**STM Lesson 119. WS2812B. Ribbon on smart RGB LEDs. Part 1**

Posted on [May 14, 2018](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-1/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-1/#respond)

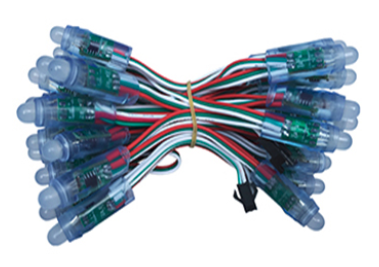
[Programming LessonsLearn all the in-house cooking **programming** training ! Get the book on the e-mail!To learn moresheremetev.info](https://an.yandex.ru/count/HsBKeEefl2i50D01CSk_ari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kGAyPRF5v8KauF02qhkScG_u0eA0W820e1AO0yy8g0CIi0C4k0J_0UW4XWJu1FwC0OW5_em1a0MPkG6W1QOgg0Ni8x05x2Eu1Oelm0NmcGV81R3g0T05a13W1J_m1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za6000040K0002f1_XHt2SDOiL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB4C4t--huKG00eUTHELjr1G302u2Z1SWBWDIJ0TaB-57S9mrYnKZe2_wC0V0B1eWCjfNUlW7e30AO3VYSPV8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uew7cmy7OuaW0?stat-id=3&test-tag=80264378212353&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODAyNjQzNDg4NjA0MTYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Dating and chatting on web camerasRegistration and communication on the site are free. There is a guest entrance.To learn morehitlove.ru](https://an.yandex.ru/count/U8RJF1eIEb050DG1CSo_ari00000ECgs7402I09Wl0Xe1724kxc20O01nfYFzRESyguWY06-YR7-Av01j92OyYAO0UxNkFyYe06CvvxN0QW1gh7jhWAu0OYYjfC2m042s07gYUa2w04E-06s0OW21A02ykA82Ra2l6MpnUI59E3m0j-ryGRu0eA0W820e1A00vYTsT02Y0E3j_pl2fW3yUm2i0C4k0J_0UW4f0Fu19XSY0MON905Z-81e0NK6wW5k1Um1RWNk0M-ES05bu03o0NMOT05nWhW1MBm1G6O1gYhgtAW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000907Aga7-570EJ9YnKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08kR74180A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiGJnQhs_XH003aArivMtK50C0BWAC5o0k0r9C1sGluKS0vCcB5IEWBc5pm2mQ83Ao-thu1w0m2c0t9icVo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AF1ufF6sE98?stat-id=4&test-tag=80264378212353&format-type=24&banner-test-tags=eyI0NjgzMTM4MDc0IjoiODAyNjQzNDg4NjA0MTYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In this lesson we will try to work with RGB LEDs, in which a chip is built in to control their glow.

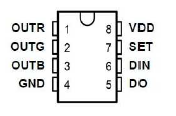
This LED is **WS2712B** from **WORLDSEMI** .

To understand what this LED represents, let's look at its predecessor, **WS2811** . It was just a chip, to which the RGB-LED was connected. This was done, for example, here in this way



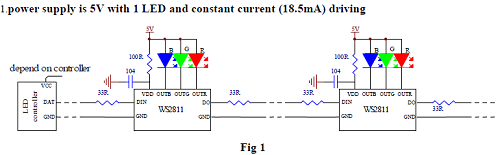
This LED garland, in each flashlight is an RGB-LED, connected to the chip WS2811, located on a small board.

Let's look at the purpose of the conclusions of the chip WS2811



In addition to the power foot and common wire, we see here the contacts for connecting the legs of the RGB-LED separately for each color, as well as the leg of the digital input DIN and the leg of the digital DO output. There is also another SET foot, which is required to activate the reduced speed mode.

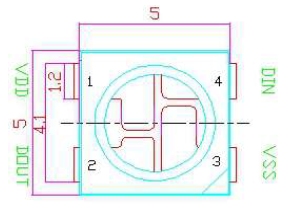
Therefore, we connect the power to the corresponding legs, the LED and transmit the digital signal to the DIN leg in a certain way so that our LED lights up in a certain color. Then, if we hold a certain time, the LED will light up with this color. But if we do not survive this time and start transmitting the next digital code, this code will move through the leg of the DO digital output to the next same chip, if any. The next chip we connect to this leg of our chip DIN foot. And thus we build a chain of such chips with LEDs connected to them (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/04/stm119img02.png)

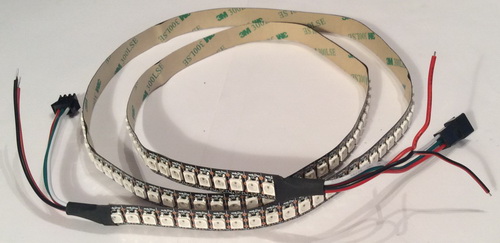
Similarly, our LED, considered in this lesson, is **arranged** - **WS2712B** . Only in it already the luminous element is immediately connected to the chip. Outside the power legs, the common bus, as well as the digital input and output



Here is the purpose of his legs along with the dimensions

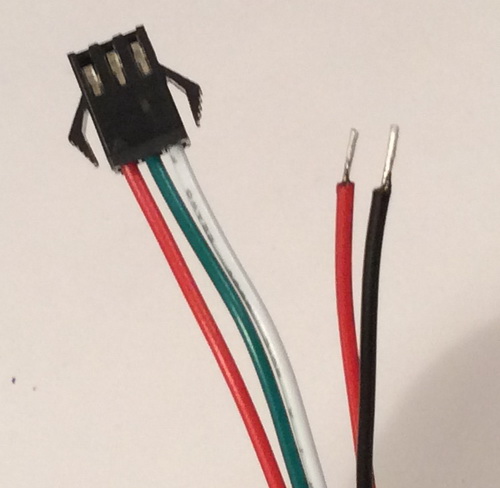


To me in hands miraculously got here such a ribbon, made on these smart LEDs (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/04/stm119img07.jpeg)

There are a lot of such tapes. They differ in color, density of LEDs (the number of LEDs on 1 meter of tape), as well as the degree of protection. This tape is black, density - 144 LEDs per meter, and protection - the smallest. Ordinary paper tape with it planted on it and connected by LEDs. I took just this for the purposes of experiments and did not plan its use in places not protected from atmospheric influences. In addition, it is necessary to pay for protection and with a higher degree of protection, tapes are more expensive. The length of my tape is only 1 meter, therefore, accordingly, and the LEDs on it are only 144. Here we will work with these 144 LEDs.

Also, we see that 5 wires are drawn from both ends of this tape



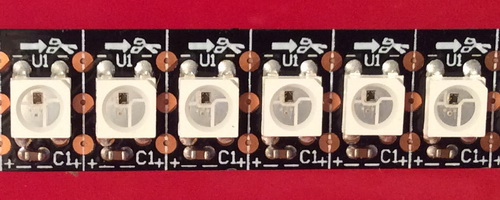
The purpose of each wire.

Red wire in the shoe - power, white in the shoe - common, green in the shoe - signal.

Red single - high-current power, black - high-current common wire.

At both ends, only the signal wires have a difference. On one side is the entrance, on the other - the exit. The illustration shows the input block, we will use this side. The output side is needed to connect the next tape.

This is how the LEDs on my tape look close



In addition to LED tapes, LEDs WS2812B also have matrices. They come in different sizes - 8x8, 32x32, etc. You can also purchase these LEDs separately. This is usually required either to assemble some of its circuits, or to replace the LED that failed. Determine this LED is not difficult, because all the working LEDs located on the tape or the matrix before it, will work properly, and this LED will not glow and also the LEDs located after it will not glow.

Now let's talk about how the signal that controls our smart LEDs is transmitted, how it is arranged, what protocol it has, as they say.

And the protocol for this control signal, it turns out to be not so complicated.

First, we transmit 24 bits of color for the very first LED. Bits are transmitted in the following sequence: first green, then red, then blue

http://narodstream.ru/wp-content/uploads/2018/04/stm119img08.png

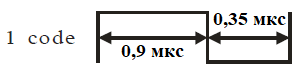
Somehow unusual (not R + G + B), but it's okay, we'll manage.

Now let's see how each bit is transmitted. For each bit, a certain time is assigned - 1.25 microseconds.

If we transmit zero, then first we set a high signal level, keep it in this state of 0.35 microseconds (a deviation of 150 nanoseconds or 0.15 microseconds is allowed). After this time, we set a low level on the foot and hold it until the next bit is transmitted 0.9 microseconds (the same deviation is allowed - 150 nanoseconds)

http://narodstream.ru/wp-content/uploads/2018/04/stm119img10.png

The unit is transferred in reverse. We also set a high level, wait for 0.9 microseconds (the deviation is the same), then lower the level, wait for 0.35 microseconds (the deviation is the same - 150 nanoseconds)



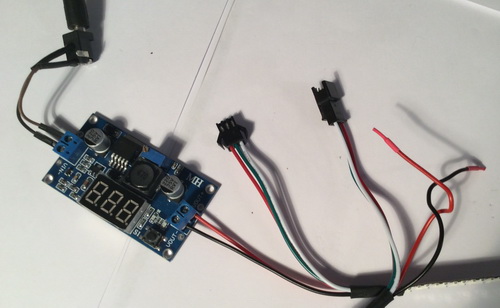
Thus, if we transmitted 24 bits in this way and did not stop there and started sending 24 more bits, the first 24 bits will move to the next LED, and the new 24 bits will be loaded into the buffer of the first one. And so it will be until after the transmission of the last 24 bits we wait for a certain time. And a certain time it's 50 microseconds or more. At that time, the first 24 bits of the first LED will not be transmitted to anyone and all the LEDs will light up with the color they took the last one. And if we start to transfer something further - then it will be a portion for the next phase of the LEDs' glow.

And, since the transmission time of one bit is constant, it is easy to calculate the data transfer rate. It will be equal to 800 kilobits per second. It is also easy to calculate that it takes 30 microseconds to transmit 24 bits, which are necessary and sufficient for the information required for one LED. Accordingly, for all 144 LEDs, 4.32 milliseconds are required. Quite good speed.

Now a little about the electrical parameters of these LEDs.

The LED is powered from 3.5 to 5.3 volts.

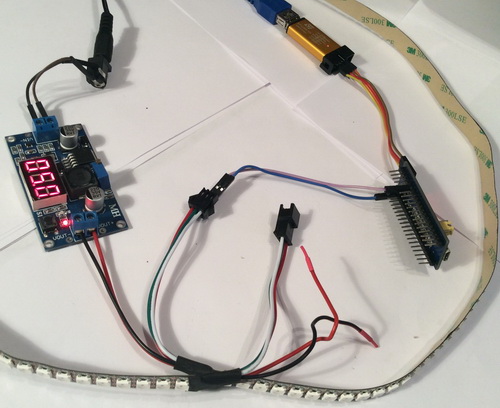
The current consumption when the LED is fully illuminated with white light (all 24 bits in units) we can also count. At the given color the consuming power - 0,3 watt is declared. Therefore, if we feed the tape with a voltage of 5 volts, it will be 60 milliamperes. And if we light all our 144 LEDs completely white for the entire brightness (units in all bits in all the LEDs), then we get the current consumption of our whole tape of 8.64 amps. So that we do not get such a huge current, we will not light our LEDs with such full light, and we will light them with certain colors, and then no more than half of their glow, so we will completely dispense with a 3-amp DC-DC converter that we connect to power wires tape



The power wires of the tape on the output side should be damped by heat shrinkage to avoid short circuits due to careless handling.

That is, we will try to ensure that when we test the LEDs burn up to a maximum of 1/6 of the maximum of their glow. For example, to make the LED light green, we set the RGB intensity to 0.128.0. And if we need to turn on the orange color, and it's both red and yellow at the same time, we'll turn on 64,0,64. That is, the total number we will not exceed 128, and this will be 1/6 of the glow, so the total current consumption of the tape we will not exceed 1.5 amperes. Therefore, you can not be afraid of anything. And even with such a glow with this density of LEDs on the tape, it will glow very brightly.

We will use the **microcontroller** folk - **STM32F103C8T6** , located on a very cheap debugger. We will also use the programmer, which is cheap, so we are almost immune to the tape controller. The tape from one leg will be controlled using PWM, included in the settings of a certain timer. Also, DMA will be used to speed up the process. At the same time, we will learn how to use DMA in PWM. This is how our board connected to the tape looks (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/04/stm119img13.jpeg)

The debug board will be powered by the ST-Link programmer, so we only use the common and signal wires to the ribbon. From what kind of leg we are driving it, we will see when we adjust the timer.

In the [**next part of**](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-2/) this lesson we will create a project, study the operation of the timer in PWM mode using DMA and try to light certain LEDs in different colors.

**STM Lesson 119. WS2812B. Ribbon on smart RGB LEDs. Part 2**

Posted on [May 16, 2018](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-2/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-2/#respond)

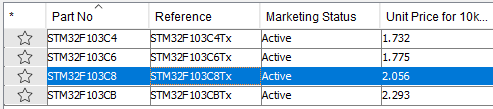
[Programming LessonsLearn all the in-house cooking **programming** training ! Get the book on the e-mail!To learn moresheremetev.info](https://an.yandex.ru/count/I3_7vy0zPka50D01CUU_ari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG8ZqKI1OIcI0_02X9NdeWBu0eA0W820i0kO0wKVg0CIi0C4k0J_0UW4jWFu19wm0OW5dh01a0M-sm6W1Uyqg0NdBB05voou1UOtm0NmcGV81R3g0T05b1JW1J_m1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHJ8QnQCL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB43ZEzixuKG001l_wEbjr1G302u2Z1SWBWDIJ0TaB-55CXh5enKZe2vwm0V0B1eWCjfNUlW7e30AO3VYSPV8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uev6AZ9RQuaW0?stat-id=3&test-tag=50027808448513&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiNTAwMjc3NzkwOTY1NzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Earn remotely, working 2-3 hours.Free course "Specialist in Internet Advertising" for 10 days! Sign up before 10.09.18To learn moreprofiinternet.artur-grant.ru](https://an.yandex.ru/count/6lt7uy4HHZq50Dm1CUY_ari00000ECgs7402I09Wl0Xe173qmTYm0e01u_CDY06viP23CP01-ewzZosO0UBngS8ge07yZhsFBQW1pEwfmYgu0PYHfjeOm042s068r8uIu06wgvW6w04a-041Y084e0AixfmHkG8ZqKI1OIcI0yW2YehjmVgOsTX_y0A4seZR2lW2We20W82m2u03-8tKHuW3qwcLfmUm0mIu1Fy1w0IR0_W4YBi2Y0M8kmAG1SYG1g05mrwe1OrGi0MDKBW5kyS1m0NrW1Z81Q2D1j05XIJW1Gdm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za6000j40K0002f1_XHq5NHQCL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW29QtpWY02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB4Eq8t9xuKG00yl82Erjr1G302u2Z1SWBWDIJ0TaB-57GLT5enKZe2uYx0l0B1eWClhtUlW7e30AO3ScoP_8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue-6JXvhQuaW0?stat-id=4&test-tag=50027808448513&format-type=24&banner-test-tags=eyI2MDU2MDI3MDY5IjoiNTAwMjc3NzkwOTY1NzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

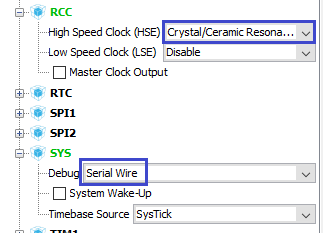
In the [**previous part of**](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-1/) our lesson we got to know more closely the smart LEDs RGB WS2712B, and also prepared a scheme for practical fixing of the material.

I think we have enough theory for today, it's time to move smoothly to the project.

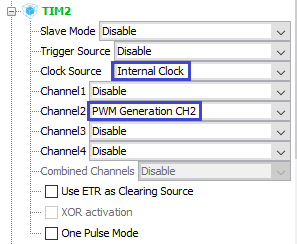
We will create a new project. To do this, run the project generators Cube MX and select our controller



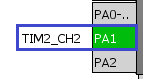
We connect a quartz resonator and a debugger



Turn the timer in PWM mode to 2 channels, since the first one is connected to the PA0 leg, which can always come in handy and it's better not to occupy it

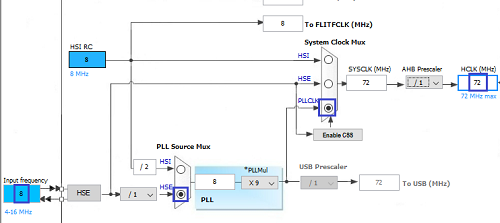


As we can see, the leg **PA1**

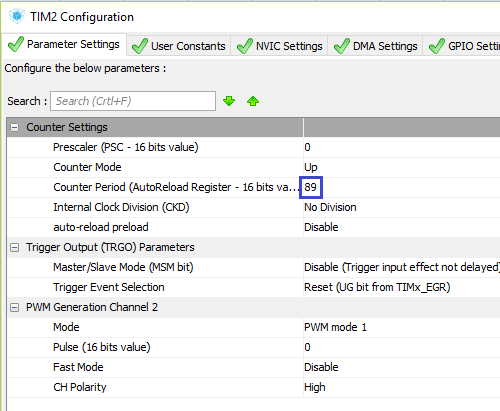


Here we will connect it to the signal wire of the tape.

Go to **Clock Configuration** and adjust everything there as follows (click on the image to enlarge the image)

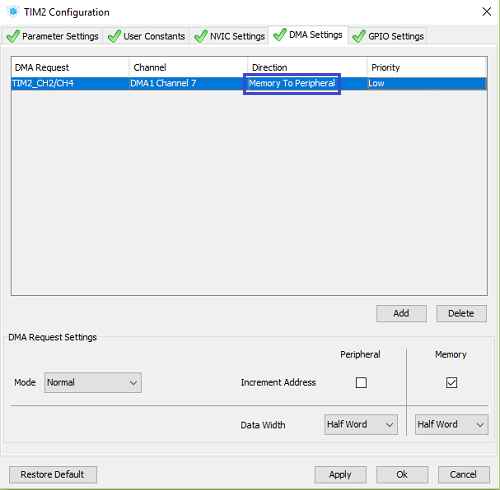
[](http://narodstream.ru/wp-content/uploads/2018/04/stm119img18.png)

Go to **Configuration** and configure the timer



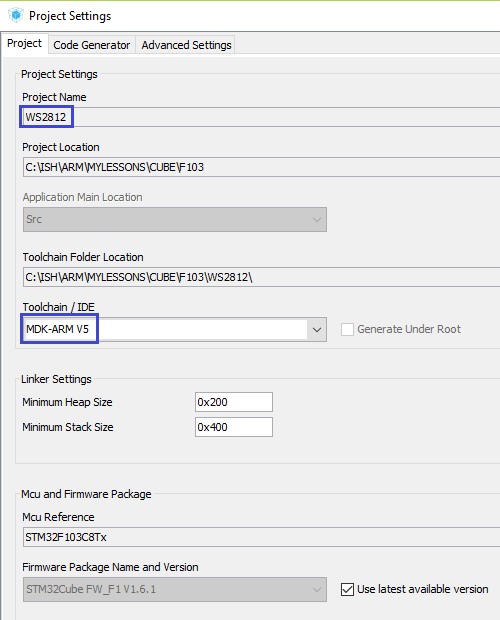
Why 89? And it's easy to explain. Firstly, we always add 1 to the value of the prescaler and period. Therefore, we have 1 predecessor, that is, it is not taken into account. A 89 + 1 = 90. Divide the clock rate of the timer 72 bus by 90 and get exactly 0.8 MHz, which is 800 kilohertz. This is what we need!

Interrupts on the timer do not include, we will not need them. We go to the **DMA** tab and add the DMA channel, the settings leave everything by default (check that they are exactly that). Switch only the direction of work DMA - **Memory To Peripheral**



And interrupts on the DMA will turn on themselves.

It seems so far. Fill in the project settings



Save the settings. generate the project for Keil, open it, configure the programmer for autocut and adjust the optimization level **1** .

First let's try to light at least one LED. Since we know how PWM works, it will be easy for us to do. True, we do not know how it works with DMA, well, nothing, we'll figure it out.

To apply DMA to PWM, and in general in any timer modes, we will have to transmit certain data at the start of the timer, and with the use of DMA we can transfer entire arrays.

In the case of PWM in the array, the pointer to which we will give the start function, there will be data with a duty cycle, or more precisely with a period during which the leg controlled by the timer will be in a high state. In our case, we have a total period of 90 (in settings 89). If we give the timer the number 44 (which is 45-1, which is exactly half the period), then we will get the period of finding the stem in the high state - 45, therefore, the rest of the period, which is also 45, the leg will be in low state. In this way, we get even impulses with a duty cycle of 50 percent. But we do not need such impulses. Therefore, first calculate the value of our period for the transmission of bit 0 to our bus. It is 0.35 microseconds on technical documentation. The total period of the bit is 1.25 microseconds. Therefore, we get the proportion:

Let's create a library for our smart LEDs right away. It will consist of a pair of files **ws2812.c** and **ws2812.h as** follows

**ws2812.c** :

**#include "ws2812.h"**

**//----------------------------------------------------------------------------**

**extern TIM\_HandleTypeDef htim2;**

**//----------------------------------------------------------------------------**

**ws2812.h** :

**#ifndef WS2812\_H\_**

**#define WS2812\_H\_**

**//--------------------------------------------------**

**#include "stm32f1xx\_hal.h"**

**#include <string.h>**

**#include <stdlib.h>**

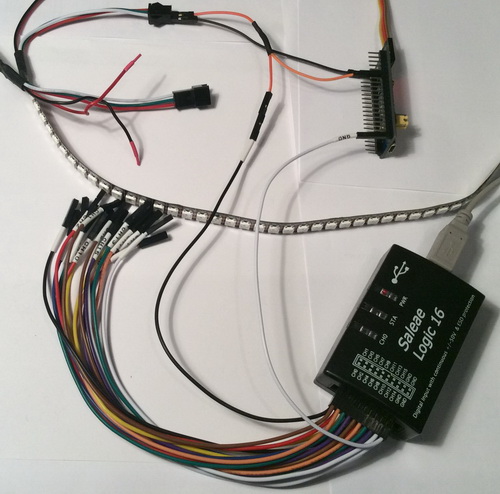
**#include <stdint.h>**

**//--------------------------------------------------**

**//--------------------------------------------------**

**#endif /\* WS2812\_H\_ \*/**

Connect the logical analyzer to our informative leg to see that we are doing everything right, well or wrong (click on the picture to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/04/stm119img22.jpeg)

Let's create a buffer for PWM periods. The buffer will be rather big. Now we'll count. First, it's better to immediately wait more than 50 microseconds for the tape to understand that this is new data. This can be done with a series of zeros. Timer period we have 1.25 microseconds, I think 48 such periods will be quite abound. Therefore, before we organize the buffer, we will **write**  it in the file ws2812.h as a macro

#include <stdint.h>

//--------------------------------------------------

**#define DELAY\_LEN 48**

Also we will add the macro of the number of controlled LEDs. We have 144 of them

#define DELAY\_LEN 48

**#define LED\_COUNT 144**

And now the macro of the length of our buffer, since we remember that for each LED, 24 bits are required, therefore, there should be 24 words in the buffer, since one half-word is the settings for the transmission of one bit. And half-word because we have a 16-bit timer

#define LED\_COUNT 144

**#define ARRAY\_LEN DELAY\_LEN + LED\_COUNT\*24**

Let's  **go to ws2812.c** and add a global array for our buffer, which we will pass to the timer start function

extern TIM\_HandleTypeDef htim2;

**//------------------------------------------------------------------**

**uint16\_t BUF\_DMA [ARRAY\_LEN] = {0};**

**//------------------------------------------------------------------**

Let's go to the **main.c** file and connect our library there

/\* USER CODE BEGIN Includes \*/

**#include "ws2812.h"**

**#include <string.h>**

**#include <stdlib.h>**

**#include <stdint.h>**

/\* USER CODE END Includes \*/

Also connect our global buffer here

/\* Private variables ---------------------------------------------------------\*/

**extern uint16\_t BUF\_DMA [ARRAY\_LEN];**

/\* USER CODE END PV \*/

The buffer has been prepared.

Only we will not need it now. We still want to understand how PWM works with DMA in general. So let's give only one buffer element for now.

To do this, we will write a number at the very beginning of the buffer. And let's not some, but immediately the number **26** . Although we calculated for zero and 24, but it works exactly 26.

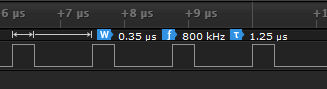
In the main () function, write this number to the zero element of the buffer and call the start function of the timer, in which we will add the address of our buffer as the input parameter, and also the number of elements as the next parameter. While we send one

/\* USER CODE BEGIN 2 \*/

**BUF\_DMA[0] = 26;**

**HAL\_TIM\_PWM\_Start\_DMA(&htim2,TIM\_CHANNEL\_2,(uint32\_t\*)&BUF\_DMA,1);**

We will collect the code, we will sew the controller. The power supply is not fed to the tape yet, in order to avoid accidental ignition of all LEDs with full brightness. Let's see the result in the program of logical analysis



We see that everything is perfect for us. Only we also see that the impulses follow endlessly, we do not need this. However, it should be so, since we started the timer and did not stop it.

We need to find a place to stop it. This place will be an interrupt handler from the DMA peripherals. Therefore, go to the file **stm32f1xx\_it.c** and connect the handle of our timer

/\* USER CODE BEGIN 0 \*/

**extern TIM\_HandleTypeDef htim2;**

/\* USER CODE END 0 \*/

And then in the appropriate place in the interrupt handler from DMA stop the timer

void DMA1\_Channel7\_IRQHandler(void)

{

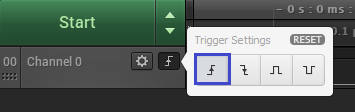
  /\* USER CODE BEGIN DMA1\_Channel7\_IRQn 0 \*/

**HAL\_TIM\_PWM\_Stop\_DMA(&htim2,TIM\_CHANNEL\_2);**

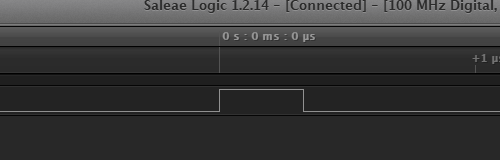
Let's return to the **main.c** file and change the number of elements to 2 in the **main ()** function . With one it will not work

HAL\_TIM\_PWM\_Start\_DMA(&htim2,TIM\_CHANNEL\_2,(uint32\_t\*)&BUF\_DMA,**2**);

Before flashing the controller, configure the logical analysis program. Set the trigger to the rising edge



We will collect the code, we will impose the controller, and we will see that now we have only one pulse for information contact



Delete the string with the initialization of the first element of the buffer

~~BUF\_DMA [ 0 ] = 26 ;~~

Now, go back to the **ws2812.h** header file  and write macros for the PWM settings for zero and one. For the unit, the number **65**

#define ARRAY\_LEN DELAY\_LEN + LED\_COUNT\*24

**#define HIGH 65**

**#define LOW 26**

Let's **go to ws2812.c** and write the initialization function for our tape. We are here just to score all the bytes, except for the delayed numbers for transferring zero to our signal wire

**//------------------------------------------------------------------**

**void ws2812\_init(void)**

**{**

**int i;**

**for(i=DELAY\_LEN;i<ARRAY\_LEN;i++) BUF\_DMA[i] = LOW;**

**}**

**//------------------------------------------------------------------**

Let's write a prototype for this function and call it in **main.c** in **main ()**

/\* USER CODE BEGIN 2 \*/

**ws2812\_init();**

In the file  **ws2812.h,** add the macro setting the bit in any number

#define LOW 26

//--------------------------------------------------

**#define BitIsSet(reg, bit) ((reg & (1<<bit)) != 0)**

**//--------------------------------------------------**

Let's return to  **ws2812.c** and add a function that will fill the buffer in the position corresponding to the LED's defined values ​​for its colors. Accordingly, the incoming parameters will be the values ​​of each color, as well as the number of the LED in the tape

**//------------------------------------------------------------------**

**void ws2812\_pixel\_rgb\_to\_buf\_dma(uint8\_t Rpixel , uint8\_t Gpixel, uint8\_t Bpixel, uint16\_tposX)**

**{**

**volatile uint16\_t i;**

**for(i=0;i<8;i++)**

**{**

**if (BitIsSet(Rpixel,(7-i)) == 1)**

**{**

**BUF\_DMA[DELAY\_LEN+posX\*24+i+8] = HIGH;**

**}else**

**{**

**BUF\_DMA[DELAY\_LEN+posX\*24+i+8] = LOW;**

**}**

**if (BitIsSet(Gpixel,(7-i)) == 1)**

**{**

**BUF\_DMA[DELAY\_LEN+posX\*24+i+0] = HIGH;**

**}else**

**{**

**BUF\_DMA[DELAY\_LEN+posX\*24+i+0] = LOW;**

**}**

**if (BitIsSet(Bpixel,(7-i)) == 1)**

**{**

**BUF\_DMA[DELAY\_LEN+posX\*24+i+16] = HIGH;**

**}else**

**{**

**BUF\_DMA[DELAY\_LEN+posX\*24+i+16] = LOW;**

**}**

**}**

**}**

**//------------------------------------------------------------------**

With this function, I think everything is clear. We distribute the values ​​of the elements depending on the colors and knowing that the green is transmitted first, then red, then blue.

Let's write a prototype for this function and call it in the **main.c** file in the **main ()** function , thereby trying to color the very first light-emitting diode of our tape, for example, in green. In calling the timer start function, we can already pass as an argument our entire buffer, since it is clogged with zero levels

**ws2812\_pixel\_rgb\_to\_buf\_dma(0, 128, 0, 0);**

HAL\_TIM\_PWM\_Start\_DMA(&htim2,TIM\_CHANNEL\_2,(uint32\_t\*)&BUF\_DMA,**ARRAY\_LEN**);

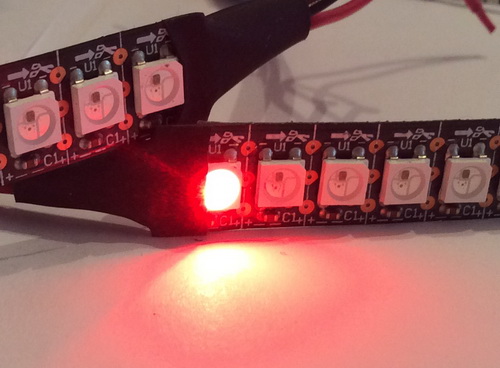
We will connect the power of the tape, we will collect the code, we will sew the controller and see the result



Let's light this LED in a different color, for example red (again use only half the glow - 128)

ws2812\_pixel\_rgb\_to\_buf\_dma ( **128 , 0 ,**0 , 0 );

Let's see the result



Now try to paint in blue, and not this one, but for example the fourth LED

ws2812\_pixel\_rgb\_to\_buf\_dma ( **0 ,**0 , **128 , 3** );

Let's see the result



Let's light the first 4 LEDs now with different colors, not forgetting our golden rule - the total sum of the luminescence units of all LED colors should not exceed 128, which is 1/6 of the intensity of the glow with a full white color

ws2812\_pixel\_rgb\_to\_buf\_dma(**128, 0, 0, 0**);

ws2812\_pixel\_rgb\_to\_buf\_dma(**0, 128, 0, 1**);

ws2812\_pixel\_rgb\_to\_buf\_dma(**0, 0, 128, 2**);

ws2812\_pixel\_rgb\_to\_buf\_dma(**64, 64, 0, 3**);

Let's see the result



Excellent! So we are moving in the right direction.

To initialize each time each LED is not very convenient. So let's expand the functionality of our library.

Also, for the same purpose, we will not initialize all the LEDs, but only 12, the remaining 12 will repeat the color of these 12 and so on. Therefore, we will need to write the corresponding function, which our 12 colors will distribute along the remaining sections of the tape with 12 LEDs. There are even two such functions. The first of them will first enter the bit values ​​of only 12 LEDs into the corresponding temporary buffer, and the second - to distribute it along the main buffer, which will then go to work.

In the file  **ws2812.c** add a temporary buffer that will be two-dimensional, its first size will be the number of LEDs, and the second - the number of colors

uint16\_t BUF\_DMA [ARRAY\_LEN] = {0};

**uint8\_t rgb\_temp[12][3];**

After the function  **ws2812\_pixel\_rgb\_to\_buf\_dma** write the function of preparing a temporary buffer

**//------------------------------------------------------------------**

**void ws2812\_prepareValue (uint8\_t r00, uint8\_t g00, uint8\_t b00,**

**uint8\_t r01, uint8\_t g01, uint8\_t b01,**

**uint8\_t r02, uint8\_t g02, uint8\_t b02,**

**uint8\_t r03, uint8\_t g03, uint8\_t b03,**

**uint8\_t r04, uint8\_t g04, uint8\_t b04,**

**uint8\_t r05, uint8\_t g05, uint8\_t b05,**

**uint8\_t r06, uint8\_t g06, uint8\_t b06,**

**uint8\_t r07, uint8\_t g07, uint8\_t b07,**

**uint8\_t r08, uint8\_t g08, uint8\_t b08,**

**uint8\_t r09, uint8\_t g09, uint8\_t b09,**

**uint8\_t r10, uint8\_t g10, uint8\_t b10,**

**uint8\_t r11, uint8\_t g11, uint8\_t b11)**

**{**

**rgb\_temp[0][0]=r00; rgb\_temp[0][1]=g00; rgb\_temp[0][2]=b00;**

**rgb\_temp[1][0]=r01; rgb\_temp[1][1]=g01; rgb\_temp[1][2]=b01;**

**rgb\_temp[2][0]=r02; rgb\_temp[2][1]=g02; rgb\_temp[2][2]=b02;**

**rgb\_temp[3][0]=r03; rgb\_temp[3][1]=g03; rgb\_temp[3][2]=b03;**

**rgb\_temp[4][0]=r04; rgb\_temp[4][1]=g04; rgb\_temp[4][2]=b04;**

**rgb\_temp[5][0]=r05; rgb\_temp[5][1]=g05; rgb\_temp[5][2]=b05;**

**rgb\_temp[6][0]=r06; rgb\_temp[6][1]=g06; rgb\_temp[6][2]=b06;**

**rgb\_temp[7][0]=r07; rgb\_temp[7][1]=g07; rgb\_temp[7][2]=b07;**

**rgb\_temp[8][0]=r08; rgb\_temp[8][1]=g08; rgb\_temp[8][2]=b08;**

**rgb\_temp[9][0]=r09; rgb\_temp[9][1]=g09; rgb\_temp[9][2]=b09;**

**rgb\_temp[10][0]=r10;rgb\_temp[10][1]=g10;rgb\_temp[10][2]=b10;**

**rgb\_temp[11][0]=r11;rgb\_temp[11][1]=g11;rgb\_temp[11][2]=b11;**

**}**

**//------------------------------------------------------------------**

Now the next function, which from the temporary buffer will move the data to our permanent, at the same time and propagate the sections

**//------------------------------------------------------------------**

**void ws2812\_setValue(void)**

**{**

**uint8\_t n=0;**

**for(n=0;n<12;n++)**

**{**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[0][0], rgb\_temp[0][1], rgb\_temp[0][2], n\*12);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[1][0], rgb\_temp[1][1], rgb\_temp[1][2], n\*12+1);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[2][0], rgb\_temp[2][1], rgb\_temp[2][2], n\*12+2);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[3][0], rgb\_temp[3][1], rgb\_temp[3][2], n\*12+3);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[4][0], rgb\_temp[4][1], rgb\_temp[4][2], n\*12+4);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[5][0], rgb\_temp[5][1], rgb\_temp[5][2], n\*12+5);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[6][0], rgb\_temp[6][1], rgb\_temp[6][2], n\*12+6);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[7][0], rgb\_temp[7][1], rgb\_temp[7][2], n\*12+7);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[8][0], rgb\_temp[8][1], rgb\_temp[8][2], n\*12+8);**

**ws2812\_pixel\_rgb\_to\_buf\_dma( rgb\_temp[9][0], rgb\_temp[9][1], rgb\_temp[9][2], n\*12+9);**

**ws2812\_pixel\_rgb\_to\_buf\_dma(rgb\_temp[10][0],rgb\_temp[10][1],rgb\_temp[10][2],n\*12+10);**

**ws2812\_pixel\_rgb\_to\_buf\_dma(rgb\_temp[11][0],rgb\_temp[11][1],rgb\_temp[11][2],n\*12+11);**

**}**

**}**

**//------------------------------------------------------------------**

And the next function will call the timer start function. The meaning of this function is simplicity, since the official one is too large, and we will always have the same arguments

**//------------------------------------------------------------------**

**void ws2812\_light(void)**

**{**

**HAL\_TIM\_PWM\_Start\_DMA(&htim2,TIM\_CHANNEL\_2,(uint32\_t\*)&BUF\_DMA,ARRAY\_LEN);**

**}**

**//------------------------------------------------------------------**

We add the prototypes to these 3 functions and try to apply them to the **main ()** function of the **main.c**file .

Old code, except for initialization, in the second user section will be deleted and add a new one, it will remain there now this

/\* USER CODE BEGIN 2 \*/

ws2812\_init();

**ws2812\_prepareValue(128,   0,   0,**

**0, 128,   0,**

**0,   0, 128,**

**64,  64,   0,**

**0,  64,  64,**

**64,   0,  64,**

**96,  32,   0,**

**96,   0,  32,**

**32,  96,   0,**

**0,  96,  32,**

**0,  32,  96,**

**32,   0,  96);**

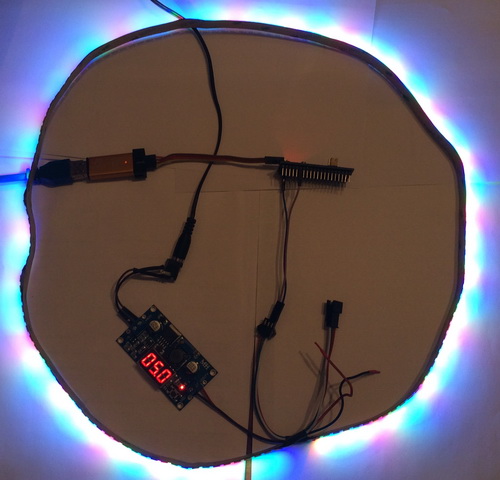
**ws2812\_setValue();**

**ws2812\_light();**

/\* USER CODE END 2 \*/

In principle, we can disable the logical analyzer. Everything is working right for us.

We'll collect the code, we'll sew the controller and get the following result (click on the image to enlarge the image)



Everything works fine and all blocks are correctly repeated.

Just look at the picture - it's still not the same. Therefore at the end of the lesson there is a link to the video tutorial on Youtube, there will be much more beautiful there.

But, of course, it will look even more beautiful when you will assemble this scheme yourself, attach a ribbon and see it all live.

In the **next part of** this lesson, we will write some interesting tests to see how smart LEDs behave beautifully if they are able to control their glow.

**STM Lesson 119. WS2812B. Ribbon on smart RGB LEDs. Part 3**

Posted on [May 16, 2018](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-3/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [1 comment ↓](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-3/#comments)

[Programming LessonsLearn all the in-house cooking **programming** training ! Get the book on the e-mail!To learn moresheremetev.info](https://an.yandex.ru/count/0nYe_mtO8Pm50D01CHR0ari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kGBX6d-Y8cpM3F02mlYjsW7u0eA0W820i1AO0wKVg0CIi0C4k0J_0UW4fWFu1A6s0OW5eRO1a0MRtm6W1TSsg0MhBh05gowu1Tyum0NmcGV81R3g0T05x1JW1J_m1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHs7RUSyL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB4639nmhuKG00jZh6FLjr1G302u2Z1SWBWDIJ0TaB-57OTjvpnKZe2w6s0V0B1eWCjfNUlW7e30AO3VYSPV8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ueu7JsFlSuaW0?stat-id=3&test-tag=80814134026241&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODA4MTQxMDQ2NzQzMDQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Dating and chatting on web camerasRegistration and communication on the site are free. There is a guest entrance.To learn morehitlove.ru](https://an.yandex.ru/count/VzZniUizk7450DW1CHV0ari00000ECgs7402I09Wl0Xe1724kxc20O01nfYFzRESyguWY06-YR7-Av01j92OyYAO0UxNkFyYe06CvvxN0QW1gh7jhWAu0OYYjfC2m042s07gYUa2w04E-06s0OW21A02ykA82Ra2uHf_eY9irWpm0hoWZgC8-0A2W820WB0IW0EOdTdG0eW3WxVyxmgO0_7i0h031BW4_m7e1Dq3-0Ilom681Q_B0P05dDm1e0M1FQW5sZEm1Tepk0N-Dy05bu03o0NMOT05eXVW1MBm1G6O1fZr_d6W1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000907Aga7-54qi09qnKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08kR74180A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiGnFIuclXH002dISyzMtK50C0BWAC5o0k0r9C1sGluKJIm0dJ5IEWBhyi1y0i6Y0oiljw-0UWC0fWDoR9dyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZuSgUWDtYI000?stat-id=4&test-tag=80814134026241&format-type=24&banner-test-tags=eyI0NjgzMTM4MDc0IjoiODA4MTQxMDQ2NzQzMDQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of**](http://narodstream.ru/stm-urok-119-ws2812b-lenta-na-umnyx-svetodiodax-rgb-chast-2/) this lesson we created a project, studied the operation of a timer in PWM mode using DMA, and learned how to light certain LEDs in different colors.

Also we wrote a code that allowed us to see the glow of all the LEDs in different colors.

Only this was a static picture. Now let's write some dynamic tests.

The first test we will have - running lights. The truth under running lights means a few lonely lights running in a circle, and we will have all the colors of the LEDs also run in a circle. And, it turns out, this is all done is not very difficult.

Let's move to file  **ws2812.c** and add a function for such a test.

Before we add it, add a global array for another buffer. which will store the colors of only one LED. This buffer is required to move the colors of our LEDs one position towards the very first LED, and then to the very last position we will add the color of the very first. And so that he does not forget, and this buffer is needed

uint8\_t rgb\_temp[12][3];

**uint16\_t DMA\_BUF\_TEMP[24];**

And now the actual function, which we will write to the bottom of our file

**//------------------------------------------------------------------**

**void ws2812\_test01(void)**

**{**

**uint8\_t i,j;**

**ws2812\_prepareValue(128, 0, 0, 0, 128, 0, 0, 0, 128, 64, 64, 0,**

**0, 64, 64, 64, 0, 64, 96, 32, 0, 96, 0, 32,**

**32, 96, 0, 0, 96, 32, 0, 32, 96, 32, 0, 96);**

**ws2812\_setValue();**

**for(j=0;j<50;j++)**

**{**

**memcpy((void\*)DMA\_BUF\_TEMP,(void\*)(BUF\_DMA+48),48);**

**for(i=0;i<143;i++)**

**{**

**memcpy((void\*)(i\*24+BUF\_DMA+48),(void\*)(i\*24+BUF\_DMA+72),48);**

**}**

**memcpy((void\*)(BUF\_DMA+48+3432),(void\*)DMA\_BUF\_TEMP,48);**

**ws2812\_light();**

**HAL\_Delay(100);**

**}**

**}**

**//----------------------------------------------------------------------------**

Now a few words about the contents of the body of this function.

First, we fill the buffer with some colors. Then create a loop of 50 iterations. This is the amount of movement for the running belt. Then we copy the contents of the first position of the main buffer to the temporary buffer. Then we move all the other 143 positions towards the first position. And then we copy the stored pixel from the temporary buffer to the last position. Why just 48, not 24? Because the **memcpy**function works with bytes and not with halfwords. Then we start the timer, the colors will move to 1 position. And then a delay of 100 milliseconds. That's about with such a period and our lights will run, if you do not take into account the execution time of all the code in the cycle.

We write the prototype for this function and call it in a loop function **main ()** the file **main.c**

  /\* USER CODE BEGIN 3 \*/

**ws2812\_test01();**

}

We will collect the code, we will collate the controller and look at the result of the execution. Only show the result, I will not be here, since it will be almost the previous picture for here I can not show the dynamics. So welcome on the channel in Youtube.

But one test is certainly not enough. We'll write one more that will make our LEDs blink smoothly and they will only change the intensity of the glow, but not the color. To do this, we again go to  **ws2812.c** and add another function

**//----------------------------------------------------------------------------------------**

**void ws2812\_test02(void)**

**{**

**uint8\_t i,j,jj;**

**int k=0;**

**for(jj=0;jj<10;jj++)**

**{**

**for(j=0;j<32;j++)**

**{**

**for(i=0;i<144;i++)**

**{**

**k=1000\*(32-j)/32;**

**ws2812\_pixel\_rgb\_to\_buf\_dma(rgb\_temp[i%12][0]\*k/1000,rgb\_temp[i%12][1]\*k/1000,rgb\_temp[i%12][2]\*k/1000,i);**

**}**

**ws2812\_light();**

**HAL\_Delay(10);**

**}**

**for(j=0;j<32;j++)**

**{**

**for(i=0;i<144;i++)**

**{**

**k=1000\*(j+1)/32;**

**ws2812\_pixel\_rgb\_to\_buf\_dma(rgb\_temp[i%12][0]\*k/1000,rgb\_temp[i%12][1]\*k/1000,rgb\_temp[i%12][2]\*k/1000,i);**

**}**

**ws2812\_light();**

**HAL\_Delay(10);**

**}**

**}**

**}**

**//-----------------------------------------------------------------------------**

This function, despite the cycles of triple nesting, is not difficult.

In this function, we add 10 cycles. This is the number of blinking LEDs. Then there are two cycles, in the first of which there is a gradual decrease in the glow of the LEDs, and in the second - a reverse increase. 32 gradations for smoothness, I think that will be enough. In the first of these cycles, we add a cycle of 144 iterations corresponding to the number of our LEDs. In the body of this cycle, which is the cycle of the lowest level, we calculate the coefficient, which will later serve as a multiplier for calculating the colors of this LED. To avoid working with a floating point, which will give an extra load to the ALU, we get a factor that is 1000 times larger than the required one, then we take this into account. Next we call the function of filling the position of the ribbon with colors, in which we take into account our coefficient. Leaving the cycle of the lowest level, we call the timer start function, wait for 10 milliseconds and the process is repeated already with the increased coefficient value. Having reached the maximum value, we find ourselves in the next similar cycle, in which practically everything is done the same, but only with the difference that there is an increase in the coefficient due to the use of a clever multiplier in the calculation.

Also, write a prototype for this function and call it in a loop function **main ()** the file **main.c**

ws2812\_test01();

**ws2812\_test02();**

We will collect the code, we will sew the controller and see the result. I will show one of the frames, again not too lazy to repeat that on video it looks much better



Let's move to file  **ws2812.c** and add the function to the next test, in which we will in the same way all the LEDs flash slowly, only 12 of them will be blinking to decrease, and 12 following this time - upwards, then vice versa. It gives the impression of a very beautiful effect

**ws2812\_test03**

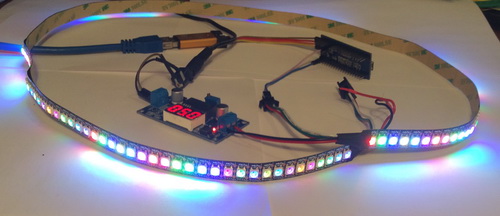
The code is similar to the code of the previous function, only we are already tracking the sections here, thus calculating one equal coefficient for even parcels, and one for odd ones, and the other for odd ones. Therefore, the nesting here has also increased by 1.

Again, add a prototype for this function and call it in a loop function **main ()** the file **main.c**

ws2812\_test02();

**ws2812\_test03();**

We will collect the code, we will tell the controller and we will see the result



Let's  **go to ws2812.c** and add another test, which will run with lights gently lit with one color. To explain this in words is difficult, it is better to see once, at least in the picture

**ws2812\_test04**

Despite the fact that the code is large, it is also uncomplicated, since there is a switch with cases corresponding to certain colors of 6 possible. Therefore, the test function has an input integer argument. The test is repeated 12 times, in the loop we add another cycle of 12 iterations, each of which initializes one of the 12 positions with a color in the temporary buffer. In this loop, we call the **switch statement**which will determine with which base color we want to work. And, depending on the color number, in the corresponding case we put the settings in the corresponding elements of the temporary buffer. After the data has been entered, we call the data distribution function on the main buffer and light our LEDs. And then the code goes the same as in the very first test, that is, our lights similarly run around the circle of the ribbon

We add a prototype to this function and call it in an infinite loop of the **main ()** function of the **main.c**file , using all possible color scales

ws2812\_test03();

**ws2812\_test04(1);**

**ws2812\_test04(2);**

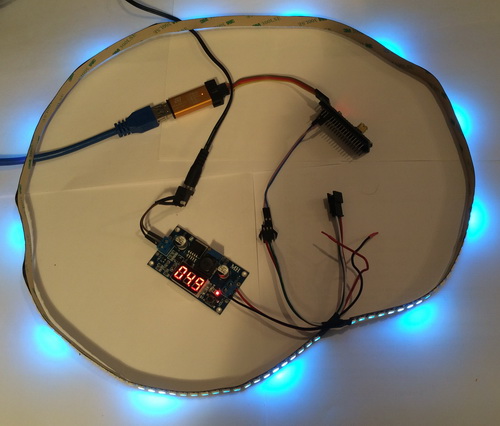
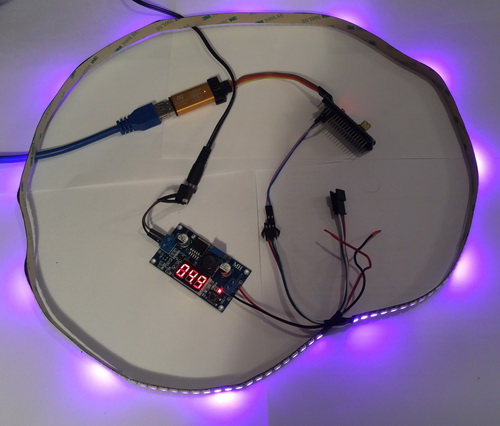
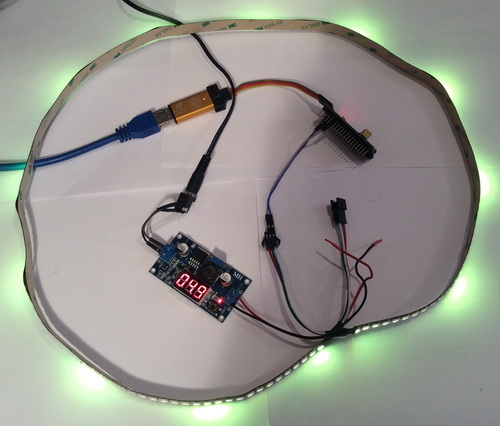
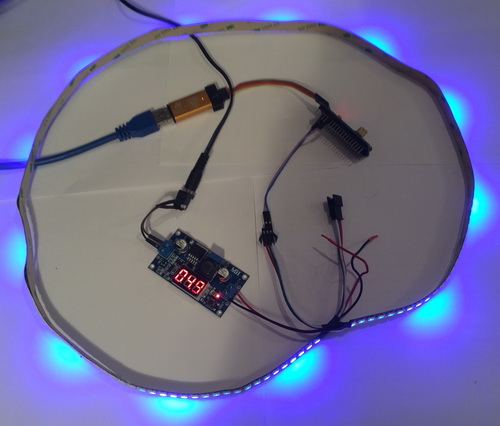
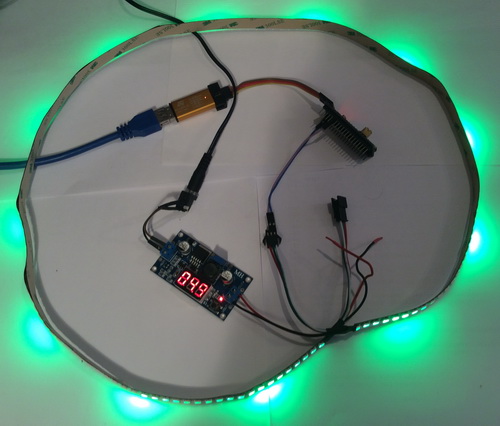
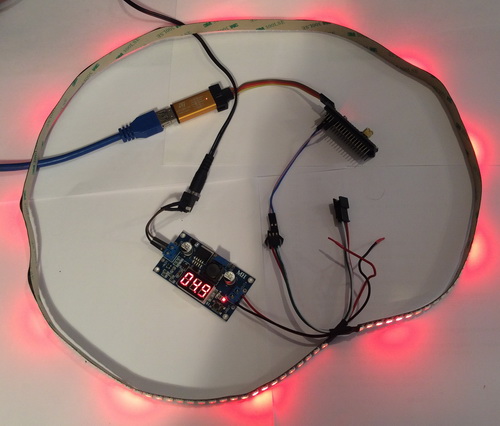
**ws2812\_test04(3);**

**ws2812\_test04(4);**

**ws2812\_test04(5);**

**ws2812\_test04(6);**

We will collect the code, we will tell the controller and we will see the result

Go to  **ws2812.c** and add another test. This test will be the last in this lesson and it will wink each LED from the 12 in the block with some color that will be determined for each of these LEDs in a certain way. Only we must rigorously honor our golden rule - the total glow of the LED - not more than 1/6 of the maximum. The remaining blocks will, respectively, repeat the behavior of this block

**//----------------------------------------------------------------------------**

**void ws2812\_test05(void)**

**{**

**uint8\_t i,j,jj;**

**int k=0;**

**for(jj=0;jj<40;jj++)**

**{**

**for(j=0;j<32;j++)**

**{**

**for(i=0;i<144;i++)**

**{**

**if(j<16) k=1000\*(j)/16;**

**else k=1000\*(32-j)/16;**

**if(i%12==0) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*k/1000,128\*(1000-k)/1000,0,i);**

**if(i%12==1) ws2812\_pixel\_rgb\_to\_buf\_dma(0,128\*k/1000,128\*(1000-k)/1000,i);**

**if(i%12==2) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*k/1000,0,128\*(1000-k)/1000,i);**

**if(i%12==3) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*(1000-k)/1000,128\*k/2000,128\*k/2000,i);**

**if(i%12==4) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*k/2000,128\*k/2000,128\*(1000-k)/1000,i);**

**if(i%12==5) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*k/2000,128\*(1000-k)/1000,128\*k/2000,i);**

**if(i%12==6) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*(1000-k)/2000,128\*(1000-k)/2000,128\*k/1000,i);**

**if(i%12==7) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*k/1000,128\*(1000-k)/2000,128\*(1000-k)/2000,i);**

**if(i%12==8) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*(1000-k)/2000,128\*k/1000,128\*(1000-k)/2000,i);**

**if(i%12==9) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*(1000-k)/3000,128\*(1000-k)/667,128\*k/1000,i);**

**if(i%12==10) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*k/3000,128\*(1000-k)/667,128\*(1000-k)/1000,i);**

**if(i%12==11) ws2812\_pixel\_rgb\_to\_buf\_dma(128\*(1000-k)/3000,128\*k/667,128\*(1000-k)/3000,i);**

**}**

**ws2812\_light();**

**HAL\_Delay(20);**

**}**

**}**

**}**

**//----------------------------------------------------------------------------**

The code is also simple. We do not use the temporary buffer at all, but we work immediately with the main buffer, using for this in the cycle the division by the module. As you can see, for each LED in the base color, some specific and unique colors predominate.

Following tradition, add the prototype for this function and call it in a loop function **main ()** the file **main.c**

ws2812\_test04(6);

**ws2812\_test05();**

We will collect the code, we will sew the controller and see the result. Of course, again looking at it statically - it's not that. It is necessary to watch the video



So, in this lesson we've worked with smart LEDs, learned how to manage them, and also manage them in a chain, using an LED ribbon of 144 LEDs. Also, we once again practiced in working with memory sites, which, I hope, increased our level of erudition in our hard work - programming.

Thank you all for attention!