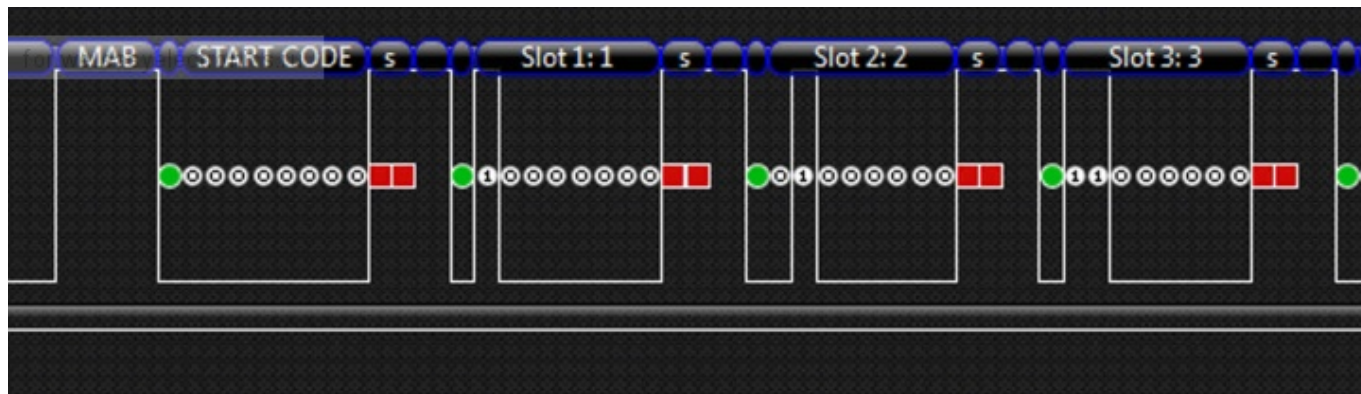


DMX тестер на контроллере STM32

Как-то понадобилось изучить протокол — DMX-512 И научиться правильно «принимать» посылки. В интернете, о протоколе [DMX](#) — информации достаточно [DMX](#) Простите за низкое качество фото.



DMX-512 целиком произошел от стандарта RS-485.

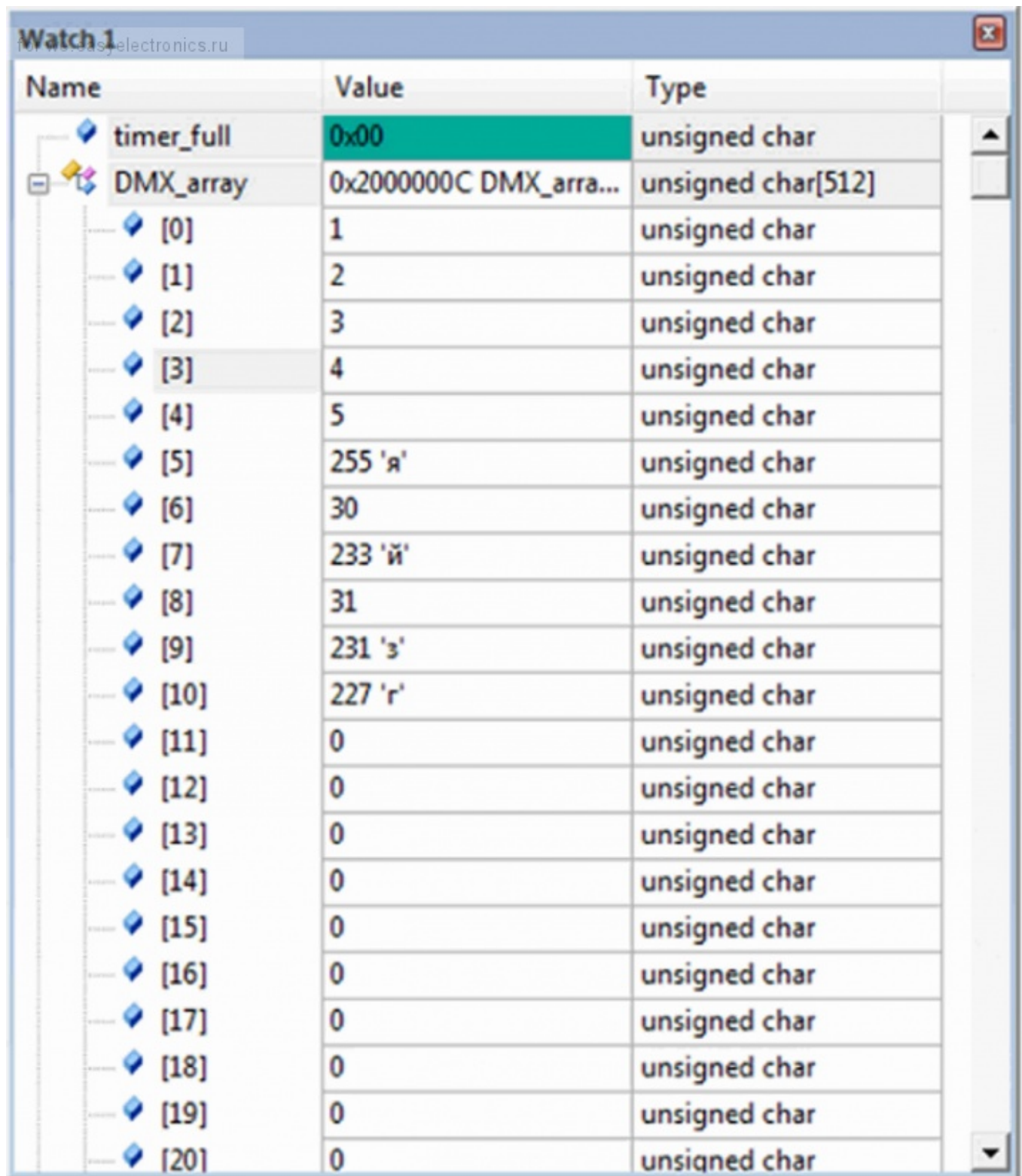
Основное отличие [DMX](#) от RS-485 это то, что в [DMX](#) есть «**Break**». В RS-485 это воспринимается как ошибка передачи данных. В [DMX](#) скорость передачи данных строго определена и составляет 250 кб. в сек. Передача данных осуществляется восьми битным асинхронным протоколом с одним стартовым битом (низкий активный уровень) и двумя стоповыми битами. Один фрейм = 11 бит. Длина пакета 44 микросекунды. Передаваемая информация находится между стартовым и стоповыми битами и имеет 256 уникальных состояний от 0 до 255. Появление низкого уровня после всего пакета (512 фреймов) данных воспринимается как начало нового пакета. Количество каналов = фреймов, считываемых прибором с линии ДМХ зависит от количества функций, заложенных производителем в прибор. Например одноканальный диммер (в смысле на одну нагрузку) отъедает от линии [DMX](#) только один (канал = фрейм). При изменении цифр от 0 до 255 в ДМХ пульте управления, процессор диммера считывает эту информацию и плавно изменяет выходное напряжение на нагрузке от 0 вольт до 220 вольт. Линия ДМХ позволяет управлять пятьсот двенадцатью одноканальными приборами.

Если в приборе указать стартовый адрес ДМХ отличный от 1, например 24, то прибор считывает число только с 24 фрейма. На остальных каналах (фреймах) могут находиться другие приборы. Главное сообщить прибору с какого фрейма он начинает принимать информацию.

<http://www.x-light.ru/dmx.html>

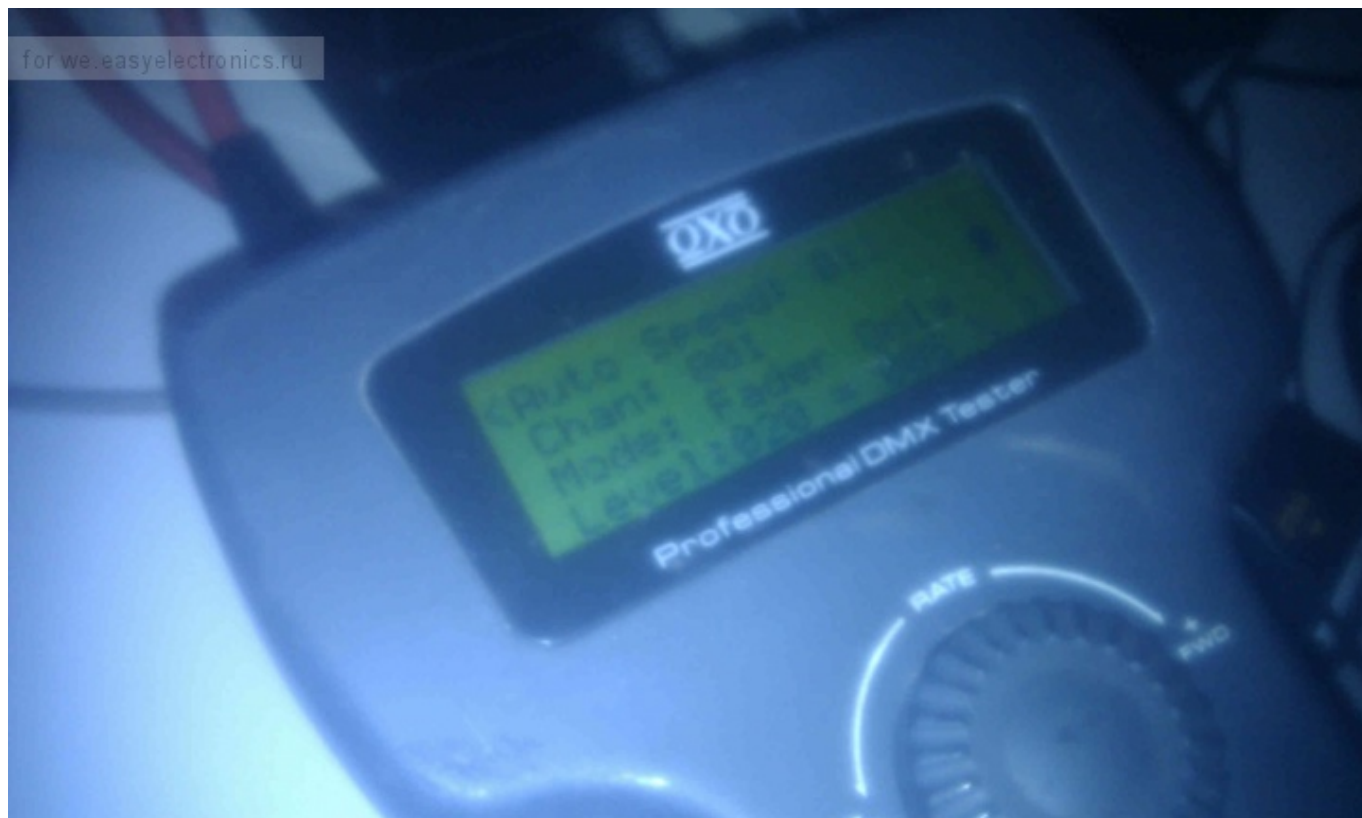
Сборка тестера

Результат работы в массиве `DMX_array[512]`; (Keil, Debugger).



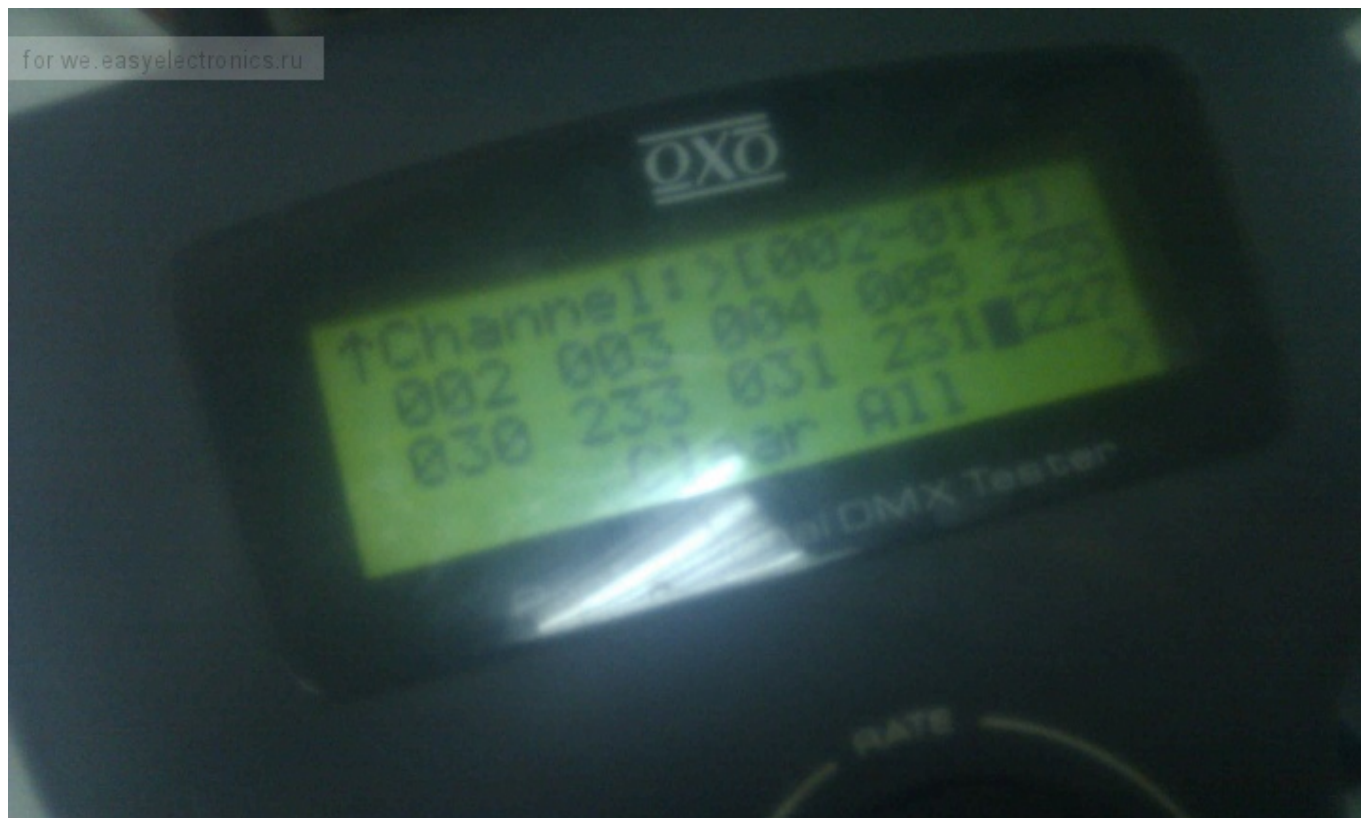
Name	Value	Type
timer_full	0x00	unsigned char
DMX_array	0x2000000C DMX_arr...	unsigned char[512]
[0]	1	unsigned char
[1]	2	unsigned char
[2]	3	unsigned char
[3]	4	unsigned char
[4]	5	unsigned char
[5]	255 'я'	unsigned char
[6]	30	unsigned char
[7]	233 'ћ'	unsigned char
[8]	31	unsigned char
[9]	231 'з'	unsigned char
[10]	227 'г'	unsigned char
[11]	0	unsigned char
[12]	0	unsigned char
[13]	0	unsigned char
[14]	0	unsigned char
[15]	0	unsigned char
[16]	0	unsigned char
[17]	0	unsigned char
[18]	0	unsigned char
[19]	0	unsigned char
[20]	0	unsigned char

DMX тестер посылает «20» в первом канале.



b15ca2

Watch 1		
Name	Value	Type
timer_full	0x00	unsigned char
DMX_array	0x2000000C DMX_arr...	unsigned char[512]
[0]	1	unsigned char
[1]	2	unsigned char
[2]	3	unsigned char
[3]	4	unsigned char
[4]	5	unsigned char
[5]	255 'я'	unsigned char
[6]	30	unsigned char
[7]	233 'ћ'	unsigned char
[8]	31	unsigned char
[9]	231 'з'	unsigned char
[10]	227 'г'	unsigned char
[11]	0	unsigned char
[12]	0	unsigned char
[13]	0	unsigned char
[14]	0	unsigned char
[15]	0	unsigned char
[16]	0	unsigned char
[17]	0	unsigned char
[18]	0	unsigned char
[19]	0	unsigned char
[20]	0	unsigned char



Исходный код тестера

STMDMX512Tester.c

```
#include "stm32f10x.h"
#include "core_cm3.h"

void USART3_init(void);
void _delay_ms (int long);
void USART3_TX_data (unsigned char);
uint8_t USART2_RX_data(void);
void TIM2_enabled(uint8_t, long int arr, long int psc);
void Printf(char *data);
void EXTI3_enabled(uint8_t on);
uint8_t USART3_RX_data(void);

uint8_t timer_full;

uint8_t start_bit;

uint8_t DMX_array[512];

unsigned char temp;

int long reset;

long int i;
```

```
int main(void) {

    RCC -> APB2ENR |= RCC_APB2ENR_IOPBEN;
    RCC -> APB1ENR |= RCC_APB1ENR_USART2EN;
    RCC -> APB1ENR |= RCC_APB1ENR_USART3EN;
    RCC -> APB2ENR |= RCC_APB2ENR_IOPAEN;
    RCC -> APB2ENR |= RCC_APB2ENR_IOPDEN;
    RCC-> APB2ENR  |= RCC_APB2ENR_AFIOEN;
    RCC-> APB1ENR  |= RCC_APB1ENR_TIM2EN;

    GPIOB->CRH|=GPIO_CRH_MODE10;          GPIOB->CRH |=
GPIO_CRH_CNF10; GPIOB->CRH &=~ GPIO_CRH_CNF10_0; // PORTB 10 - out
    GPIOB->CRH &=~ GPIO_CRH_MODE11;        GPIOB->CRH |=
GPIO_CRH_CNF11; GPIOB->CRH &=~ GPIO_CRH_CNF11_0; // PORTA 3 - input
    [[:wiki:dmx_512|DMX]]
    GPIOD->CRH |= GPIO_CRH_MODE11;        GPIOD->CRH &=~
GPIO_CRH_CNF11;                          // PORTD 9 - out
    USART_debuger
    GPIOB->CRH &=~ GPIO_CRH_MODE9;          GPIOB->CRH |=
GPIO_CRH_CNF9; GPIOB->CRH &=~ GPIO_CRH_CNF9_0;
    RCC -> CFGR = RCC_CFGR_SW_1; // PLL-on

    USART3_init();

    while (1)
    {

        // Основной цикл

        if(!(GPIOB->IDR & GPIO_IDR_IDR11)) //Начало, start_bit, timer_full
=0
        {
            if(start_bit==0)
            {
                EXTI3_enabled(1); // Включить внешнее прерывание на
DMX_input
                TIM2_enabled(1, 80, 100); // На 115 Мкс
                start_bit=1;
            }
        } // Выходим, крутимся пока не сработает таймер.
        // Пока он считает до 115 Мкс - внешних прерываний быть не должно.
        // Если будут - reset++, и будет больше 0, значит это не стартовый
бит

        if(timer_full==1) // первое прерывания срабатывает сразу после
включения внешнего прерывания
        {
            if(reset<=1) // Не больше 1
            {
```

```

        while(!(GPIOB->IDR & GPIO_IDR_IDR11)){}; // Подождем
        когда закончится низкий уровень стартового бита,

                                                    // сюда надо
        добавить сторожа на заикливание!!
        EXTI3_enabled(0); // Внешние прерывания выключить

        // Определение MAB
        timer_full=0;
        TIM2_enabled(1, 5, 100); // На 9,500 Мкс
        reset=0;
        EXTI3_enabled(1); // Внешние прерывания включить

        while(timer_full==0){}; // Ждем прерывания таймера через
        9,500 Мкс

        if(reset==0) // Перепадов быть не должно
        {
            temp = USART3_RX_data();
            TIM2_enabled(0, 0, 0); // Выключить
            reset=0;
            EXTI3_enabled(0); // Выключить
            // Первый бит
            start_bit = USART3_RX_data();
            if(start_bit==0){ // Должен быть 0
                // Старт DMX распознан правильно, попадаем к данным
                // Цикл чтения 512 байтов

                for(i=0;i<512;i++)
                {
                    DMX_array[i]= USART3_RX_data();

                }

                i=0; // Все обнулить
                timer_full=0;
                start_bit=0;
                TIM2_enabled(0,0,0);
                reset=0;
            }}

            // Не стартовый бит, ошибка
            timer_full=0;
            start_bit=0;
            reset=0;
            TIM2_enabled(0,0,0);
        }
    }
}

```

```
void EXTI3_enabled(uint8_t on)
{
    if(on==1){
        AFIO->EXTICR [0] |= AFIO_EXTICR1_EXTI3_PA; // Прерывание на PIND 3
        NVIC_EnableIRQ (EXTI3_IRQn); // Разрешить прерывание
        EXTI->IMR |= EXTI_IMR_MR3; // Прерывание
        EXTI->RTSR |= EXTI_IMR_MR3; // Спаду
        EXTI->FTSR |= EXTI_IMR_MR3; // Подъему
    } else
    {
        AFIO->EXTICR [0] &=~ AFIO_EXTICR1_EXTI3_PA; // Прерывание на PIND 3
        NVIC_EnableIRQ (EXTI3_IRQn); // Разрешить прерывание
        EXTI->IMR &=~ EXTI_IMR_MR3;
        EXTI->EMR &=~ EXTI_IMR_MR3;
        EXTI->RTSR &=~ EXTI_IMR_MR3;
        EXTI->FTSR &=~ EXTI_IMR_MR3;
    }
}

void EXTI3_IRQHandler (void)
{
    if (EXTI->PR & (1<<3))
    {
        EXTI->PR |= (1<<3); // Сбросить флаг EXTI3
        reset++; // Если по выходу переменная будет > 0, значит это не стартовый
        бит
    }
}

void TIM2_IRQHandler (void) // TIM2 INTERRUPT
{
    TIM2->SR &= ~TIM_SR_UIF;
    timer_full++;
    start_bit=0;
    TIM2_enabled(0,0,0);
}

void TIM2_enabled(uint8_t condition, long int arr, long int psc){

    if(condition==1)
    {
        RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
        NVIC_EnableIRQ(TIM2_IRQn);
        TIM2->ARR = arr;
        TIM2->PSC = psc;
        TIM2->CR1 |= TIM_CR1_CEN;
        TIM2->DIER |= TIM_DIER_UIE;
        TIM2->CNT = 0x00;
    }
}
```

```
if(condition==0)
{
    RCC->APB1ENR &=~ RCC_APB1ENR_TIM2EN;
    NVIC_DisableIRQ(TIM2_IRQn);
    TIM2->ARR = arr;
    TIM2->PSC = psc;
    TIM2->CR1 &=~ TIM_CR1_CEN;
    TIM2->DIER &=~ TIM_DIER_UIE;
    TIM2->CNT = 0x00;
}
}

void USART3_init(void){
    USART3->CR1 |= (USART_CR1_TE)|(USART_CR1_RE);
    USART3->BRR= 0x00000120; // 250000 kb/s (72 000 000 + 250 000 /
2) / 250 000
    USART3->CR1 |=USART_CR1_UE;
    USART3->CR2 |= USART_CR2_STOP_1;
}

void USART3_TX_data(uint8_t data){
    while (!(USART3->SR & USART_SR_TXE)) {}
    USART3->DR=data;
    while (!(USART3->SR & USART_SR_TXE)) {}
    USART3->SR &=~ USART_SR_TXE;
}

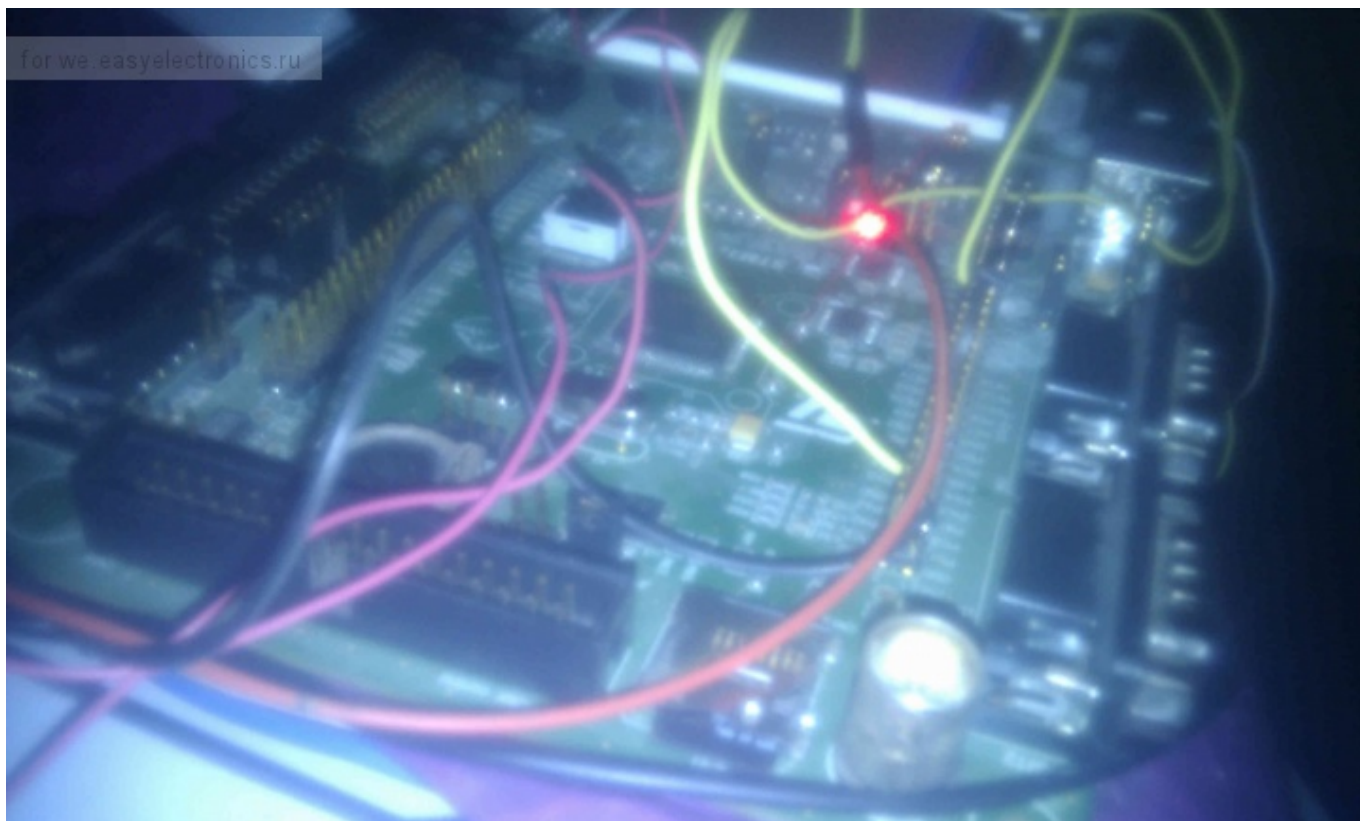
void Printf(char *data){
    while(*data)
    {
        while (!(USART3->SR & USART_SR_TXE)) {}
        USART3->DR= *data++;
        while (!(USART3->SR & USART_SR_TXE)) {}
    }
}

uint8_t USART3_RX_data(void){
    uint8_t a;

    while (!(USART3->SR & USART_SR_RXNE)){ };
    a = USART3->DR;
    return a;
}

void _delay_ms (int long delay){
    int long a;
    for (a=0; a<delay; a++){}}
```

Отлаживалось на демоплате STM_Eval-Board-STM32 (STM32F103VB)



Спасибо за статью [khomin](#)

Комплектующие

Купить [STM32F103](#) на [Aliexpress](#) ~ 350-500руб.

From:
<https://dmx-512.ru/> - **DMX512.RU Управление светом**

Permanent link:
https://dmx-512.ru/zheleznaja_chast/dmx-512_tester_na_mikrokontrollere_stm32?rev=1476375025

Last update: **2017/06/09 20:04**

