1 DAC 简介

1.1 DAC 模块简介

DAC 为数字/模拟转换模块，故名思议，它的作用就是把输入的数字编码，转换成对应的模拟电压输出，它的功能与 ADC 相反。在常见的数字信号系统中，大部分传感器信号被化成电压信号，而 ADC 把电压模拟信号转换成易于计算机存储、处理的数字编码，由计算机处理完成后，再由 DAC 输出电压模拟信号，该电压模拟信号常常用来驱动某些执行器件，使人类易于感知。如音频信号的采集及还原就是这样一个过程。

1.2 STM32 的 DAC 模块

野火 STM32-ISO 开发板使用的 STM32F103ZET6 主控芯片，具有片上 DAC 模块，它的分辨率可配置为 8 位或 12 位的数字输入信号，它有两个 DAC 通道，分别是：PA4-通道 1，PA5-通道 2。在双 DAC 模式下，它的两个通道互不影响，每个通道都可以使用 DMA 功能，都具有出错检测能力，可外部触发。

正弦波信号是频率最为单一的一种信号，也是模拟信号的代表。本章将把 STM32 的 DAC 跟 DMA 控制器配合使用，产生连续的正弦波。

1.3 DAC 工作过程分析

STM32 的 DAC 模块框图见图 1-1，整个 DAC 模块围绕图片下方的“数字至模拟转换器 x”展开，它的左边分别是参考电源的引脚：V_{DDA}、V_{SSA}及V_{ref+}，其中 STM32 的 DAC 规定了它的参考电压V_{ref+}输入范围为 2.4——3.3V。“数字至模拟转换器 x”的输入为 DAC 的数据寄存器“DORx”的数字编码，经过它转换得的模拟信号由图中右侧的“DAC_OUTx”输出。而数据寄存器“DORx”又受“控制逻辑”支配，它可以控制数据寄存器加入一些伪噪声信号或配置产生三角波信号。图中的左上角为 DAC 的触发源，DAC 根据触发源的信号来进行 DAC 转换，其作用就相当于 DAC 转换器的开关，它可以配置的触发源为外部中断源触发、定时器触发或软件控制触发。如本章实验中需要控制正弦波的频率，就需要定时器定时触发 DAC 进行数据转换。

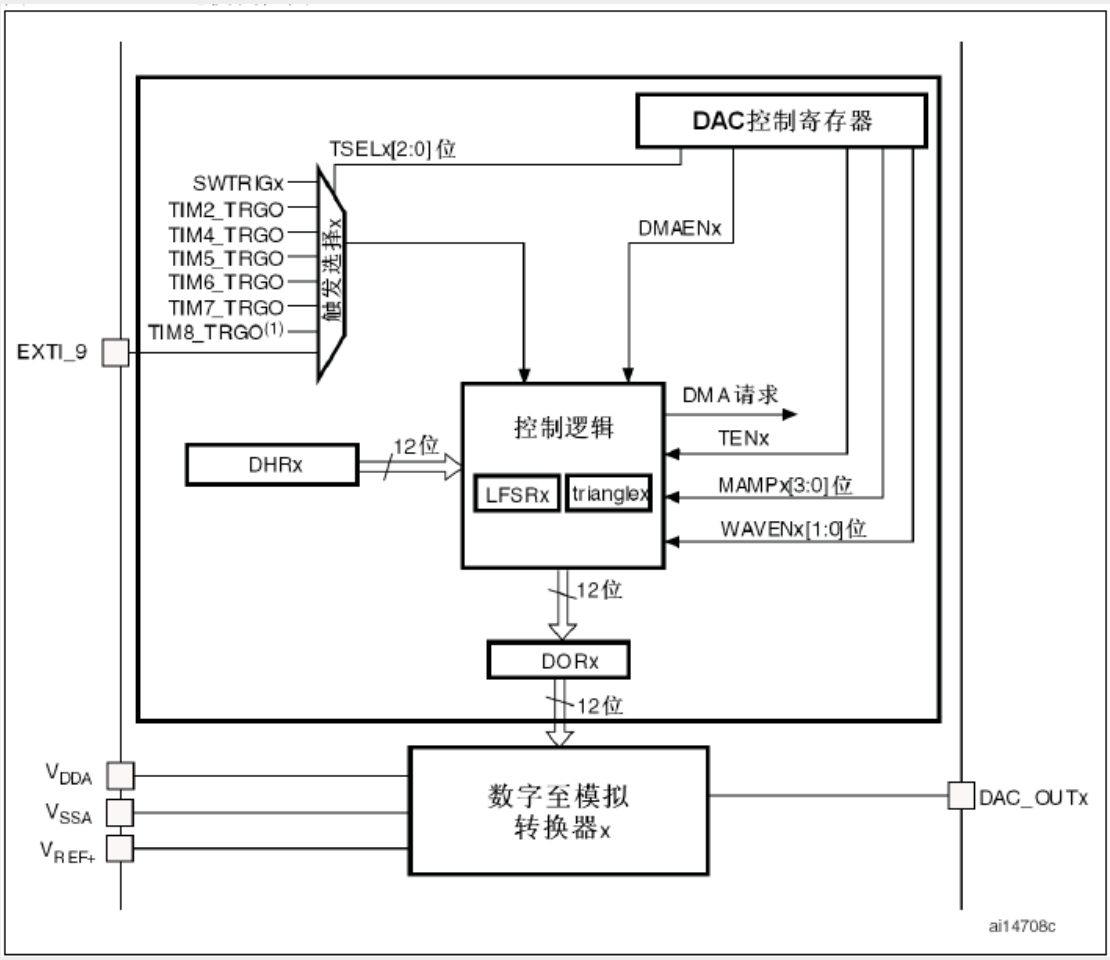


图 1-1 STM32 DAC 框图

1.4 STM32 的 DAC 性能指标

❑ **分辨率**

STM32 的 DAC 分辨率为 12 位。不能直接输出负电压，即其最小量化单位 LSB = $V_{ref+}/2^{12}$。

❑ **偏移量误差**

偏移量误差是指输入数字量为零时，输出模拟量对零的偏移值，STM32 的这个偏移量误差是±3LSB。

❑ **转换时间：**

DAC 的转换时间是指输入的数字信号转换为输出的模拟信号所需要的时间。由于不同的情况下转换时间不定，我们以 DAC 的设置时间(输入编码从最小值转为最大值的时间)来衡量转换时间，STM32 的 DAC 设置时间为 4us。

❑ **输出电压**

STM32 的输出电压只能为正值，它的电压输出范围为 0.2 至$V_{ref+} - 0.2$ V。

❑ **线性度**

线性度指 DAC 实际转换的曲线跟理论要转换的曲线偏差的值。STM32 的 DAC 非线性度属失真为 0.5LSB。

1.5 DAC 产生正弦波原理

首先我们要了解 MCU(微型计算机)是如何产生正弦波的。MCU 是无法生成完美的正弦波的，只能对正弦波上按一定时间间隔对曲线上的点进行取样，再用数字/模拟转换模块（DAC）将这些点输出，得到尽可能逼近正弦波的图形，于是我们就应用了正弦波表。有读者可能会问为何不用正弦函数 y=sin(x) 来生成实时的正弦函数的点坐标，一个原因是这个函数是浮点数类型，运算量较大；还有另一个原因，就是 DAC 输出一般都为正值，而众所周知，y=sin(x) 函数结果有一半为负值，若要用 DAC 输出，还需要一定的数学处理方法。综上所述，我们这里采用正弦波表更为简单直接。

经过测试，野火在这里采用 32 个点为一个周期的正弦波表，从图 1-1matlab 环境下的 正弦波可看出，波形已经相对平滑了。

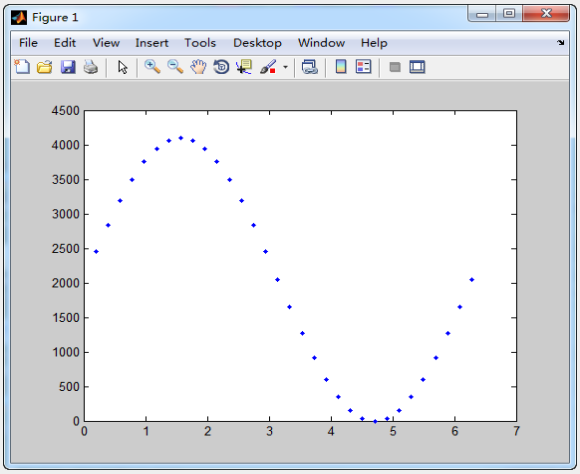


图 1- 1matlab 环境下的 正弦波

2 DAC 正弦波实例分析

接下来我们通过实例来分析 DAC 正弦波，通过学习这个程序，读者会进一步熟悉 STM32 的 DAC 和 DMA 技术。

2.1 实验描述及工程文件清单

实验描述	使用 STM32 的双 DAC 功能实现正弦波的输出
硬件连接	PA4 与示波器 CH1 探针相连， PA5 与示波器 CH2 探针相连
用到的库文件	*startup/start_stm32f10x_hd.c* *CMSIS/core_cm3.c* *CMSIS/system_stm32f10x.c* *FWlib/stm32f10x_gpio.c* *FWlib/stm32f10x_rcc.c* *FWlib/stm32f10x_tim.c* *FWlib/stm32f10x_dma.c* *FWlib/stm32f10x_dac.c*
用户编写的文件	*USER/main.c* *USER/stm32f10x_it.c*

	USER/bsp_dac.c

表 2- 1 工程文件清单

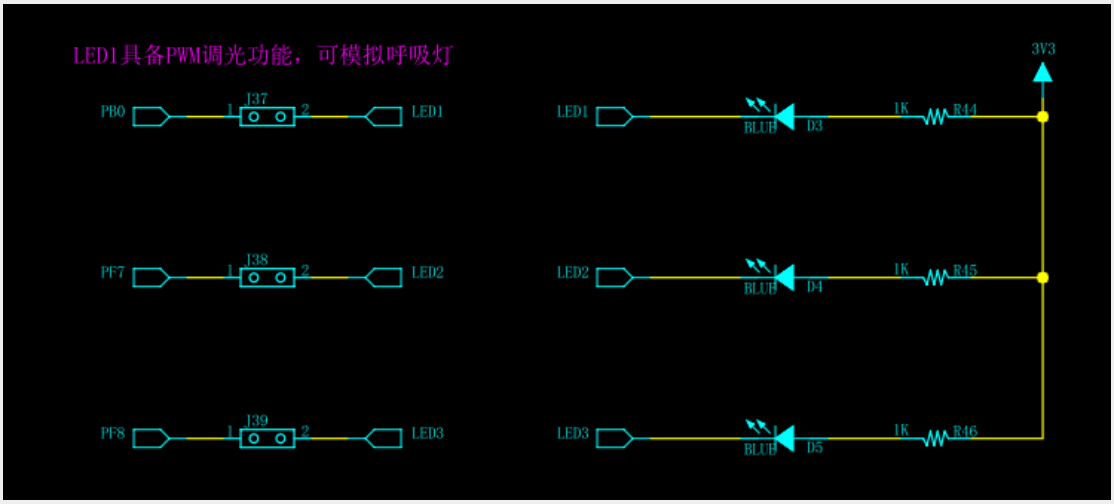


图 2- 1 正弦波用到的引脚图

2.1.1 配置工程环境

本 DAC 正弦波实验中我们用到了 GPIO、RCC、TIM、DMA、DAC 外设，还使用了中断，所以我们先要把以下库文件添加到工程：stm32f10x_gpio.c、stm32f10x_rcc.c、stm32f10x_tim.c、stm32f10x_dma、stm32f10x_dac。

新建 bsp_dac.c 及 bsp_dac.h 文件，并在 stm32f10x_conf.h 中把使用到的 ST 库的头文件注释去掉。见代码清单 2- 1 配置实验环境。

```
/**
  ******************************************************************************
  * @file    Project/STM32F10x StdPeriph Template/stm32f10x_conf.h
  * @author  MCD Application Team
  * @version V3.5.0
  * @date    08-April-2011
  * @brief   Library configuration file.
  ******************************************************************************
  **/
#ifndef __STM32F10x_CONF_H
#define __STM32F10x_CONF_H

#include "stm32f10x_dac.h"
#include "stm32f10x_dma.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"
```

代码清单 2- 1 配置实验环境

2.2 main 文件

本工程的 main 函数十分简单，仅调用了一个初始化 DAC 的函数 DAC_Mode_Init()，见代码清单 2-2 main 函数。

```
1 #include "stm32f10x.h"
2 #include "bsp_dac.h"
3
4 /**
5   * @brief  主函数
6   * @param  无
7   * @retval 无
8  **/
9 int main(void)
10 {
11     /*初始化 DAC，开始 DAC 转换*/
12     DAC_Mode_Init();
13
14     while (1);
15 }
```

代码清单 2-2 main 函数

2.3 配置 DAC

DAC 初始化函数按顺序初始化各个模块：调用 DAC_Config()使能 DAC 时钟和初始化 DAC 用到的 GPIO；调用 DAC_TIM_Config()配置 DAC 用到的定时器 TIM；调用 DAC_DMA_Config()配置 DAC 的 DMA 通道，最后通过循环体将 DMA 内存数据进行右对齐。详情见代码清单 2-3 DAC_Mode_Init()函数定义：

```
1 /**
2   * @brief  DAC 初始化函数
3   * @param  无
4   * @retval 无
5   */
6 void DAC_Mode_Init(void)
7 {
8     uint32_t Idx = 0;
9
10     DAC_Config();
11     DAC_TIM_Config();
12     DAC_DMA_Config();
13
14     /* 填充正弦波形数据，双通道右对齐*/
15     for (Idx = 0; Idx < 32; Idx++) {
16         DualSine12bit[Idx] = (Sine12bit[Idx] << 16) + (Sine12bit[Idx]);
17     }
18 }
```

代码清单 2-3 DAC_Mode_Init()函数定义

2.4 生成正弦曲线数据

要实现 DAC 正弦波，必须先将正弦曲线上按相同的时间间隔取点，将数据存储后利用 DMA 通道查询，并通过 DAC 模块输出。这个正弦波形数据表的数据存储在程序中的数组 Sine12bit 中，见代码清单 2-4 正弦波波形数据表。

```
/* 波形数据 ----------------------------------------------------------*/
const uint16_t Sine12bit[32] = {
    2448,2832,3186,3496,3751,3940,4057,4095,4057,3940,
    3751,3496,3186,2832,2448,2048,1648,1264,910,600,345,
    156,39,0,39,156,345,600,910,1264,1648,2048};
```

代码清单 2-4 正弦波波形数据表

读者看到这里是不是感觉有点熟悉呢？是的，前面呼吸灯我们也是使用类似方法。只是前者的数据用来记录时间，后者数据用来描点而已。

把这个表的数据画成图，图 2-2 正弦波表变化曲线。

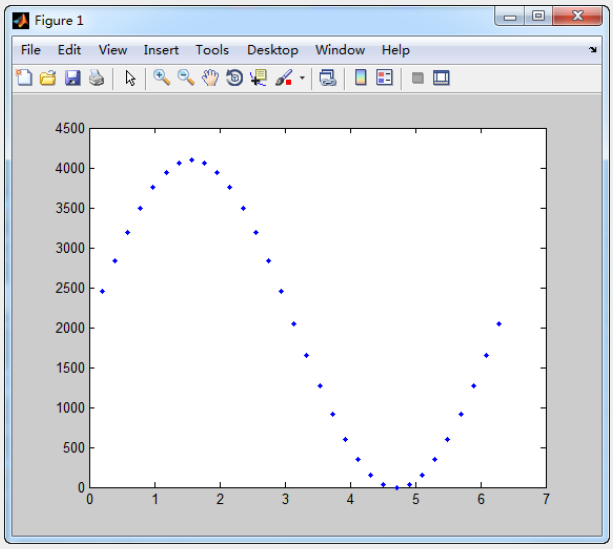


图 2-2 正弦波表变化曲线

这个表有 32 个数字，从图中可以看到这些数字组成的图形跟数学上正弦曲线图片一致，正好是呼吸灯的一个控制周期。数字的大小范围是 0~4095，即把正弦曲线的变化范围设置为 4095 个变化范围。

当程序运行时，每隔一个相同的周期，就会通过 DMA 通道访问早已记录 DualSine12bit 内存数据地址，再用 DAC（数字/模拟转换模块）输出，如此周而复始，达到实现输出正弦波的效果。

这个表的数据是野火使用 matlab 软件生成的，该 matlab 代码见代码清单 2-5 用于生成正弦波曲线的 matlab 代码，可供读者参考。该代码运行后会生成一个“sinWave.m”的文件，用户把该文件中的数据复制到工程中的数组中即可。

```
%用于产生正弦数据表，输出到文件 dac_sinWave.c 文件中，复制到 c 语言数组即可

n = 2*pi/32 : 2*pi/32 : 2*pi        %分成 32 等份

a = sin(n)+1;                       %求取 sin 函数值并向上平移一个单位，消除负数值
a = a * 3.3/2;                      %调整幅值，使范围限制为 0~3.3
r = a* (2.^12) /3.3                 %求取 dac 数值，12 位 dac LSB = 3.3/2.^12
r = uint16(r);                       %把 double 型数据转化成 16 位整型数据

for i = 1:32
if r(i) > 4095                        %限制数据最大不超过 4095
    r(i) = 4095
end
end

dlmwrite('dac sinWave.c',r);        %把数据写入到文件,方便添加到 stm32 工程中
plot(n,r,'.') %把这些点画出来
```

代码清单 2-5 用于生成正弦波曲线的 matlab 代码

2.5 初始化 GPIO，DAC 通道

在 DAC_Config()函数中，我们完成了 DAC 通道的 GPIO 的初始化和 DAC 通道的配置。1.DAC 使用 PA4，PA5 为输出通道，DAC1 通道由 PA4 输出，DAC2 通道由 PA5 输出，工作模式为模拟输出；2.DAC 选择 TIME2 为外部触发通道，不用波形发生器，不使用 DAC 输出缓冲。见 代码清单 2-6 初始化 DAC

```
  * @brief  使能 DAC 的时钟，初始化 GPIO
  * @param  无
  * @retval 无
  */
static void DAC_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    DAC_InitTypeDef  DAC_InitStructure;

    /* 使能 GPIOA 时钟 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    /* 使能 DAC 时钟 */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

    /* DAC 的 GPIO 配置，模拟输入 */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_4 | GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);



    /* 配置 DAC 通道 1 */
    DAC_InitStructure.DAC_Trigger = DAC_Trigger_T2_TRGO;          //使用 TIM2 作为触发源
    DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None; //不使用波形发生器
    DAC_InitStructure.DAC_OutputBuffer= DAC_OutputBuffer_Disable;//不使用 DAC 输出缓冲
    DAC_Init(DAC_Channel_1, &DAC_InitStructure);
```

```
29
30      /* 配置 DAC 通道 2 */
31      DAC_Init(DAC_Channel_2, &DAC_InitStructure);
32
33      /* 使能通道 1 由 PA4 输出 */
34      DAC_Cmd(DAC_Channel_1, ENABLE);
35      /* 使能通道 2 由 PA5 输出 */
36      DAC_Cmd(DAC_Channel_2, ENABLE);
37
38      /* 使能 DAC 的 DMA 请求 */
39      DAC_DMACmd(DAC_Channel_2, ENABLE);
40  }
```

代码清单 2-6 初始化 DAC

代码第 24~39 行是 DAC 的配置，

1) DAC_Trigger

选择触发源。对 DAC 事件的发生条件进行设置，这里的值可以为 None，TIM2，TIM4-8，EXTI LINE9 还可以是软件触发 Software。我们这里准备使用 TIM2 产生一个定时周期为 20 的一个定时器来触发 DAC 事件。

2) DAC_WaveGeneration

波形发生器选择。可选择的值有 None，Noise，Triangle。此处的波形我们已用 matlab 生成，所以设置为 None 即可。

3) DAC_OutputBuffer

DAC 缓冲设置。这里可选为 Disable 和 Enable。此处不需要要 DAC 输出缓冲，设置为 Disable。

具体 TIM 配置分频，周期，模式的内容，下节会给出。

2.6 配置定时器模式

在 DAC_TIM_Config()函数中，我们配置 DAC 的触发方式 TIM2 的周期，分频以及记数方式进行了配置。因为这些参数我们在前面呼吸灯章节我们已经有了详细的讲解，这里就不多累赘，有疑问的同学请参考呼吸灯章节的定时器配置，详细代码见代码清单 2-7 TIM3_Mode_Config 函数定义。

```
1  /**
2    * @brief  配置 TIM
3    * @param  无
4    * @retval 无
5    */
6  static void DAC_TIM_Config(void)
7  {
8
```

```
    TIM_TimeBaseInitTypeDef    TIM_TimeBaseStructure;

    /* 使能 TIM2 时钟，TIM2CLK 为 72M */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    /* TIM2 基本定时器配置 */
// TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
    TIM_TimeBaseStructure.TIM_Period = 19;                          //定时周期 20
    TIM_TimeBaseStructure.TIM_Prescaler = 0x0;                      //预分频，不分频
72M / (0+1) = 72M
    TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;                  //时钟分频系数
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;     //向上计数模式
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

    /* 配置 TIM2 触发源 */
    TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);

    /* 使能 TIM2 */
    TIM_Cmd(TIM2, ENABLE);

}
```

代码清单 2-7 TIM3_Mode_Config 函数定义

因为 DAC 使用 TIM2 当触发源，所以这里将对 TIM2 进行配置。配置 TIM2 定时周期为 20，不分频。即 TIM2 每个 20*（1/72M）秒就会触发一次 DAC 事件。这里的 TIM2 的定时周期以及分频会在后面的计算正弦波频率具体应用到。

2.7 DMA 通道配置

本函数为 DMA 的通道的配置，目的是利用 DMA 通道将 matlab 软件所生成的正弦波表的值，通过 DAC 方式在 PA4，PA5 两个 IO 口以电压的形式输出。跟 TIM2 的配置一样，由于前面 DMA 通道章节已进行了详细的讲解，若还有不明白的同学可以参照 DMA 通道章节进行学习，代码清单 2-8 DAC 所用到 DMA 配置。

```
/**
  * @brief  配置 DMA
  * @param  无
  * @retval 无
  */
static void DAC_DMA_Config(void)
{
 DMA_InitTypeDef  DMA_InitStructure;

    /* 使能 DMA2 时钟 */
 RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA2, ENABLE);

    /* 配置 DMA2 */
 DMA_InitStructure.DMA_PeripheralBaseAddr = DAC_DHR12RD_Address;    //外设数据地址
 DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&DualSine12bit ;    //内存数据地址
DualSine12bit
 DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;          //数据传输方向内存至外设
 DMA_InitStructure.DMA_BufferSize = 32;                      //缓存大小为 32 字节
 DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; //外设数据地址固定
```

```
19  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;          //内存数据地址自增
20  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word; //外设数据以
                                                                          字为单位
21  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;//内存数据以字为单位
22  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;               //循环模式
23  DMA_InitStructure.DMA_Priority = DMA_Priority_High;            //高 DMA 通道优先级
24  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;                        //非内存至内存模式
25
26  DMA_Init(DMA2_Channel4, &DMA_InitStructure);
27
28   /* 使能 DMA2-14 通道 */
29  DMA_Cmd(DMA2_Channel4, ENABLE);
30 }
```

代码清单 2-8 DAC 所用到 DMA 配置

本实验 DMA 通道的内存基地址为 DualSine12bit 的数据地址，正弦波表共 32 个字节，所以缓存 DMA_BufferSize 值为 32，实验中会将 DMA 内存基地址上的数据循环传到 DAC 输出，DMA_Mode 采用 Circular 循环模式。

2.8 正弦波频率计算方式

到这里，正弦波的函数基本已经讲解完毕。下面我们来说一下如何生成正弦波，以及正弦波的频率计算方式。

生成正弦波的流程是：

a） 通过 matlab 软件将生成一个 32 个等时间间距的正弦波表；

b） 将正弦波表数值 DualSine12bit 的内存数据地址传给 DMA 通道；

c） 利用 TIM2 触发 DAC 通道，每次将一个正弦波表上的值通过 DAC 以电压形式输出，当 TIM2 触发 32 次后，即为一个完整的正弦波。

正弦波频率的计算方式：

系统时钟周期为 1/72000000，定时器 TIM2 分频 TIM_Prescaler 为 0+1=1，定时周期 Ttim 为 1/72000000*1；定时周期数 TIM_Period 为 19+1=20，定时器触发周期 Tupdate 为 1/72000000*20；正弦波表有 32 个点，则显示一个正弦波的时间为 Tupdate*32。综上所述正弦波频率为 1/Tupdate*32。

计算公式为 f=1/(Tsysclk*(TIM_Prescaler+1)*(TIM_Period+1)*point)（其中 point 为正弦波表点的数目 32），详见代码清单 2-9 正弦波频率计算。

计算得频率约为 112500Hz，我们通过示波器显示得出频率约为 111.1khz，基本一致。详见图 2- 3。

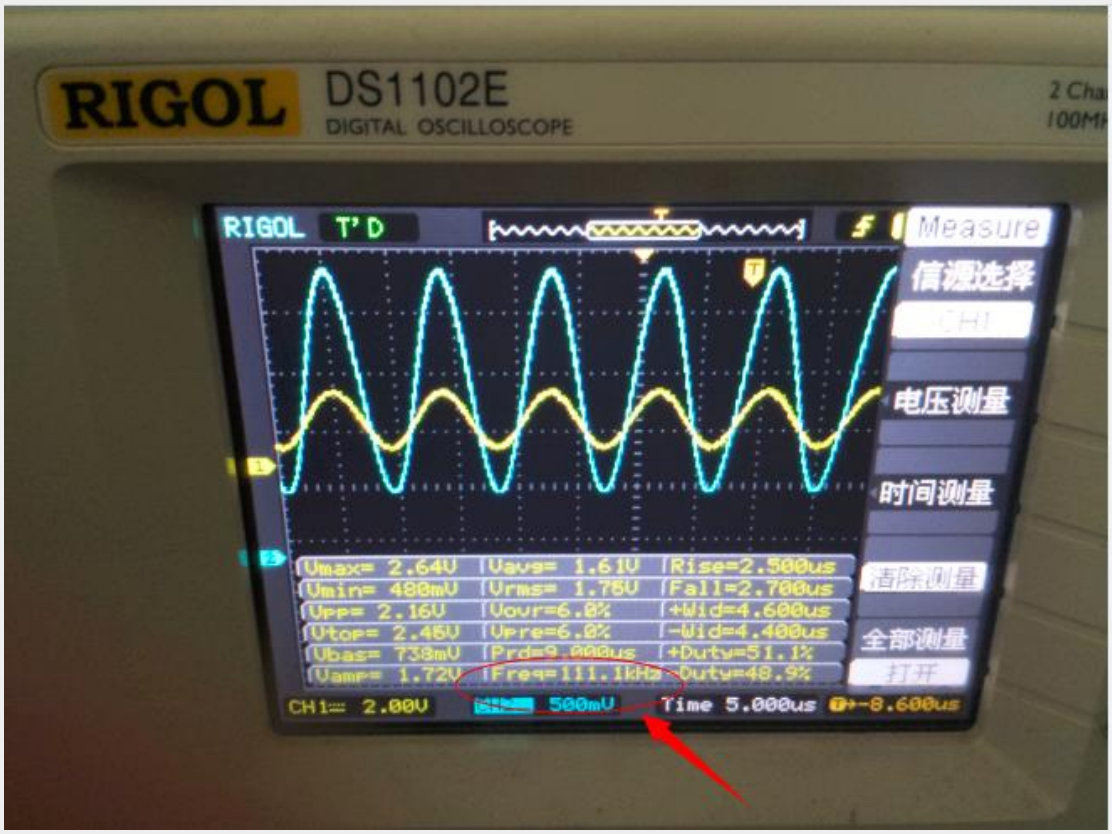


图 2- 3

```
%用于通过设定 period 和 prescaler 求解 DAC 输出的正弦信号频率

TIM Period     = 19;                  %定时周期数
TIM_Prescaler = 0;                    %定时时钟分频
point =32;                            %单周期信号点数
Tsysclk=1/72000000;                    %系统时钟周期
Ttim = Tsysclk * (TIM_Prescaler+1);   %定时时钟周期
Tupdate = Ttim * (TIM Period +1);     %定时器触发周期
Tsin = Tupdate * point;                %一个正弦波周期内的 dac 点数
fsin = 1/Tsin                          %正弦波频率
```

代码清单 2- 9 正弦波频率计算

❑ 疑问

为什么正弦波表是 32 个点，而不取别的值？

野火觉得因为 32 个坐标表示一个正弦波已足矣。并不是规定正弦波表数组数一定要为 32 个。但为了方便计算，一般数组都为 2 的 N 次方，如 16 个，64 个，都可以。读者若有兴趣，可以自行修改，因为程序比较简单，只需⑴修改

Sine12bit 的数组位数和具体数值(可以用 matlab 软件生成，也可网上搜索一下)，⑵DualSine12bit 位数需与 Sine12bit 相同，⑶DMA_BufferSize，DAC_Mode_Init()中的 Idx 的值也需跟 Sine12bit 的位数一致。

3.实验现象

下载程序，拔掉红色区域三个跳线帽（默认是插上的）。如图 3- 1 所示：

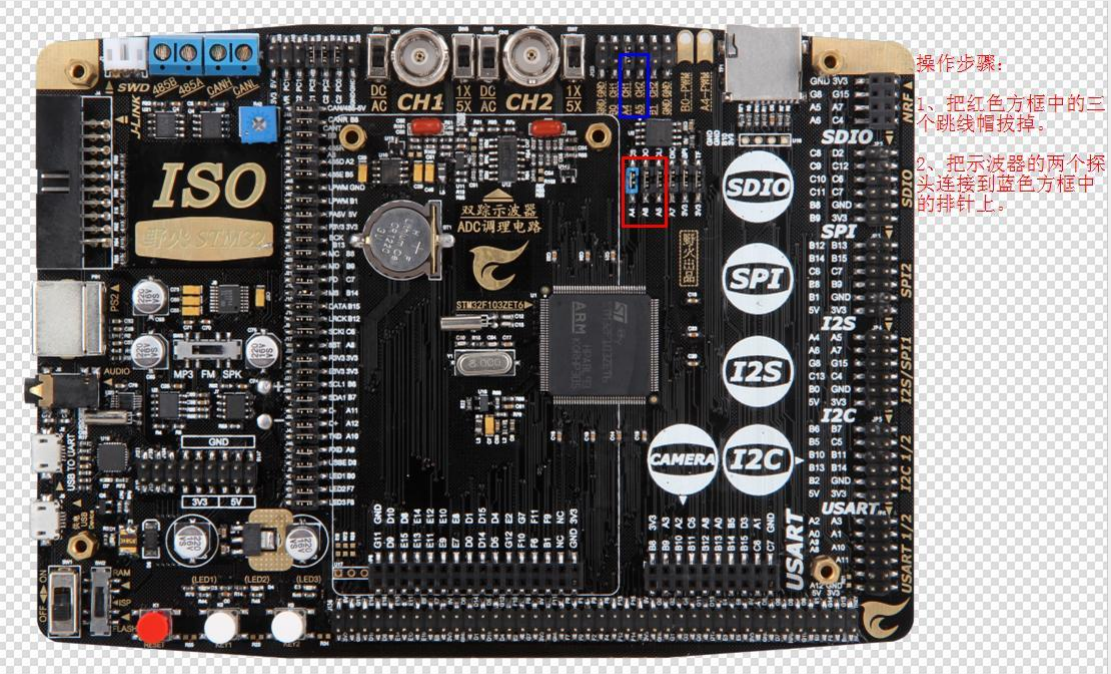


图 3- 1 示波器探头连接 PA4，PA5 引脚

打开示波器，将示波器两个探头连接蓝色区域的排针上（记得接地），调整波形，得到类似图 3- 2 的正弦波：

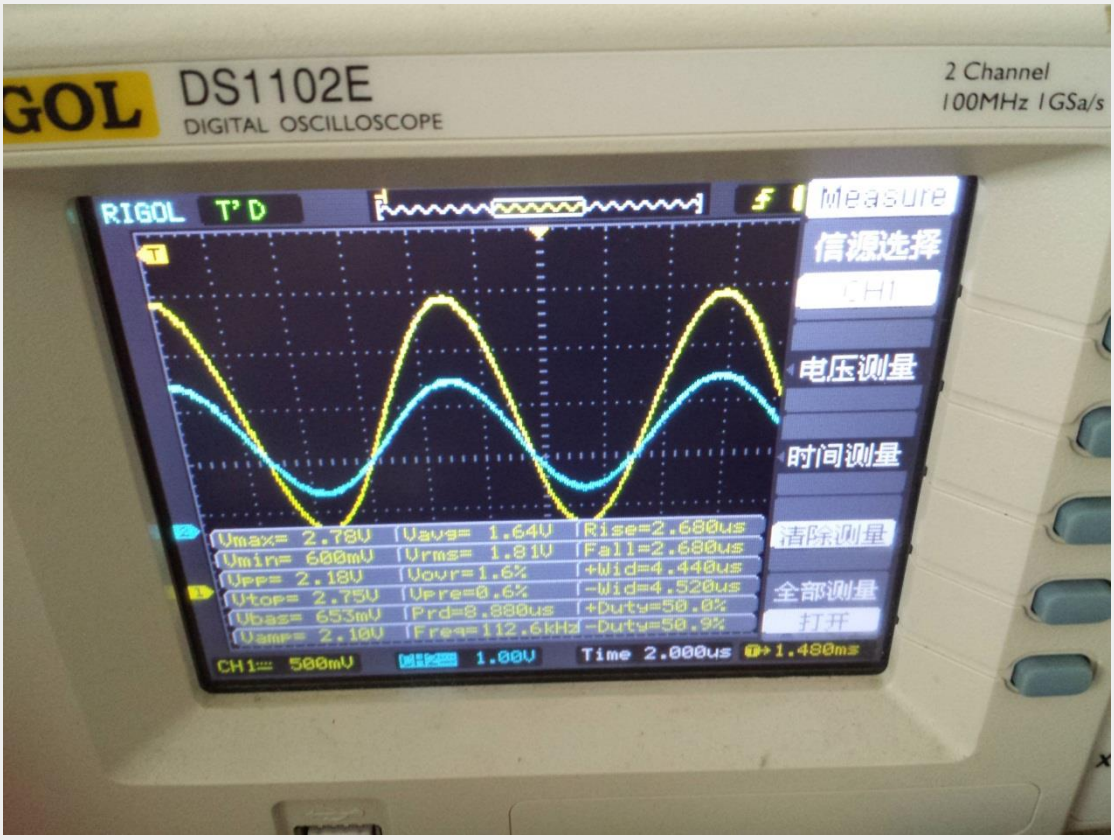

GOL
DS1102E
DIGITAL OSCILLOSCOPE
2 Channel
100MHz 1GSa/s
RIGOL
T'D
Measure
信源选择
CH1
电压测量
时间测量
清除测量
全部测量
打开
Vmax= 2.78V
Vavg= 1.64V
Rise=2.680us
Vmin= 600mV
Vrms= 1.81V
Fall=2.680us
Vpp= 2.18V
Vovr=1.6%
+Wid=4.440us
Vtop= 2.75V
Vpre=0.6%
-Wid=4.520us
Vbas= 653mV
Prd=8.880us
+Duty=50.0%
Vamp= 2.10V
Freq=112.6kHz
-Duty=50.9%
CH1 500mV
1.00V
Time 2.000us
1.480ms

图 3- 2