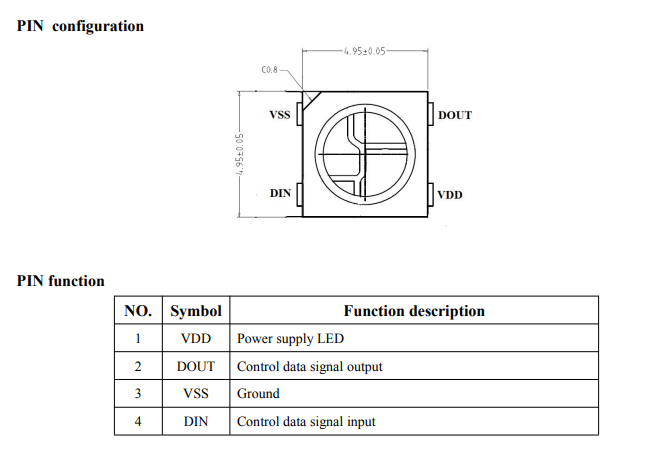
# WS2812小灯

## 1.简介

WS2812是一种可编程LED灯，可显示2^24种颜色。

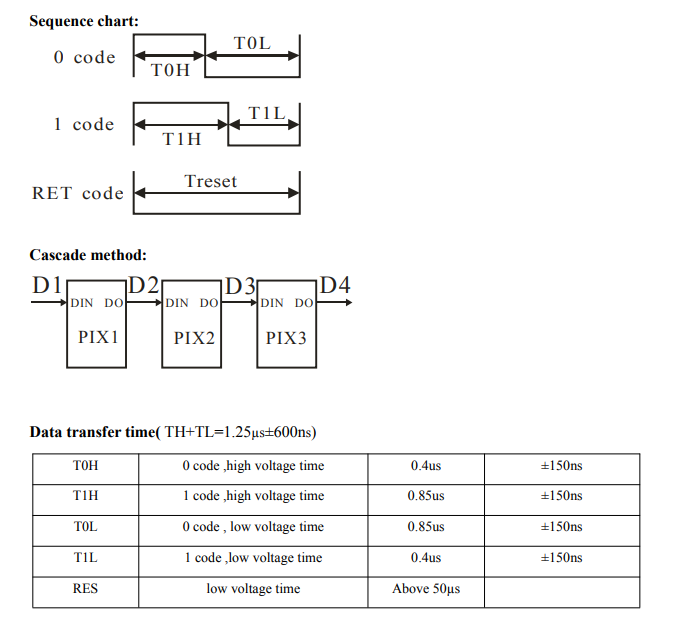
引脚功能如下：



## 2.时序结构

WS2812采用单总线通讯方式，根据总线上高低电平的持续时间来分辨0和1。数据从第一个小灯的DIN引脚进入，第一个小灯会先锁存24bit数据，之后的数据由WS2812内部整形放大后再从DOUT引脚输出到第二个小灯，以此类推，实现但总线控制多个小灯。

通讯时序如下：（注：不同厂家不同具体型号的WS2812时序可能不同，具体时序标准以使用的WS2812硬件手册为准）

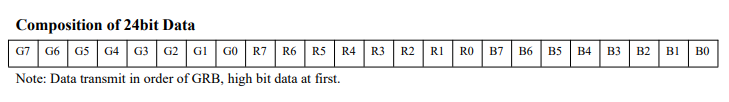


数据0：高电平持续T0H，低电平持续T0L

数据1：高电平持续T1H，低电平持续T1L

RESET：低电平持续50us以上，可在数据传输结束后拉低表示这次传输结束,因此数据的发送间隔不能超过50us，否则第下一个灯的数据会被上一个灯接收。

在一个固定周期内，可以通过调整高低电平所占的比例，确定输出是0还是1.（通常情况下常设置周期为1.25us。



一个LED灯需要24bit的颜色数据来控制，分别为8位绿色+8位红色+8位绿色，高位在前，每个颜色可实现256级亮度显示，通过组合实现显示2^24种颜色。

## 3.控制方式

以下例子使用的MCU为沁恒的CH32V307VCT6，频率72Mhz。

### 3.1 延时方式

这种方式需要设定一个很短的延时时间，依据WS2812的时序翻转引脚电平。这种方式使用简单，只需要一个普通的GPIO引脚，容易移植，但缺点是比较占用MCU资源，控制精度也不高。驱动解析如下：

头文件如下所示：

#ifndef \_\_WS2812B\_H

#define \_\_WS2812B\_H

#include "debug.h"

#define WS2812B\_GPIO\_PORT       GPIOA

#define WS2812B\_GPIO\_CLK        RCC\_APB2Periph\_GPIOA

#define WS2812B\_GPIO\_PIN        GPIO\_Pin\_15

void WS\_Init();

//======================================================================

//函数名称：ws\_sendOnePix

//函数参数：ptr：代表灯珠颜色的字节数组;num:灯珠数量

//函数返回：无

//功能概要：设置ws2812灯珠的颜色

//======================================================================

void WS\_SendOnePix(uint8\_t \*ptr,uint8\_t num);

#endif

首先我们需要完成GPIO的初始化，控制WS2812的GPIO只需要使用任意的普通GPIO，配置成推挽输出模式即可。

void WS\_Init()

{

    GPIO\_InitTypeDef GPIO\_InitStructure = {0};

    RCC\_APB2PeriphClockCmd(WS2812B\_GPIO\_CLK, ENABLE);

    GPIO\_InitStructure.GPIO\_Pin = WS2812B\_GPIO\_PIN;

    GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

    GPIO\_Init(WS2812B\_GPIO\_PORT, &GPIO\_InitStructure);

    Delay\_Ms(10);

}

1个灯珠需要3个字节的颜色数据，那我们可以定义1个8位数组，以第1个灯珠到最后1个灯珠的顺序，依次填入3个字节的GRB颜色数据，从高位逐一发送。发送需要按照时序调整总线电平（不同厂家不同型号的WS2812时序可能不同），延时可使用\_\_asm volatile ("nop");，在72MHz频率下，执行一次\_\_asm volatile ("nop");大约需要13.88ns，可以此调整延时时间。发送程序如下：

void WS\_SendOnePix(uint8\_t \*ptr,uint8\_t num)

{

    uint8\_t i,j,temp;

    for(j=0;j<num\*3;j++)//一个灯珠需要3个字节的颜色数据

    {

        temp=ptr[j];

        for(i=0;i<8;i++)

        {

            if(temp&0x80)    //从高位开始发送

            {

                GPIO\_SetBits(WS2812B\_GPIO\_PORT, WS2812B\_GPIO\_PIN);  //发送“1”码,高电平750ns-1.6us,低电平220ns-420ns

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                 \_\_asm volatile ("nop");

                 \_\_asm volatile ("nop");

                 \_\_asm volatile ("nop");

                 \_\_asm volatile ("nop");

                 \_\_asm volatile ("nop");

                 \_\_asm volatile ("nop");

                GPIO\_ResetBits(WS2812B\_GPIO\_PORT, WS2812B\_GPIO\_PIN);

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

            }

            else             //发送“0”码，高电平220ns-420ns，低电平750ns-1.6us

            {

                GPIO\_SetBits(WS2812B\_GPIO\_PORT, WS2812B\_GPIO\_PIN);

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                GPIO\_ResetBits(WS2812B\_GPIO\_PORT, WS2812B\_GPIO\_PIN);

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

                \_\_asm volatile ("nop");

            }

            temp=(temp<<1);    //左移一位，发送下一字节

        }

    }

    GPIO\_ResetBits(WS2812B\_GPIO\_PORT, WS2812B\_GPIO\_PIN);

    Delay\_Us(500);

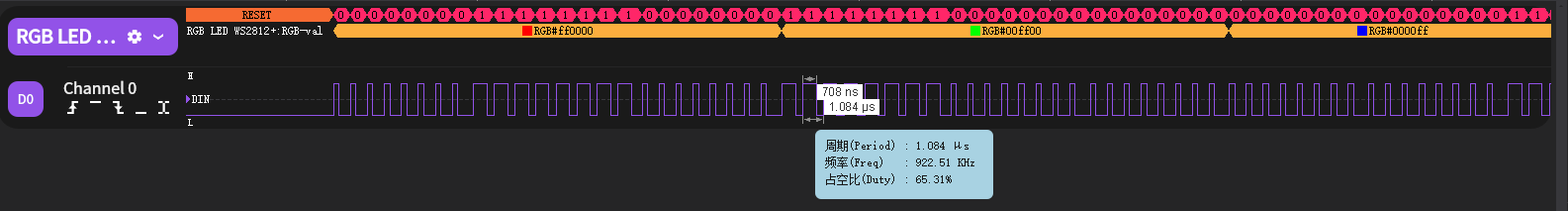
}

发送完毕后，总线需要保持大于50us的低电平，复位WS2812使配置生效。最后设置3个小灯颜色依次为红、绿、蓝，调整颜色数值也可以改变颜色亮度。

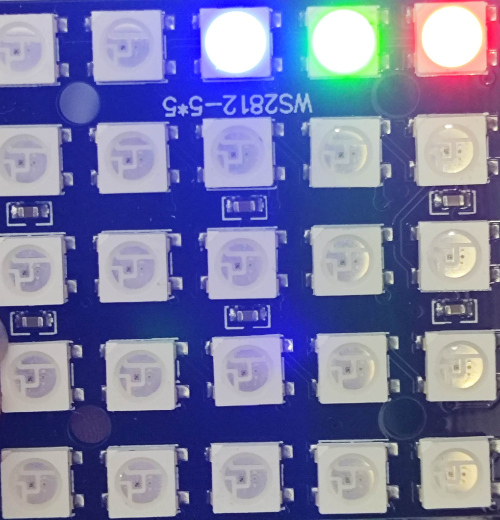
uint8\_t test[9] = {0x00,0xFF,0x00,0xff,0x00,0x00,0x00,0x00,0xff};

WS\_SendOnePix(test,3);

使用逻辑分析仪抓取波形如图所示：



显示效果如图所示：



### 3.2 PWM+DMA方式

这种方式使用PWM来输出信号，PWM的频率设置为800K就可以实现1.25us的周期循环，然后改变周期内的占空比来实现0和1的输出。例如输出0时，将占空比设置为1/3，即高电平持续416ns，低电平持续833ns。输出1时则，将占空比设置为2/3，高电平持续833ns，低电平持续416ns，符合WS2812时序要求。需要W2812复位时，就将占空比设置为0。

关于周期间占空比的切换方式。如果使用定时器中断来切换，就会每1.25us进入一次中断，十分占用系统资源。因此可使用DMA的方式调整占空比。

占空比由比较寄存器CCR的值决定，在CH32V307VCT6中，可以通过配置实现定时器每产生一次溢出事件，就请求DMA搬运一次内存中的数组数据到比较寄存器CCR中，这样PWM波在每次周期之后，就可以自动改变占空比，无需CPU干涉。程序中只需要完成三件事，1.配置好DMA。2.准备好要搬运的数据。3.开关DMA。

程序如下：

**WS2812.h**

#ifndef \_\_WS2812\_H

#define \_\_WS2812\_H

#include "debug.h"

#define PIXEL\_NUM  3                   //表示灯的数量

#define GRB  24                        //24bit

#define WS2812\_ONE            59       //输出"1"的占空比

#define WS2812\_ZERO           29       //输出"0"的占空比

#define WS2812\_GPIO\_PORT       GPIOA

#define WS2812\_GPIO\_CLK        RCC\_APB2Periph\_GPIOA

#define WS2812\_GPIO\_PIN        GPIO\_Pin\_15

#define WS2812\_TIM             TIM2

#define WS2812\_TIM\_CLK         RCC\_APB1Periph\_TIM2

#define WS2812\_DMA             DMA1\_Channel5     //这里使用TIM2\_CH1,选择对应的DMA通道

#define WS2812\_DMA\_FLAG        DMA1\_FLAG\_TC5     //对应通道

void WS2812\_Init();

void WS2812\_Show();

void WS2812\_CloseAll();

void WS2812\_SetPixelRGB(uint32\_t \*grbHex);

#endif

**WS2812.c**

#include "ws2812.h"    //包含ws2812头文件

//====================内部函数声明=======================================

uint16\_t WS2812Buffer[PIXEL\_NUM][GRB]={0};

void WS2812\_Reset(void);

void WS2812\_Init()

{

    //GPIO初始化

    GPIO\_InitTypeDef GPIO\_InitStructure = {0};

    RCC\_APB2PeriphClockCmd(WS2812\_GPIO\_CLK, ENABLE);

    RCC\_APB1PeriphClockCmd(WS2812\_TIM\_CLK,ENABLE);

    //重映射到PA15，如不需要可去掉

    RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO, ENABLE);

    GPIO\_PinRemapConfig(GPIO\_FullRemap\_TIM2, ENABLE);

    GPIO\_InitStructure.GPIO\_Pin = WS2812\_GPIO\_PIN;

    GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_PP;

    GPIO\_Init(WS2812\_GPIO\_PORT, &GPIO\_InitStructure);

    //GPIO\_ResetBits(WS2812\_GPIO\_PORT, WS2812\_GPIO\_PIN);

    //PWM初始化

    TIM\_TimeBaseInitTypeDef TIM\_TimeBaseInitStructure;

    TIM\_OCInitTypeDef TIM\_OCInitStructure;

    TIM\_TimeBaseInitStructure.TIM\_Period = 90-1;//1.25us

    TIM\_TimeBaseInitStructure.TIM\_Prescaler = 0;//不分频

    TIM\_TimeBaseInitStructure.TIM\_ClockDivision = TIM\_CKD\_DIV1;

    TIM\_TimeBaseInitStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

    TIM\_TimeBaseInitStructure.TIM\_RepetitionCounter = 0;//定时结束后立马触发事件

    TIM\_TimeBaseInit( WS2812\_TIM, &TIM\_TimeBaseInitStructure);

    TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_PWM1;

    TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable;

    TIM\_OCInitStructure.TIM\_Pulse = 0;

    TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High;

    TIM\_OC1Init(WS2812\_TIM, &TIM\_OCInitStructure);

    TIM\_OC1PreloadConfig( WS2812\_TIM, TIM\_OCPreload\_Enable );

    TIM\_ARRPreloadConfig( WS2812\_TIM, ENABLE );

    TIM\_CtrlPWMOutputs(TIM2, ENABLE);

    //DMA配置

    DMA\_InitTypeDef DMA\_InitStructure = {0};

    RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_DMA1, ENABLE);

    DMA\_DeInit(WS2812\_DMA);

    DMA\_InitStructure.DMA\_PeripheralBaseAddr = (u32)&WS2812\_TIM->CH1CVR;//外设基地址

    DMA\_InitStructure.DMA\_MemoryBaseAddr = (uint32\_t)WS2812Buffer ;//传送数据内存基地址

    DMA\_InitStructure.DMA\_DIR = DMA\_DIR\_PeripheralDST;//数据传输方向

    DMA\_InitStructure.DMA\_BufferSize = PIXEL\_NUM \* GRB;//一次传输数据量

    DMA\_InitStructure.DMA\_PeripheralInc = DMA\_PeripheralInc\_Disable;//传输数据时外设地址不递增

    DMA\_InitStructure.DMA\_MemoryInc = DMA\_MemoryInc\_Enable;//传输数据时内存数据递增

    DMA\_InitStructure.DMA\_PeripheralDataSize = DMA\_PeripheralDataSize\_HalfWord;//外设数据长度，半字传输（1次16bit)

    DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord;//内存数据长度，同样半字传输

    DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Normal;//不使用自动重装

    DMA\_InitStructure.DMA\_Priority = DMA\_Priority\_VeryHigh;

    DMA\_InitStructure.DMA\_M2M = DMA\_M2M\_Disable;

    DMA\_Init(WS2812\_DMA, &DMA\_InitStructure);

    TIM\_DMACmd(WS2812\_TIM, TIM\_DMA\_CC1, ENABLE);

    //TIM\_CtrlPWMOutputs(WS2812\_TIM, ENABLE);// 高级定时器需要将这个使能才能输出PWM

    DMA\_Cmd(WS2812\_DMA, DISABLE);

    TIM\_Cmd( WS2812\_TIM, DISABLE );

    WS2812\_CloseAll();

    Delay\_Ms(100);

}

void WS2812\_Show(void)

{

  DMA\_SetCurrDataCounter(WS2812\_DMA, PIXEL\_NUM \* GRB);

  TIM\_Cmd(WS2812\_TIM, ENABLE);

  DMA\_Cmd(WS2812\_DMA, ENABLE);

  while(DMA\_GetFlagStatus(WS2812\_DMA\_FLAG) != SET);

  Delay\_Us(1);

  WS2812\_Reset();

  TIM\_Cmd(WS2812\_TIM, DISABLE);

  DMA\_Cmd(WS2812\_DMA, DISABLE);

  DMA\_ClearFlag(WS2812\_DMA\_FLAG);

}

void WS2812\_Reset(void)

{

    uint8\_t i;

    uint8\_t j;

    for(i = 0; i < 3; i++)

    {

      for(j = 0; j < 24; j++)

      {

        WS2812Buffer[i][j] = 0;

      }

    }

    DMA\_SetCurrDataCounter(WS2812\_DMA, 3 \* GRB);

    while(DMA\_GetFlagStatus(WS2812\_DMA\_FLAG) != SET);

}

void WS2812\_CloseAll(void)

{

  uint8\_t i;

  uint8\_t j;

  for(i = 0; i < PIXEL\_NUM; i++)

  {

    for(j = 0; j < 24; j++)

    {

      WS2812Buffer[i][j] = WS2812\_ZERO;

    }

  }

  WS2812\_Show();

}

void WS2812\_SetPixelRGB(uint32\_t \*grbHex)

{

    uint8\_t i;

    uint8\_t j;

    for(i=0;i<PIXEL\_NUM;i++)

    {

          for(j = 0;j<24;j++)

          {

              WS2812Buffer[i][j] = ((grbHex[i] << j) & 0x800000) ? WS2812\_ONE : WS2812\_ZERO;

          }

    }

    WS2812\_Show();

}

程序解析：

驱动初始化过程分为三部分，1.GPIO初始化；2.PWM初始化；3.DMA初始化

GPIO配置如下

本程序使用的引脚是PA15，根据CH32V307的手册（如下图）



PA15的默认复用功能没有定时器，但重映射后可以使用TIM2\_CH1，TIM2支持输出PWM（如果使用的引

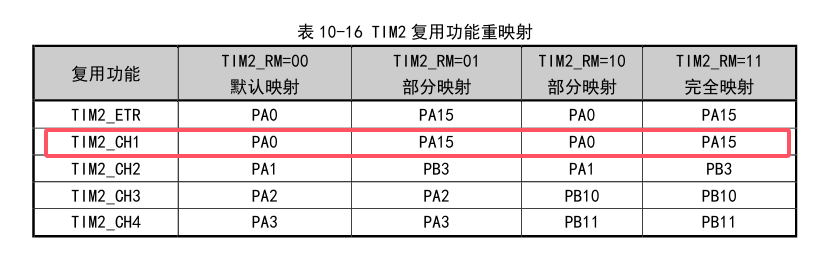
PWM,就不需要重映射），重映射操作如下

    //重映射到PA15，如不需要可去掉

    RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO, ENABLE);

    GPIO\_PinRemapConfig(GPIO\_FullRemap\_TIM2, ENABLE);

重映射前需要使能AFIO时钟。重映射功能有两种模式，分别是部分映射和完全映射，其中TIM2的部分映射又分为两种，具体需要参考手册。



这里我们配置为完全映射，配置完后PA15就可以使用TIM2通道1的PWM输出。

GPIO模式设置为复用推挽输出

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_PP;

PWM的配置如下

首先定时器预分频器设置为0，也就是不分频，自动重载寄存器（ARR）设定为90。

   TIM\_TimeBaseInitStructure.TIM\_Period = 90-1;//1.25us

   TIM\_TimeBaseInitStructure.TIM\_Prescaler = 0;//不分频

这样频率为72Mhz，每次计数花费1/72us，计90个数为一个周期，花费1.25us，差不多符合WS2812通讯周期。

配置为向上计数

TIM\_TimeBaseInitStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

配置PWM模式为模式1

TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_PWM1;

PWM模式1在向上计数时，一旦TIMx\_CNT<TIMx\_CCR1时通道1为有效电平，否则为  
无效电平；在向下计数时，一旦TIMx\_CNT>TIMx\_CCR1时通道1为无效电平(OC1REF=0)，否则为有效电平(OC1REF=1)。TIMx\_CNT是定时器的计数寄存器，TIMx\_CCR1是定时器的捕获/比较寄存器，影响占空比。有效电平的定义则由下面这句配置。

TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High;

这句配置了有效电平是高电平。这样当计数值小于比较值时，输出高电平，反之输出低电平。

接着初始化时配置占空比为0，持续输出低电平。

TIM\_OCInitStructure.TIM\_Pulse = 0;

这句实际上设定了TIM2的比较捕获寄存器通道1的值（CCR1/ TIM2\_CH1CVR）

DMA配置如下

设定外设基地址为TIM2的比较捕获寄存器通道1（TIM2\_CH1CVR）

DMA\_InitStructure.DMA\_PeripheralBaseAddr = (u32)&WS2812\_TIM->CH1CVR;//外设基地址

内存基地为我们定义的数组，里面保存小灯的颜色数据。

DMA\_InitStructure.DMA\_MemoryBaseAddr = (uint32\_t)WS2812Buffer ;//传送数据内存基地址

数据传输方向为内存到外设

DMA\_InitStructure.DMA\_DIR = DMA\_DIR\_PeripheralDST;//数据传输方向

每次传输数据的量就是数组的长度

DMA\_InitStructure.DMA\_BufferSize = PIXEL\_NUM \* GRB;//一次传输数据量

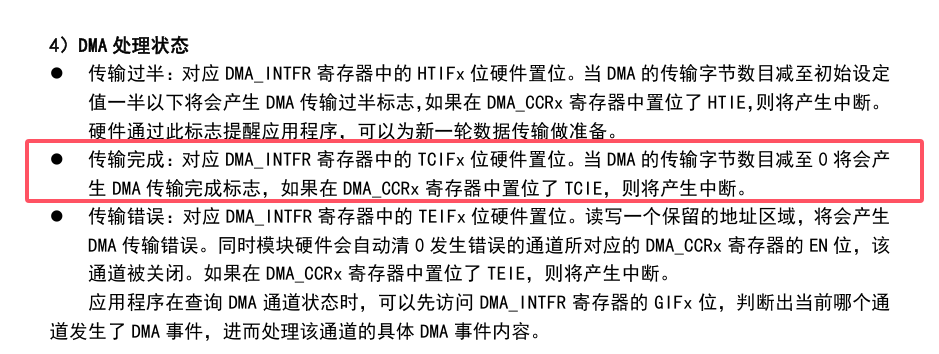
使能TIM\_DMA\_CC1更新中断

TIM\_DMACmd(WS2812\_TIM, TIM\_DMA\_CC1, ENABLE);

这里需要注意的是，依据用户手册中DMA1各通道外设映射表（如下）



我们选择中断源是TIM\_DMA\_CC1，因此对应的是DMA1通道5，.DMA传输完成后会产生中断完成标志，根据用户手册描述



可以使用DMA\_GetFlagStatus(WS2812\_DMA\_FLAG)函数获取中断标志位，这里的WS2812\_DMA\_FLAG根据使用的DMA通道来设置，比如我们使用了DMA1通道5，那么就使用DMA1\_FLAG\_TC5。

初始化完成后，我们还需要发送函数，原理与延时方式的发送函数类似，将颜色数据放进1个组数中，从最高位开始依次判断是“1”还是“0”。与延时方式不同的是，“1”的话我们将占空比调整为2/3%，“0”我们将占空比调整为1/3。

void WS2812\_SetPixelRGB(uint32\_t \*grbHex)

{

    uint8\_t i;

    uint8\_t j;

    for(i=0;i<PIXEL\_NUM;i++)

    {

          for(j = 0;j<24;j++)

          {

              WS2812Buffer[i][j] = ((grbHex[i] << j) & 0x800000) ? WS2812\_ONE : WS2812\_ZERO;

          }

    }

    WS2812\_Show();

}

void WS2812\_Show(void)

{

  DMA\_SetCurrDataCounter(WS2812\_DMA, PIXEL\_NUM \* GRB);

  TIM\_Cmd(WS2812\_TIM, ENABLE);

  DMA\_Cmd(WS2812\_DMA, ENABLE);

  while(DMA\_GetFlagStatus(WS2812\_DMA\_FLAG) != SET);

  Delay\_Us(1);

  WS2812\_Reset();

  TIM\_Cmd(WS2812\_TIM, DISABLE);

  DMA\_Cmd(WS2812\_DMA, DISABLE);

  DMA\_ClearFlag(WS2812\_DMA\_FLAG);

}

void WS2812\_Reset(void)

{

    uint8\_t i;

    uint8\_t j;

    for(i = 0; i < 3; i++)

    {

      for(j = 0; j < 24; j++)

      {

        WS2812Buffer[i][j] = 0;

      }

    }

    DMA\_SetCurrDataCounter(WS2812\_DMA, 3 \* GRB);

    while(DMA\_GetFlagStatus(WS2812\_DMA\_FLAG) != SET);

}

这里复位方式是将占空比调整为0，持续3x24x1.25us=90us的电平信号。需要注意的是使用TIM\_SetCompare1( )函数无法改变占空比，需要使用DMA\_SetCurrDataCounter()函数。

uint32\_t grbtest[PIXEL\_NUM] = {0x00FF00,0xffff00,0x0000ff};

WS2812\_SetPixelRGB(grbtest);

逻辑分析仪显示如下图所示：

