Подключение и настройка

На пин **DIN** первого светодиода (начало ленты) подаётся сигнал, формируемый STM32. Из-за разницы питающих напряжений, сигнал следует **поднять** до уровня 5 вольт с помощью специальной **микросхемы-транслятора** логики или с помощью настройки GPIO-пина в режиме **Open Drain**, подтянув его резистором.

ВАЖНО!

При использовании Open Drain нужно убедиться, что пин выдержит 5 вольт. Узнать это можно в даташите на свой МК.

Пример:

Table 5. Medium-density STM32F103xx pin definitions (continued)

Pins							Pin name	Type[1]	I / O Level[2]	Main function[3] (after reset)	Alternate functions[4]	
LFBGA100	UFBG100	LQFP48/UFQFPN48	TFBGA64	LQFP64	LQFP100	VFQFPN36					Default	Remap
G2	L2	10	G2	14	23	7	PA0-WKUP	I/O	-	PA0	WKUP/ USART2_CTS[9]/ ADC12_IN0/ TIM2_CH1_ ETR[9]	-
H2	M2	11	H2	15	24	8	PA1	I/O	-	PA1	USART2_RTS[9]/ ADC12_IN1/ TIM2_CH2[9]	-
J2	K3	12	F3	16	25	9	PA2	I/O	-	PA2	USART2_TX[9]/ ADC12_IN2/ TIM2_CH3[9]	-
K2	L3	13	G3	17	26	10	PA3	I/O	-	PA3	USART2_RX[9]/ ADC12_IN3/ TIM2_CH4[9]	-

Пин таймера без толерантности

Table 5. Medium-density STM32F103xx pin definitions (continued)

Pins							Pin name	Type[1]	I / O Level[2]	Main function[3] (after reset)	Alternate functions[4]	
LFBGA100	UFBG100	LQFP48/UFQFPN48	TFBGA64	LQFP64	LQFP100	VFQFPN36					Default	Remap
B4	A3	45	B3	61	95	-	PB8	I/O	FT	PB8	TIM4_CH3[9]	I2C1_SCL / CANRX
A4	B3	46	A3	62	96	-	PB9	I/O	FT	PB9	TIM4_CH4[9]	I2C1_SDA/ CANTX
D4	C3	-	-	-	97	-	PE0	I/O	FT	PE0	TIM4_ETR	-
C4	A2	-	-	-	98	-	PE1	I/O	FT	PE1	-	-
E5	D3	47	D4	63	99	36	V_{SS_3}	S	-	V_{SS_3}	-	-
F5	C4	48	E4	64	100	1	V_{DD_3}	S	-	V_{DD_3}	-	-

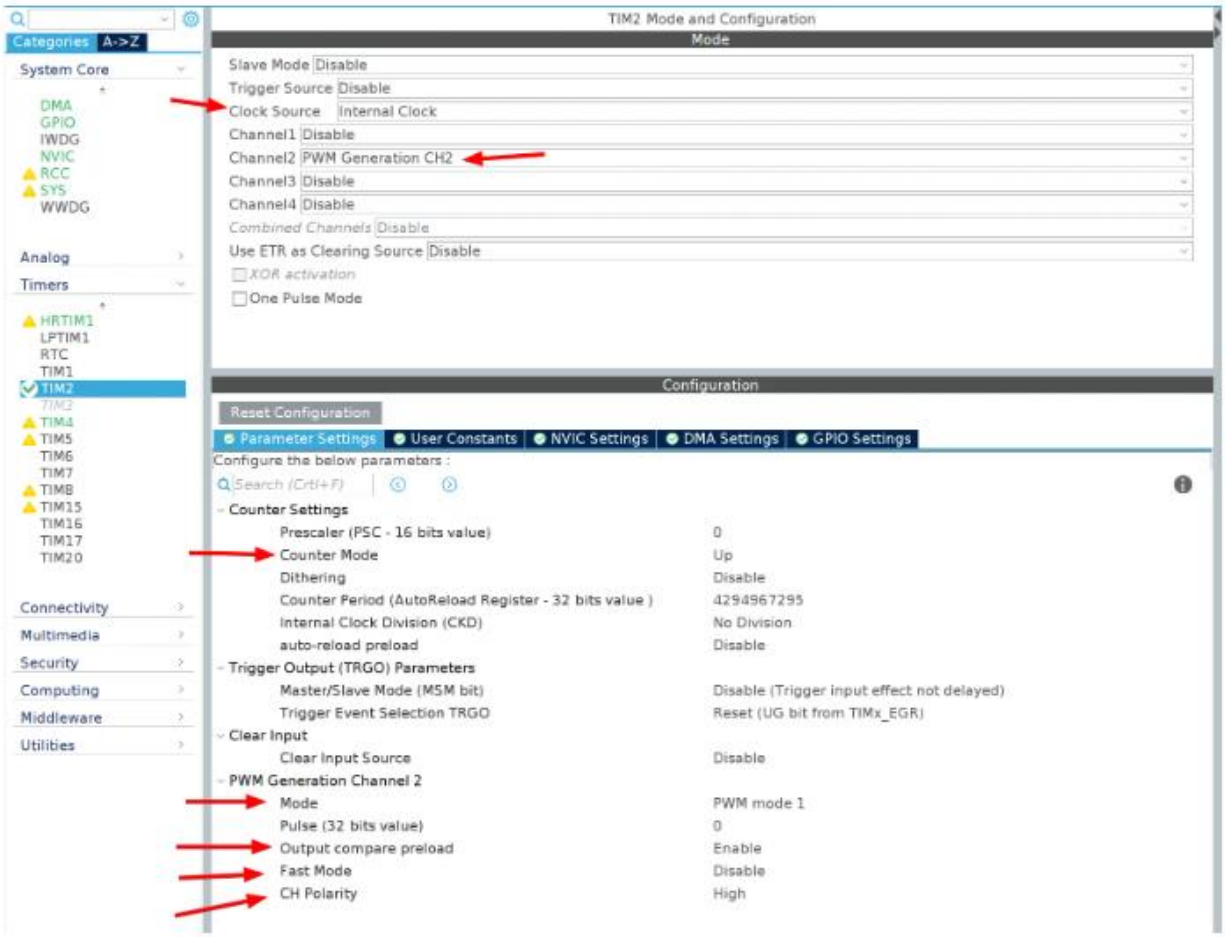
1. I = input, O = output, S = supply.
2. FT = 5 V tolerant.

Пины, толерантные к 5 Вольтам

Настройка таймера в CubeMX

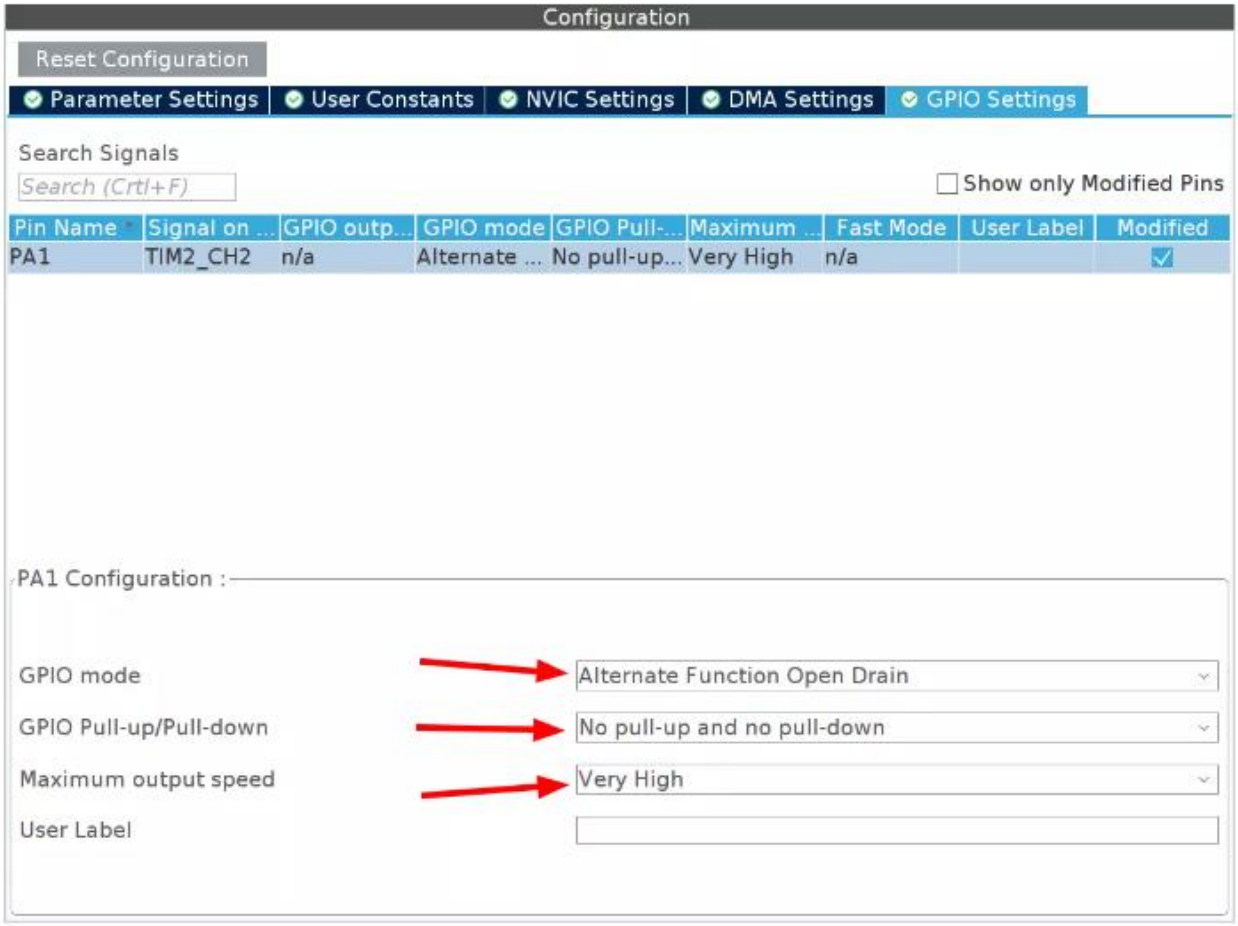
ОГРАНИЧЕНИЯ: Из-за особенностей таймеров, минимально стабильная частота работы микроконтроллера — **32 МГц**.

Сперва нужно настроить таймер в режиме **ШИМ**. Обратите внимание на отмеченные стрелками настройки.

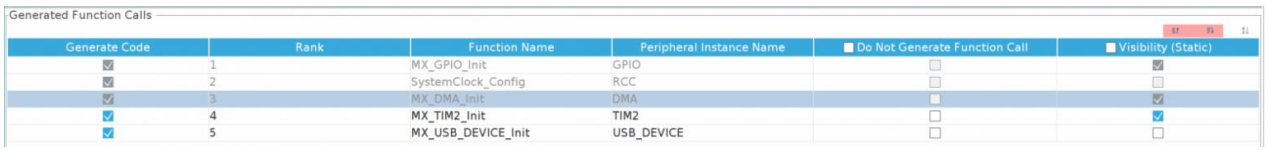


Отправка значений в таймер происходит с использованием **DMA**, поэтому настроим и этот блок.

Ножка должна иметь **наивысшую скорость** из доступных. Если выбран режим **Open Drain**, то не забудьте переключиться.



Также проверьте, что генерация **DMA_Init** стоит выше, чем **TIM_Init.** Иначе таймер не узнает про DMA, сигнал генерироваться не будет.



Настройка библиотеки

Сгенерируем код, добавим файлы библиотеки в проект. Откроем **.h**-файл и посмотрим, что можно настроить.

```
#define WS2811    ///< Семейство: {WS2811S, WS2811F, WS2812, SK6812}
// WS2811S – RGB, 400kHz;
// WS2811F – RGB, 800kHz;
// WS2812  – GRB, 800kHz;
// SK6812  – RGBW, 800kHz

#define NUM_PIXELS 4 ///< Кол-во диодов в цепочке

// Гамма-коррекция, должна чинить красный и зелёный, пробуйте и смотрите
#define USE_GAMMA_CORRECTION 1

#define TIM_NUM     2  ///< Номер таймера
#define TIM_CH      TIM_CHANNEL_2  ///< ШИМ-канал таймера
#define DMA_HANDLE hdma_tim2_ch2  ///< Канал DMA
// Канал DMA можно найти в main.c / tim.c
```

Function Reference

Теперь, для проверки, можно попробовать **забилдить** проект и посмотреть на доступные функции. Все методы возвращают enum-статусы.

```
typedef enum ARGB_STATE {
    ARGB_BUSY = 0,  ///< DMA-отправка в процессе
    ARGB_READY = 1, ///< DMA Готов к отправке
    ARGB_OK = 2,    ///< Успешное выполнение функции
    ARGB_PARAM_ERR = 3, ///< Ошибка входных параметров
} ARGB_STATE;

ARGB_STATE ARGB_Init(void);   // Инициализация
ARGB_STATE ARGB_Clear(void);  // Очистка ленты

ARGB_STATE ARGB_SetBrightness(u8_t br);  // Установить глобальную яркость

ARGB_STATE ARGB_SetRGB(u16_t i, u8_t r, u8_t g, u8_t b);  // Зажечь диод в RGB
ARGB_STATE ARGB_SetHSV(u16_t i, u8_t hue, u8_t sat, u8_t val);  // Зажечь диод в HSV
ARGB_STATE ARGB_SetWhite(u16_t i, u8_t w);   // Зажечь белый компонент (для RGBW)

ARGB_STATE ARGB_FillRGB(u8_t r, u8_t g, u8_t b);    // Залить всё в RGB
ARGB_STATE ARGB_FillHSV(u8_t hue, u8_t sat, u8_t val); // Залить всё в HSV
ARGB_STATE ARGB_FillWhite(u8_t w);   // Заливка белого компонента (для RGBW)
```

```
ARGB_STATE ARGB_Ready(void); // Получить статус DMA
ARGB_STATE ARGB_Show(void);  // Обновить диоды
```

Пример использования

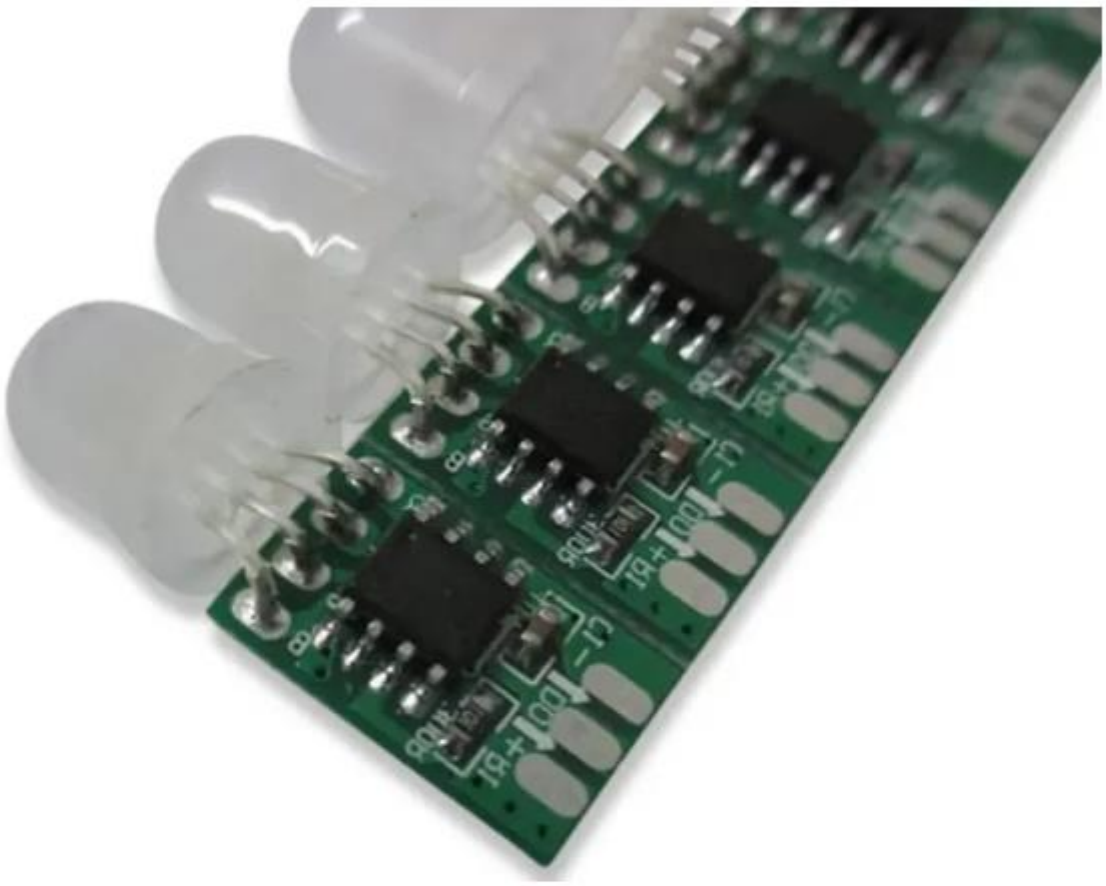
```
void main(void) {
    ARGB_Init();

    ARGB_Clear();
    while (ARGB_Show() == ARGB_BUSY) ; // Вариант 1

    ARGB_SetRGB(0, 255, 0, 128);
    ARGB_SetHSV(1, 230, 250, 255);
    while (!ARGB_Show()) ; // Вариант 2

    ARGB_SetRGB(3, 200, 0, 200);
    // Вариант 3:
    while (ARGB_GetState() != ARGB_READY) ;
    ARGB_Show();
}
```

Описание

Адресные светодиоды ленты используют для различной индикации, как бытовой, так и коммерческой. Ключевое **отличие** от обычных RGB-диодов в том, что их можно зажигать **отдельно**, каждый своим цветом.

Это поведение обусловлено тем, что у каждого диода стоит чип-драйвер. Снаружи, как в случае с WS2811, или внутри, как у WS2812 и остальных.

Чип принимает сигнал, запоминает первые импульсы, а остальные передаёт далее по цепочке.



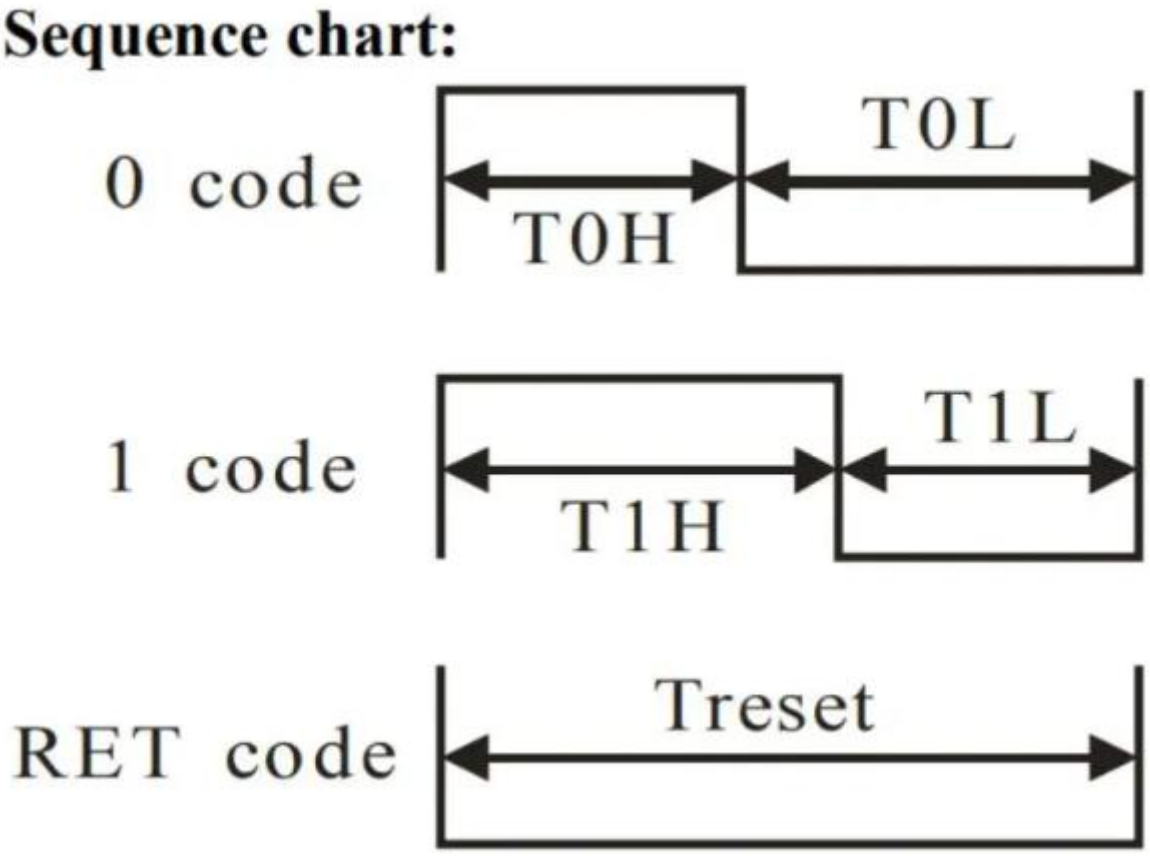
WS2811

WS2812

Протокол данных

Свечение каждого **суб**пикселя кодируется **8 битами**. Т.е. для **RGB** *(WS281X)* **24 бита**, для **RGBW** *(SK6812)* **32 бита**.

Код бита задаётся длиной импульса, то есть скважностью.

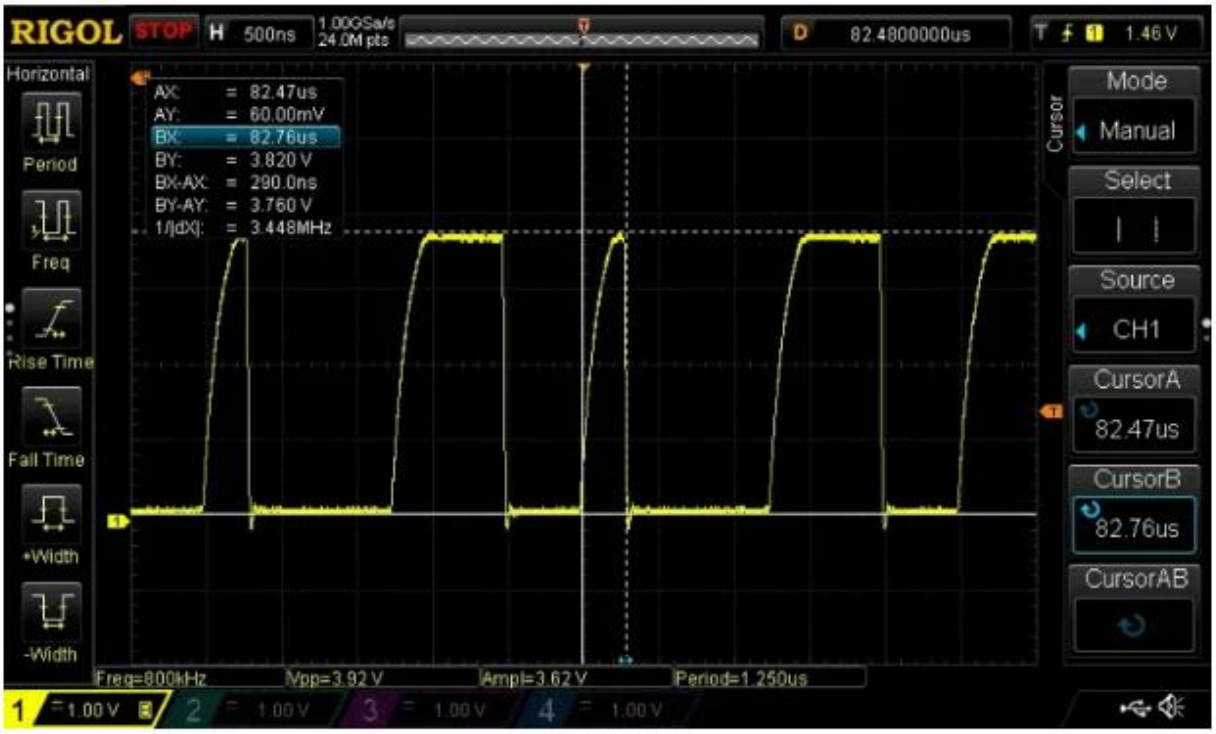


Кодирование сигнала

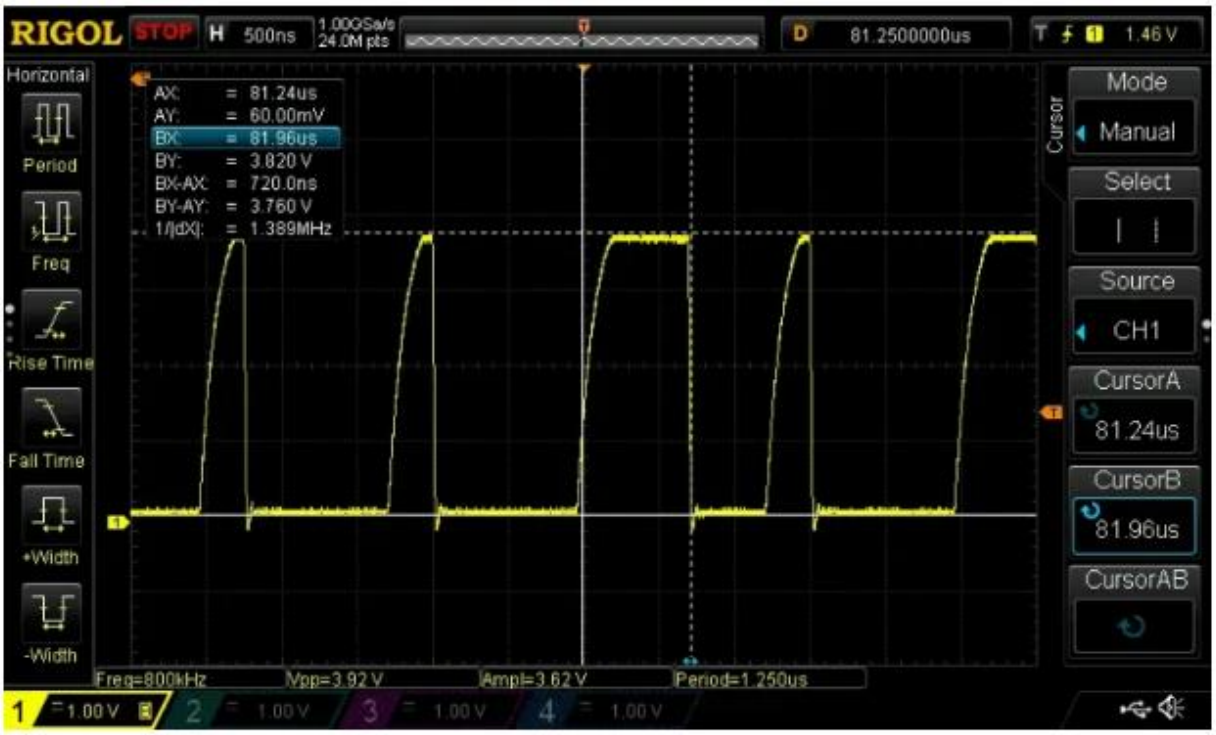
Существует и код **RET** — пауза, означающая конец передачи.

У всех контроллеров разные тайминги:

	WS2811 (slow)	**WS2811** (fast, SET=1)	**WS2812**(b)	**SK6812**
Частота	400 КГц	800 КГц	800 КГц	800 КГц
Период (T)	2,5 мкс	1,25 мкс	1,25 мкс	1,25 мкс
T0H	0,5 мкс *(20%)*	0,25 мкс *(20%)*	0,35 мкс *(28%)*	0,3 мкс *(24%)*
T1H	1,2 мкс *(48%)*	0,6 мкс *(48%)*	0,7 мкс *(56%)*	0,6 мкс *(48%)*
T0L	2,0 мкс	1,0 мкс	0,8 мкс	0,9 мкс
T1L	1,3 мкс	0,65 мкс	0,6 мкс	0,6 мкс
Допуск	+/- 150 нс	+/- 150 нс	+/- 150 нс	+/- 150 нс
RET	> 50 мкс *(20T)*	> 50 мкс *(40T)*	> 50 мкс *(40T)*	> 80 мкс *(64T)*



Пример кода "0"



Пример кода "1"

Реализация на STM32

Большинство решений основаны на использовании пустых тактов. Это означает, что *весь* процессор тормозит на время отправки сигнала. Такой способ не только тратит уйму процессорного времени, но и рискует сломаться, в случае возникновения прерывания.

Посчитаем длину передачи сигнала на 1 диод: **1,25 мкс * 24 бит = 30 мкс**.
Для **n** диодов: **T = 30 * n + 50** мкс.

30 диодов — уже **1 миллисекунда**.

Иными словами, протокол на задержках стоит использовать только для *малого* количества диодов, чтобы не мешать основной программе.

Именно из-за этой проблемы я в своё время впервые обратился к STM32.

В других вариантах используется шина **SPI**, которую настраивают на частоту 800 КГц. Я не проверял, но многие пишут про ощутимую потерю точности сигнала.

Что же делать?

В почти всех микроконтроллерах STM32 существует блок **DMA** (Direct Memory Access). Он позволяет передавать данные между **периферией** и **памятью** в разных направлениях **без участия процессора**.

В качестве исполнительной периферии используется **таймер**, настроенный в режиме **ШИМ**.

Буферный массив

Любой способ передачи сигнала подразумевает буфер, в котором хранятся значения **скважности** сигнала.

В сети встречаются множество вариантов буфера сразу под **все** диоды. Чаще всего скважность 8-битная, поэтому такой будет весить **N диодов * 24 байт**. Уже под **100 диодов** он займёт **более 2 КБ ОЗУ**.

А если записывать скважность с шириной **32 бита**, как требуют некоторые серии МК, под 100 диодов буфер будет более **9 КБ**.

Реализация моего метода была придумана не мной. В ней очень **хитро** используется память.

Буфер здесь **двойной**. Первый имеет размер **N диодов * 3 байта**. В нём хранится цвет в представлении **RGB**.

Второй буфер — для скважностей. Он фиксированный, занимает всего **48 байт**, или **64 байта** для RGBW. В него вмещаются всего **2 диода**.

Прерывания DMA позволяют заполнять одну часть буфера, пока отправляется вторая. Используя такой подход, можно растягивать цепочку диодов почти до **бесконечности**, покуда хватит памяти для первичного буфера или частоты обновления.

Преобразование логики

Дело в том, что адресные диоды воспринимают сигнал, опираясь на напряжение своего питания.

Открыв даташит на **WS2812b**, мы увидим такие строки:

	Min	**Max**
$\mathbf{V_{IH}}$	$0{,}7\ V_{DD}$	—
$\mathbf{V_{IL}}$	—	$0{,}3\ V_{DD}$

Это — границы восприятия сигнала. Иными словами, при питании от **5.1**—**5.2** Вольт, минимальный уровень сигнала — **3.57** Вольт.

Так как STM32 выдаёт сигнал величиной **3.0**—**3.3** Вольта, его нужно **увеличить**.

Вариантов это сделать несколько:

1. **Уменьшить напряжение питания ленты**
   - **Отрегулировать** напряжение на БП
   - Для небольшого отрезка запитать **всю ленту через диод**
   - Отрезать **первый** светодиод, и запитать только его **через диод**
2. **Поднять потенциал GND микроконтроллера** *(подробнее)*
3. **Воспользоваться преобразованием логики**

Так как предполагается использование в коммерческих проектах, где необходима не только надёжность, но и возможность беспроблемной замены отдельных компонентов пользователем, то самый безопасный вариант — последний.

Способы преобразования логики были рассмотрены в данной статье. В ней сделаны выводы о том, что самый подходящий преобразователь — **SN74LVC**.

Однако, при его отсутствии или для удешевления BOM, можно воспользоваться режимом **Open Drain**.

Обход буфера

DMA настраивается в **кольцевом** режиме передачи. Новые транзакции будут возникать до тех пор, пока не будут остановлены **вручную** в коде.

DMA генерирует **прерывания** каждую **половину** транзакции. Поэтому наш буфер размером в **2 диода**. Пока идёт передача сигнала
для **первого** диода, просчитывается и загружается сигнал для **второго**.

1-я половина	**2-я половина**	**Счётчик**
LED [0]	LED [1]	0
LED [2]	**LED [1]**	1

LED [2]	LED [3]	2
LED [4]	**LED [3]**	3
LED [4]	RET {1}	4
RET {2}	**RET** {1}	5
RET {2}	**DMA_STOP**	6

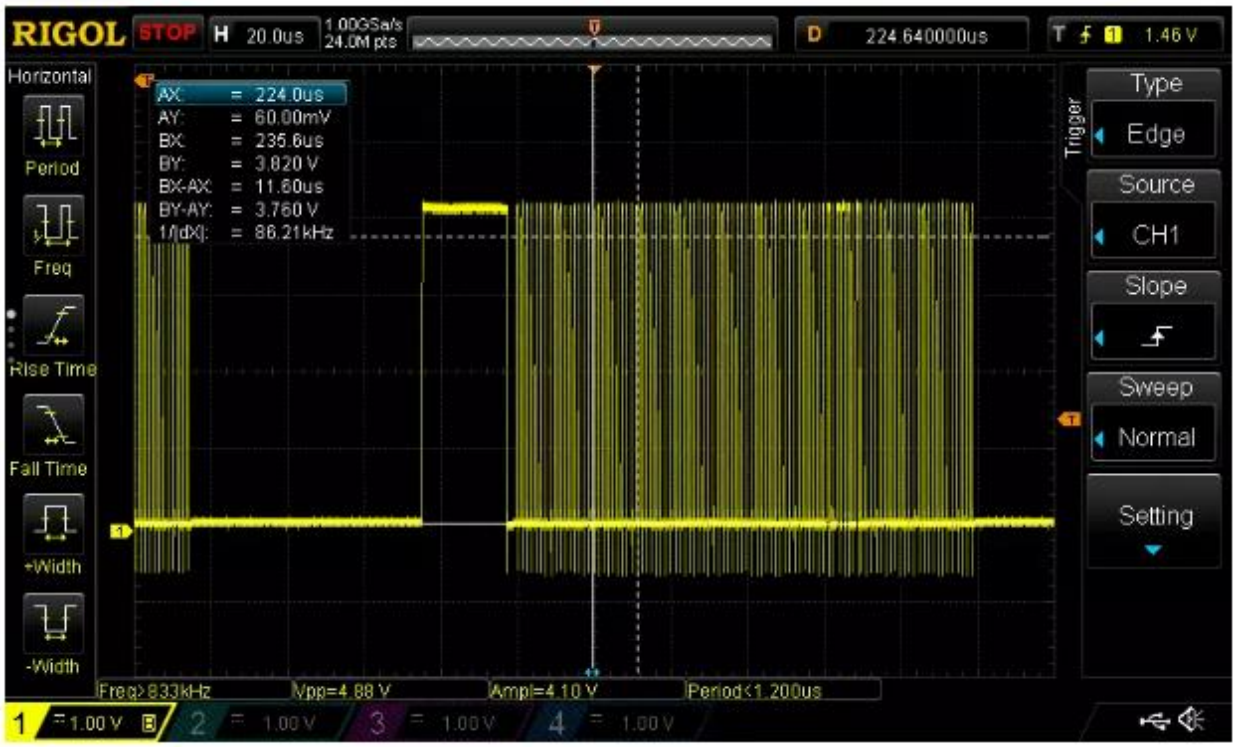
*Состояние буфера. **Bold** — текущая передача*

Проблемы при разработке

В первую очередь я столкнулся с **согласованием логики**. От USB компьютера всё работало, а от любого блока питания — нет. Решение пришло после пары тыков вольтметром и чтения даташита. Оказалось, что порты компьютера под просадкой выдавали порядка **4.6** Вольт, что есть **3,2** Вольта логической единицы. А все блоки питания стандартно выдавали в районе **5.2** Вольт, поэтому лента даже не зажигалась.

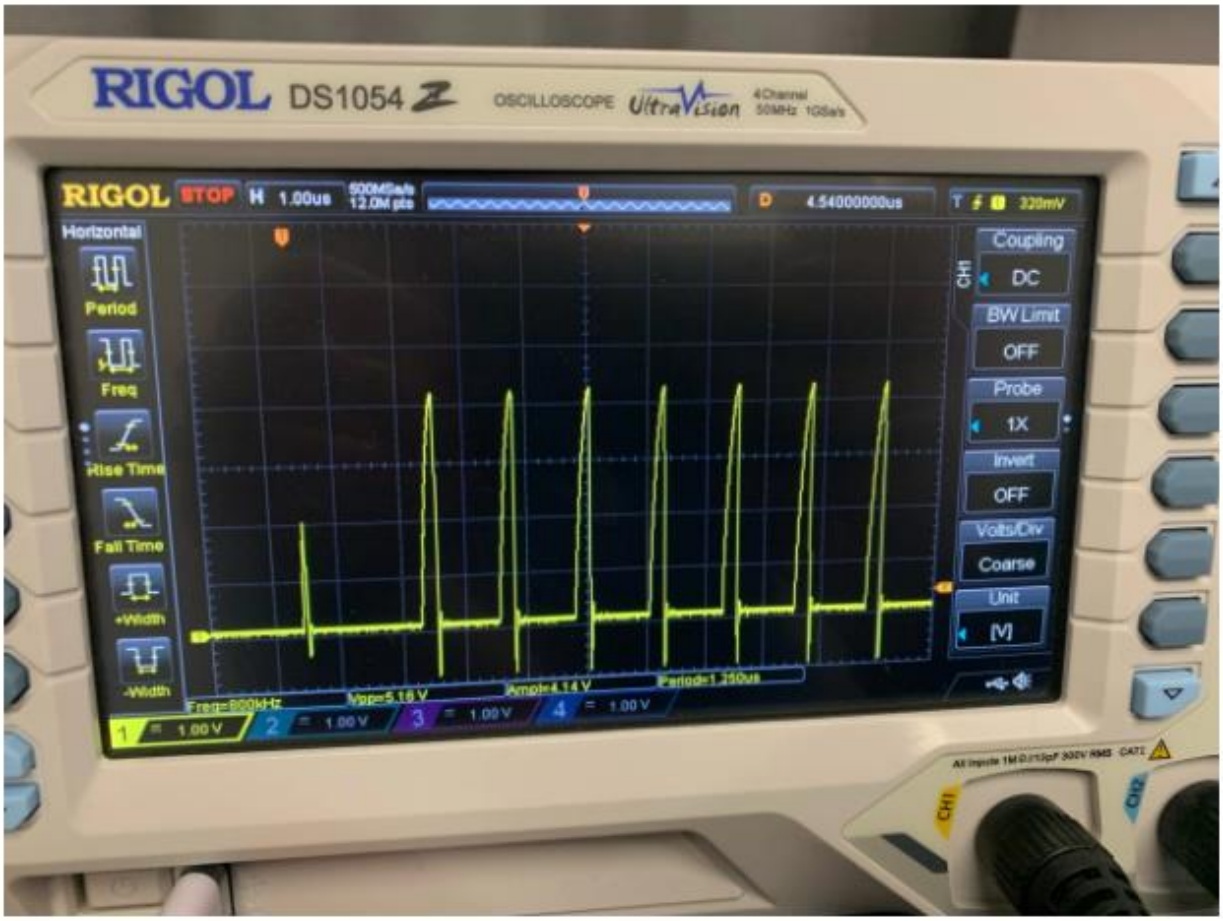
Вторую проблему принесла библиотека HAL. Дело в том, что у базовых таймеров нет **IDLE-состояния** ног. Поэтому, после **остановки** таймера, ножка входила в Z-состояние, а подтяжка выкидывала сигнал вверх.

На осцилограмме видно последовательность: **сигнал**, **RET** (пауза), **Z-state**, **запуск** таймера, **сигнал**, **RET**.



Это приводило к тому, что первый диод в ленте считывал этот импульс и **зажигался**.

Даже с помощью **остановки-запуска** таймера невозможно было достичь нормальной работы. Либо из-за массивности HAL-функций, либо из-за особенностей работы периферии, возникала небольшая временная задержка, которой было достаточно для зажигания этого диода.



Решение было не таким очевидным, но нашлось довольно быстро. В функции `HAL_TIM_PWM_Stop_DMA` была обнаружена такая строчка:

```
/* Disable the Capture compare channel */
TIM_CCxChannelCmd(htim->Instance, Channel, TIM_CCx_DISABLE);
```

Это и есть **отключение GPIO-канала** таймера. После её удаления, удалось достичь стабильной работы. Поэтому пришлось скопировать весь код этого метода к себе и немного отредактировать.

Третья проблема — фундаментальная. Заключается в особенностях работы таймеров. Если задать частоту **ниже 32 МГц**, то ощутимо теряется **точность** сигнала. Например, для **8 МГц**: Для получения частоты 800 КГц задаётся **ARR = 9**. Значит регистру **CCRx** доступны только значения **0..8**. А это примерно **100 КГц** точности или разброс в **10 мкс**, что уже очень критично.

Вариант борьбы с этим — уничтожение этой концепции генерации сигнала, переход на ассемблерные задержки или прерывания.

В любом случае, зачастую в проектах используют максимальную тактовую частоту, а снижают её только для энергосбережения, когда светодиоды уже должны быть выключены.

Другой вариант — использование отдельного МК, например F0 или G0, как **UART/SPI/I2C -> ARGB** драйвер. Такие проекты уже существуют.

Мой выбор — принять все ограничения, а для open-source сделать пометку.

Оценка скорости

Максимальная **частота** обновления адресной ленты упирается напрямую в протокол. Посчитаем предел для **25 FPS**.

25 Гц -> 40 мс = **40.000 мкс**. Передача для **1 диода** занимает **30 мкс**.
Таким образом, предельное значение — порядка **1300 шт**.

Ссылки

1. https://crazygeeks.ru/stm32-argb-lib/
2. https://github.com/Crazy-Geeks/STM32-ARGB-DMA
3. https://www.thevfdcollective.com/blog/stm32-and-sk6812-rgbw-led
4. https://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-2/
5. https://cdn-shop.adafruit.com/datasheets/WS2812.pdf
6. https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf
7. http://www.normandled.com/upload/201808/WS2815%20LED%20Datasheet.pdf
8. https://cdn-shop.adafruit.com/product-files/2757/p2757_SK6812RGBW_REV01.pdf