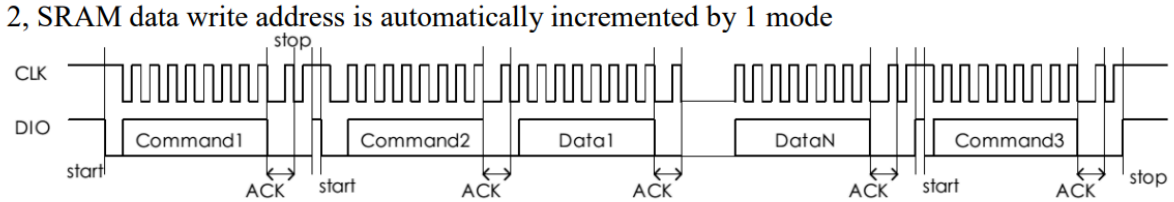
Что я поняла необычного: не работает DP для всех разрядов, кроме 2-го (во втором это двоеточие).

Что также нужно знать про этот дисплей: он не работает по I2C. Его протокол видоизменённый. В нём отсутствует адрес устройства, висящего на линии и к которому мы сейчас хотим обратиться.



На рисунке выше показан стандартный I2C. У нас же нету адреса микросхемы. Поэтому периферию I2C в контроллере мы использовать не можем.

Используем этот случай:



Command1: Set Data
Command2: Set Address
Data1 ~ N: display data transmission
Command3: Control Display

Команда -> адрес -> запись данных в буфер -> включение экрана
Опишем все шаги под этой временной диаграммой.

Command 1: Set Data.

1, the data command set

This command is used to set the data write and read, B1 and B0 are not allowed to set 01 or 11 bits.

B7	B6	B5	B4	B3	B2	B1	B0	Function	Explanation
0	1	Do not care to fill 0				0	0	Data read-write mode settings	Write data to the display register
0	1					1	0		Read key scan data
0	1				0			Address mode is set to increase	Automatic address incrementing
0	1				1				Fixed address
0	1			0				Test mode setting (internal use)	Normal mode
0	1			1					Test Mode

В этой таблице надо выбрать первую или вторую строку, потом третью или четвёртую, потом пятую или шестую и составить так команду.
Хотим писать данные в регистр дисплея. Автоматически увеличивать адрес (тогда только 1 раз надо будет написать адрес и потом просто посылать данные, как на диаграмме выше). И в нормальном моде. Т.е. получаем:

0b0100_0000 = 0x40. Короче B5 и B4 взяли равными 0. «Do not care to fill 0» = «Не важно, но ставьте 0» – это рекомендация, даже если чип формально допускает другие значения.

Command 2: Set address.

On power-up, the address defaults to 00H.

Но всё равно лучше устанавливать адрес. Потому что если мы захотим обновить значение, вызовем функцию, а адрес продолжает быть в Automatic address incrementing, и мы не сможем обновить число на дисплее.

2, set address command set

B7	B6	B5	B4	B3	B2	B1	B0	Show address
1	1	Do not care to fill 0		0	0	0	0	00H
1	1			0	0	0	1	01H
1	1			0	0	1	0	02H
1	1			0	0	1	1	03H
1	1			0	1	0	0	04H
1	1			0	1	0	1	05H

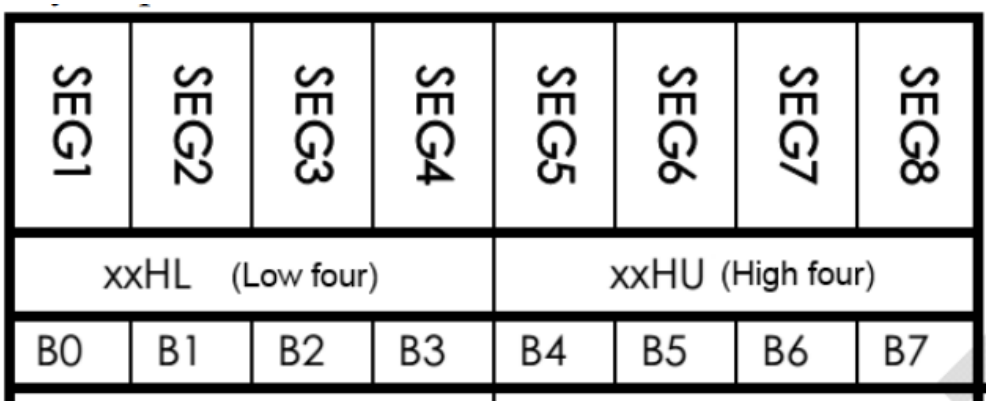
This command is used to set the display address register; If the address is set 0C6H or higher, the data is ignored until a valid address is set;

Итак, нам нужен нулевой адрес. Он соответствует первому разряду. Вообще нам нужно записать данные в 00H, 01H, 02H и 03H. Этот чип может использоваться для 6-разрядного дисплея, но нам последние два байта не нужны.

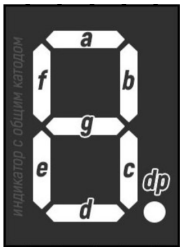
Короче пишем 0b1100_0000 = 0xC0, это грубо говоря установит курсор в начало строки.

Data1-N.

У нас дисплей 4 разрядный, то есть есть смысл писать данные 4 байта.



Порядок битов на самом деле такой: B7 B6 … B0.
Если упрощать, то биты надо писать в такой последовательности:
DP G F E D C B A



Причём помните про точку (DP), она не работает, только у второго разряда работает как двоеточие.
Соответственно, 11:00 будет:
0b00000110, // 1
0b10000110, // 1 с горящим двоеточием
0b00111111, // 0
0b00111111 // 0

Конечно, можете переставить все биты в обратном порядке и байты тоже, тогда можете перевернуть дисплей кверху ногами и смотреть на цифры иначе.

Command 3: Control display.

3, the display control

B7	B6	B5	B4	B3	B2	B1	B0	Function	Explanation
1	0	Do not care to fill 0			0	0	0	Extinction number of settings	Set the pulse width of 1/16
1	0				0	0	1		Set the pulse width of 2/16
1	0				0	1	0		Set the pulse width of 4/16
1	0				0	1	1		Set the pulse width of 10/16
1	0				1	0	0		Set the pulse width of 11/16
1	0				1	0	1		Set the pulse width of 12/16
1	0				1	1	0		Set the pulse width of 13/16
1	0				1	1	1		Set the pulse width of 14/16
1	0			0				Display switch settings	Showing Off
1	0			1					Show On

Когда все 4 байта записаны в буффер (на каждый разряд 1 байт), можно управлять яркостью и включить экран в целом.
Тут мы видим 8 вариантов яркости экрана, надо выбрать один. Где 000 – самый тусклый, 111 – самый яркий.

Команда получилась: 0x1000_1000 =0x88 (тусклый), 0x1000_1111 = 0x8F – яркий.

Отдельный момент – чтение ACK байта для проверки, не отвалился ли дисплей. Как и в I2C, ACK = 0 если всё хорошо (ведомый прижимает линию). Можно обрабатывать ситуацию, если ACK = 1.

Также отдельный момент – написание функций создания START condition и STOP condition. Но можно просто по временной диаграмме посмотреть.

Также стоит разобраться с настройкой OTYPER, OSPEEDR и PUPDR. Хотя сразу понятно, что OSPEEDR можно выкинуть без потерь.

Ну и задержку можно на таймере сделать. Да и вообще может можно ШИМ сделать на таймере, а не просто GPIO.

Код:

```
#include "stm32f4xx.h"

// Пины TM1637
#define CLK_PIN   8
#define DATA_PIN  9
#define CLK_PORT  GPIOB
#define DATA_PORT GPIOB

// Макросы для управления пинами
#define CLK_HIGH()  (CLK_PORT->BSRR = (1U << CLK_PIN))
#define CLK_LOW()   (CLK_PORT->BSRR = (1U << (CLK_PIN + 16)))
#define DATA_HIGH() (DATA_PORT->BSRR = (1U << DATA_PIN))
#define DATA_LOW()  (DATA_PORT->BSRR = (1U << (DATA_PIN + 16)))

// режим линии DIO: чтение (для ACK) и запись (для всего остального)
#define DATA_INPUT()  (DATA_PORT->MODER &= ~(3U << (DATA_PIN * 2)))
#define DATA_OUTPUT() (DATA_PORT->MODER |=  (1U << (DATA_PIN * 2)))

// задержка
void delay_us(volatile uint32_t us) {
    // При 16 МГц
    us *= 16;
    while (us--);
}
```

```
// условие start
void tm1637_start(void) {
    DATA_OUTPUT();
    DATA_HIGH();
    CLK_HIGH();
    delay_us(2);
    DATA_LOW();
    delay_us(2);
    CLK_LOW();
}

// условие stop
void tm1637_stop(void) {
    DATA_OUTPUT();
    CLK_LOW();
    DATA_LOW();
    delay_us(2);
    CLK_HIGH();
    delay_us(2);
    DATA_HIGH();
    delay_us(2);
}

// Отправка одного байта
void tm1637_write_byte(uint8_t data) {
    DATA_OUTPUT();
    // отправляет по одному биту
    for (uint8_t i = 0; i < 8; i++) {
        CLK_LOW();
        // 1 бит отображаем
        if (data & 0x01) {
            DATA_HIGH();
        } else {
            DATA_LOW();
        }
        delay_us(2);
        CLK_HIGH();
        delay_us(2);
        // переход к следующему биту
        data >>= 1;
    }
    // Ожидаем ACK (TM1637 тянет DIO вниз)
    CLK_LOW();
    DATA_INPUT();          // Переключаем DIO на вход
    delay_us(2);
    // (опционально: можно проверить, что линия LOW)
    CLK_HIGH();
    delay_us(2);
```

```
    CLK_LOW();
    DATA_OUTPUT();         // Возвращаем в режим вывода
}

// Отображение "11:00"
// Формат: DIG0 = правая цифра, DIG3 = левая
// Обычно двоеточие управляется через старший бит (бит 7) в соответствующем разряде
// Сегменты: A,B,C,D,E,F,G,DP (биты 0–7)
// Коды цифр (без DP):
const uint8_t digit_code[] = {
    0b00111111, // 0
    0b00000110, // 1
    0b01011011, // 2
    0b01001111, // 3
    0b01100110, // 4
    0b01101101, // 5
    0b01111101, // 6
    0b00000111, // 7
    0b01111111, // 8
    0b01101111  // 9
};

void tm1637_display(uint8_t dig0, uint8_t dig1, uint8_t dig2, uint8_t dig3) {
          // создаём массив кодов разрядов
    uint8_t data[4] = {
        digit_code[dig0],
        digit_code[dig1] | 0x80, // DIG1 - с двоеточием
        digit_code[dig2],
        digit_code[dig3]
    };

    // 1. Режим автоинкремента адреса
    tm1637_start();
    tm1637_write_byte(0x40); // 01000000: автоадрес, запись данных
    tm1637_stop();

    // 2. Запись данных, начиная с адреса 0xC0 (00H)
    tm1637_start();
    tm1637_write_byte(0xC0); // команда установки адреса 00H
    for (int i = 0; i < 4; i++) {
        tm1637_write_byte(data[i]);
    }
    tm1637_stop();

    // 3. Включить дисплей с минимальной яркостью
    tm1637_start();
    tm1637_write_byte(0x88); // 10001000: включён, яркость = 1/16 (минимум)
```

```
    tm1637_stop();
}

int main(void) {
    // Включаем тактирование порта B
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;

    // Настраиваем PB8 и PB9 как выход с Open-Drain (как в I2C)
    // Но TM1637 требует push-pull? На практике — open-drain с подтяжкой
    // Однако TM1637 втягивает линию сам, поэтому можно использовать push-pull
    // Установим как выход общего назначения, push-pull, low speed
    GPIOB->MODER &= ~((3U << (8*2)) | (3U << (9*2)));
    GPIOB->MODER |=  ((1U << (8*2)) | (1U << (9*2))); // Output mode

    GPIOB->OTYPER &= ~((1U << 8) | (1U << 9)); // Push-pull
    GPIOB->OSPEEDR &= ~((3U << (8*2)) | (3U << (9*2))); // Low speed
    GPIOB->PUPDR &= ~((3U << (8*2)) | (3U << (9*2)));   // No pull

    // Инициализация TM1637
    tm1637_display(0, 5, 2, 3);

    while (1) {
    }
}
```