应用笔记

ACM32H5 系列芯片

DAC 应用笔记

版本：V0.1

日期：2025-03-21

上海航芯电子科技股份有限公司

1. 概述

本应用手册适用于 ACM32H5 系列芯片 DAC 模块。它描述了与 DAC 模块相关的设置和功能使用方法，以便在应用程序中进行优化设计。关于 DAC 的转换速度，精度等各项电气特性参数，详见相关数据手册

本应用手册需要结合用户手册中对应的 DAC 模块部分、SDK 中相关的 demo 及 HAL 库驱动一起阅读。

2. DAC 概述

ACM32H5xx 系列有 2 个 12bit 精度的数字/模拟转换器（DAC）模块（DAC1 和 DAC2）。它可以将 12 位的数字数据转换为外部引脚上的电压输出。数据可以采用 8 位或 12 位模式，在 12 位模式下，数据可以采用左对齐或右对齐模式。当使能了外部触发，DMA 可被用于更新输入端数字数据。

每个 DAC 模块有 2 个输出通道，每个通道都有一个单独的转换器。在 DAC 双通道模式下，2 个通道可以独立地进行转换，也可以同时进行转换并同步地更新 2 个通道的输出。

DAC 输出到 PAD 可以断开，内部连接到芯片内其它模拟外设。在输出电压时，可以利用 DAC 输出 BUFFER 来获得更高的驱动能力。DAC 输出支持在低功耗模式下的采样保持模式。

2.1. 主要特性

- 8 位或者 12 位分辨率
- 12 位模式下数据左对齐或右对齐
- 每个输出通道均为单独的转换器
- 支持有符号数输入
- 自带噪声波形生成功能
- 自带三角波形生成功能
- 自带锯齿波形生成功能
- DAC 双通道独立或同时转换
- DMA 双数据模式降低总线开销
- 每个通道都有 DMA 功能
- 外部触发转换
- 输出 BUFFER 可选，BUFFER 偏差可校准
- 每个通道可以与 PAD 断开且可以输出到内部互联模块
- STOP 模式支持采样保存功能
- 输入参考电压 VREFP

3. 应用使用说明

DAC 的功能简单的说就是根据输入的基准源电压，将数字码转换成模拟电压输出。除了需要让 DAC 模块本身工作的电压 VDDA 之外，基准源输入电压 VREFP 必不可少。并且 VREFP 的精度直接影响到 DAC 输出的精度。

DAC 输出转换公式：

DAC output voltage = (VREFP / 4095) DOR

其中 DOR 为数字码

3.1. DAC 的 VREFP

DAC 的 VREFP 可以选择外部的电压输入，也可以选择使用内部的 VERFBUF 作为 DAC 输出的参考电压。VREFP 上的任何扰动（例如，噪声干扰，绝对精度，温度）都将直接成比例的作用到 DAC 的输出电压上。

3.2. 输出缓冲 buffer

ACM32H5xx 的 DAC 模块内部自带缓冲器 buffer，可配置成带 buffer 输出和不带 buffer 输出。它们的区别如下；

图 3-1 没有输出 buffer 的通道电压（有负载/无负载）

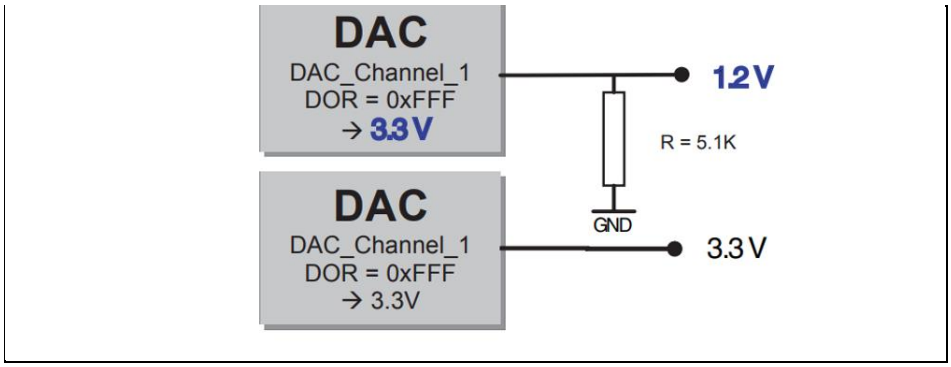
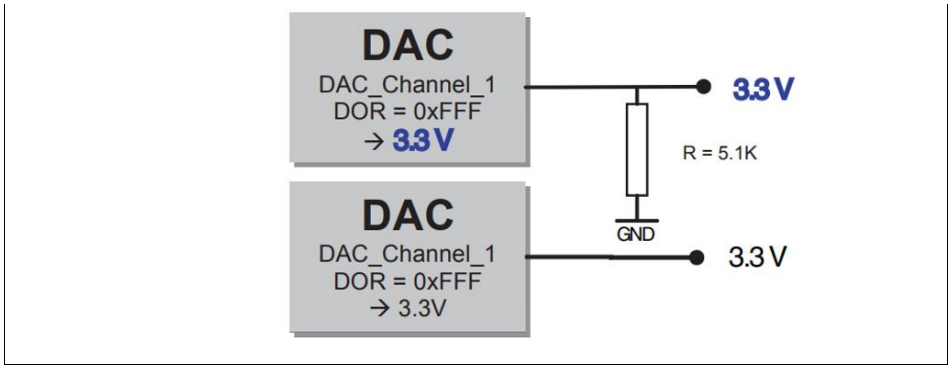


图 3-2 带输出 buffer 的通道电压（有负载/无负载）



3.3. 双通道模式

DAC 有两个输出通道，每个通道各有一个转换器。在双 DAC 通道模式下，转换可以单独进行，也可以同时进行。 当 DAC 通道由同一个触发源触发后，两个通道将组合在一起同步执行更新操作，转换也会同时行。

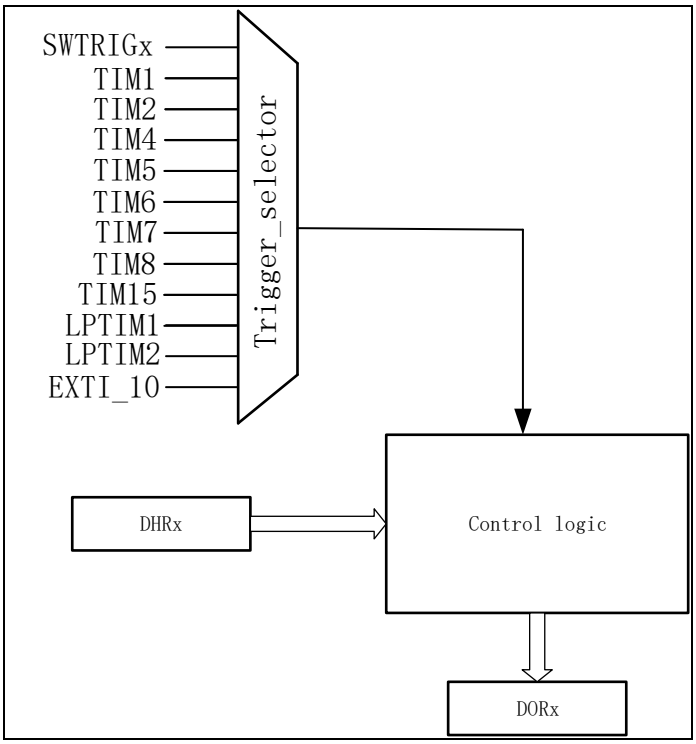
3.4. DAC 转换的触发

DAC 转换的触发，即 DHR 里的数据加载到 DOR，有以下几种。具体选择那种触发方式由实际应用决定。

- 直通方式

如果没有选中硬件触发（即寄存器 DAC_CR1 的 TENx 位置 0），那么写入到 DAC_DHRx 的数据会自动加载到 DAC_DORx 进行输出。

图 3-3 DAC 触发源选择



- 软件触发

当 DAC_CR1 的 TEXx 位=1 时，那么 DAC_DHRx 里的数据会在软件置位 DAC_SWTRIGR 寄存器的 SWTRIGx 后才加载到 DORx 进行输出。

- 定时器触发

  可以选择 TIM1/TIM2/TIM4/TIM5/TIM6/TIM7/TIM8/TIM15 内部互联的方式来作为 DAC 的触发源。

- 低功耗定时器触发

  可以选择 LPTIM1/LPTIM2 内部互联的方式来作为 DAC 的触发源

- 外部管脚触发

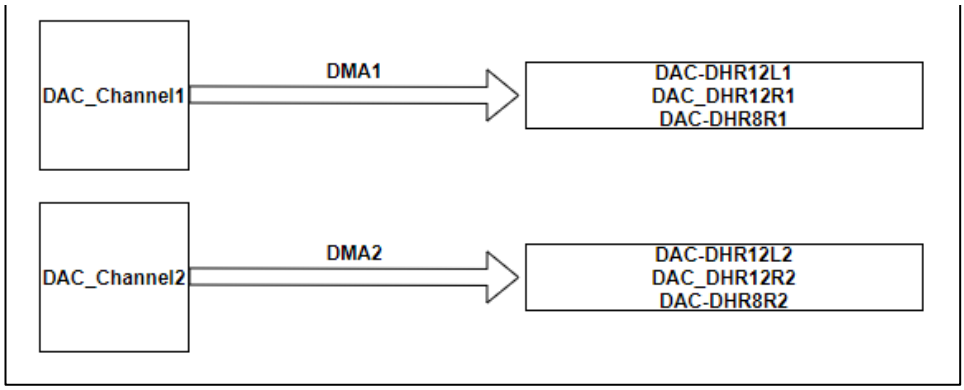
外部管脚 EXTI9 可以作为 DAC 的转换触发，也可以作为锯齿波发生器的复位触发。

外部管脚 EXTI10 可以作为锯齿波发生器的 Setp 触发

3.5. DMA 搬运

当需要输出复杂的电压波形信号时，使用 DMA 来搬运数据到 DAC 的 DOR 将有效的减轻 CPU 的负载。每个 DAC 通道都具有 DMA 功能。2 个 DMA 通道可分别用于 2 个 DAC 通道的 DMA 请求。一旦有外部触发(而不是软件触发)发生，则产生一个 DMA 请求,通过 DMA 可将要输出到 DAC 的数据搬运至通道数据寄存器中。在单 DAC 模式下，可使能对应通道的 DMA 请求，完成 DMA 传输，原理如下图所示:

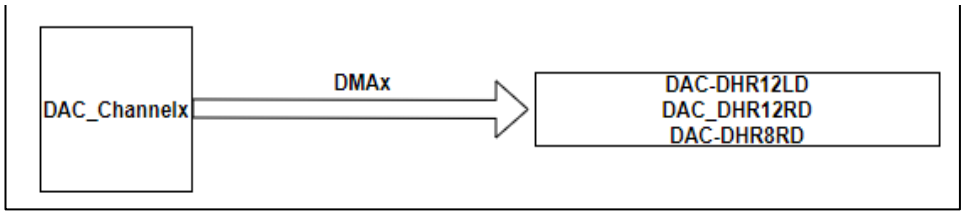
图 3-4 DAC 搬运



3.5.1. DMA 双通道

而在双 DAC 模式下，如果 2 个通道都使能 DMA，如上图一般，则会产生 2 个 DMA 请求。如果实际只需要一个 DMA 传输，则应只选择使能其中一个 DAC 通道的 DMA，利用一个通道的 DMA 将数据搬运至双通道数据寄存器中。这样，程序可以在只使用一个 DMA 请求，一个 DMA 通道的情况下，处理工作在双 DAC 模式的 2 个 DAC 通道，原理如图。

图 3-5 DAC DMA 双通道



3.5.2. DMA 双数据

在普通的模式下，一个 DMA 请求只传输 12 位（或 8 位）有效数据。AHB 总线位宽是 32 位的，理论上一次传输可以同时写入两个 12 位数据。设置 DAC_MCR 寄存器的 DMADOUBLEx 位可以使能一次 DMA 请求传输两个 12 位数据模式，即 DMA 双数据模式。

当使能 DMA 双数据模式，每两次外部触发产生一次 DMA 请求。

当检测到第一次外部触发，DAC_DHRx 和 DAC_DHRBx 的数据会传送到 DAC_DORx 和 DAC_DORBx 寄存器。当前有效的 DAC 数据存放在 DAC_DORx 寄存器中。随后，产生一次 DMA 请求。DMA 写入新的数据到 DAC_DHRx 和 DAC_DHRBx 中。

当检查到第二次外部触发，当前有效的 DAC 数据存放到在 DAC_DORBx 中（通过第一次触发从 DAC_DHRBx 传送），不会产生新的 DMA 请求。状态寄存器中的 DORSTATx 位表明当前 DOR 中哪个数据作用于模拟 DAC 转换。第二次触发会把 DORBx 数据搬运到 DORx 寄存器输出；如果设置了噪声叠加，数据从 DORBx 搬运到 DORx 时，会叠加上噪音。

3.6. DAC 初始化

DAC 初始化使用函数 HAL_DAC_Init，主要是对 DAC 使用的相关的输出管脚进行初始化配置。

HAL_StatusTypeDef HAL_DAC_Init(DAC_HandleTypeDef *hdac)

```
{
    /* Check the parameters */
    assert_param(IS_DAC_INSTANCE(hdac->Instance));
    HAL_DAC_MspInit(hdac);
    return HAL_OK;
```

```
}
```

DAC 初始化示例如下：

```
DAC_HandleTypeDef hdac1;
hdac1.Instance = DAC1;
HAL_DAC_Init(&hdac1);
```

3.7. DAC 通道配置

DAC 的通道配置使用函数 HAL_DAC_ConfigChannel。在调用该函数前，需要对相关参数进行配置：

```
typedef struct
{
    FunctionalState DAC_DMADoubleDataMode; /*!< Specifies if DMA double data mode should be enabled or
                                              not for the selected channel.
                                              This parameter can be ENABLE or DISABLE */

    FunctionalState DAC_SignedFormat;      /*!< Specifies if signed format should be used or not for the
                                              selected channel. This parameter can be ENABLE or DISABLE
                                              */

    uint32_t DAC_SampleAndHold;           /*!< Specifies whether the DAC mode.
                                             This parameter can be a value of @ref DAC_SampleAndHold */
    union{
        uint32_t DAC_Trigger;             /*!< DAC normal trigger */
        uint32_t SawtoothResetTrigger;     /*!< DAC SawTooth wave reset trigger */
    }u;

    uint32_t SawtoothStepTrigger;         /*!< DAC SawTooth wave step trigger */

    uint32_t DAC_OutputBuffer;            /*!< Specifies whether the DAC channel output buffer is
                                             enabled or disabled.
                                             This parameter can be a value of @ref DAC_output_buffer */

    uint32_t DAC_ConnectOnChipPeripheral ; /*!< Specifies whether the DAC output is connected or not to
                                              on chip peripheral .
                                              This parameter can be a value of @ref
                                              DAC_ConnectOnChipPeripheral */

    uint32_t DAC_UserTrimming;            /*!< Specifies the trimming mode
                                             This parameter must be a value of @ref DAC_UserTrimming
                                             DAC_UserTrimming is either factory or user trimming */

    uint32_t DAC_TrimmingValue;            /*!< Specifies the offset trimming value
                                              i.e. when DAC_SampleAndHold is DAC_TRIMMING_USER.
                                              This parameter must be a number between Min_Data = 1 and
                                              Max_Data = 31 */

    DAC_SampleAndHoldConfTypeDef  DAC_SampleAndHoldConfig;  /*!< Sample and Hold settings */
}DAC_ChannelConfTypeDef;
```

现以 DAC 输出锯齿波为例介绍通道的配置：

```
void DAC_Config_OutPut_Sawtooth(DAC_TypeDef* DACx)
{
    DAC_ChannelConfTypeDef sConfig={0};
```

```
    hdac1.Instance = DAC1;
    HAL_DAC_Init(&hdac1);                                          //DAC1 初始化，管脚配置，DAC 时钟开启
    sConfig.SawtoothResetTrigger = DAC_TRIGGER_SOFTWARE;     //通道 1 锯齿波复位触发：软件触发
    sConfig.SawtoothStepTrigger = DAC_TRIGGER_SOFTWARE;      //通道 1 锯齿波 Setp 触发：软件触发
    sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;  //采样保持模式关闭
    sConfig.DAC_OutputBuffer =  DAC_OUTPUTBUFFER_DISABLE;    //输出 buffer 关闭
    sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_EXTERNAL; //输出到外部管脚
    sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;         //DAC TRIM 值使用出厂校准值
    /* DAC 通道 1 锯齿波 reset:软件触发,step:软件触发 */
    HAL_DAC_ConfigChannel(&hdac1, &sConfig,DAC_CHANNEL_1);  //DAC1 通道 1 配置

    /* DAC 通道 2 锯齿波 reset:软件触发,step:TIM15 触发 */
    sConfig.SawtoothResetTrigger = DAC_TRIGGER_SOFTWARE;     //通道 2 锯齿波复位触发：软件触发
    sConfig.SawtoothStepTrigger = DAC_TRIGGER_T15_TRGO;      //通道 1 锯齿波 Step 触发：TIM15 触发
    HAL_DAC_ConfigChannel(&hdac1, &sConfig,DAC_CHANNEL_2);  //DAC1 通道 2 配置
}
```

3.7.1. DAC 的 trim 校准

如上节所示，DAC 的 TRIM 配置可以使用出厂时的校准值：

```
sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
```

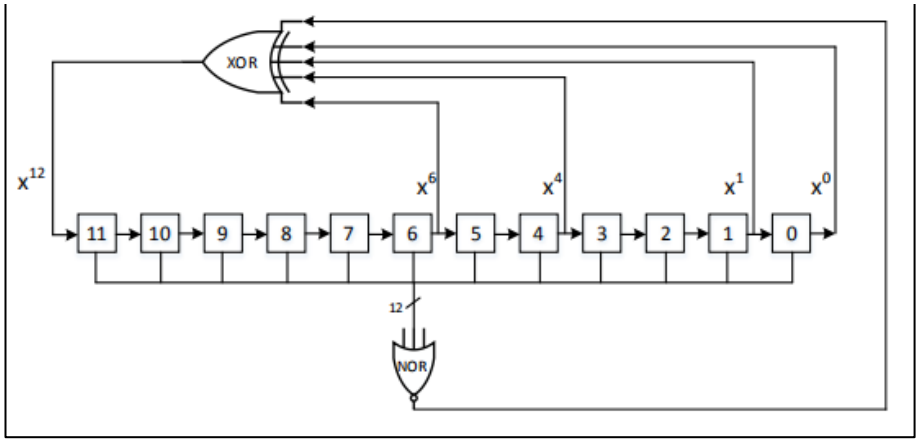
根据实际使用场景，也可以使用自校准函数 HAL_DACEx_SelfCalibrate 来重新进行校准

3.8. 典型应用

3.8.1. 输出白噪声

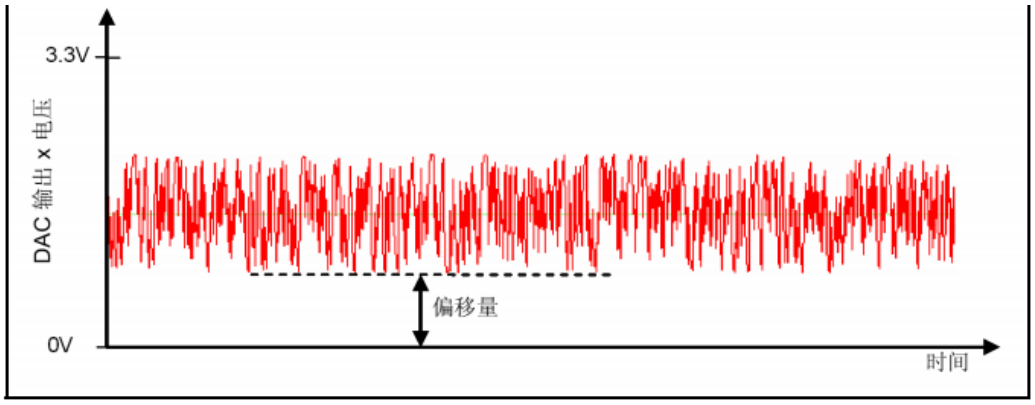
ACM32H5xx 系列芯片的 DAC 为用户提供了一个伪随机码发生器。根据移位寄存器上使用的节拍数，在序列重复前，可生成具有最多 2n-1 个数的序列。

图 3-6 DAC 伪随机码发生器



由噪声发生器生成的噪声具有均匀的频谱分布，可将这些噪声视为白噪声。不过，白噪声分布均匀，不具备高斯输出特性。

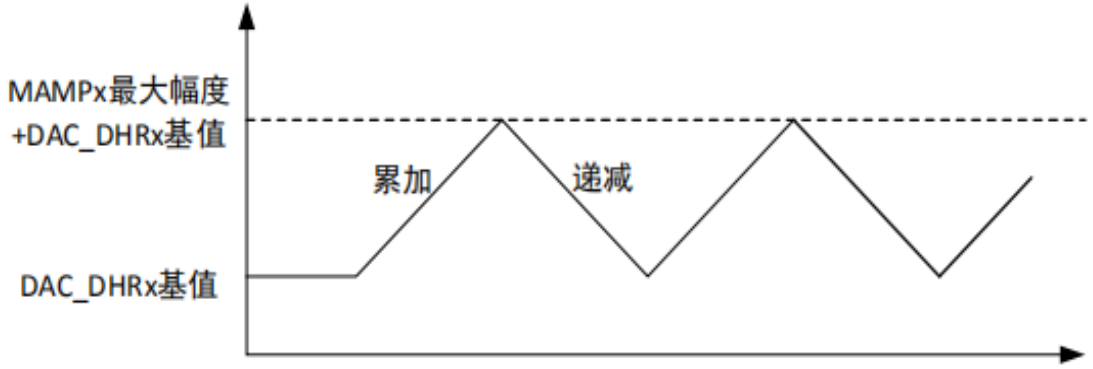
图 3-7 DAC 输出白噪声



3.8.2. 输出三角波

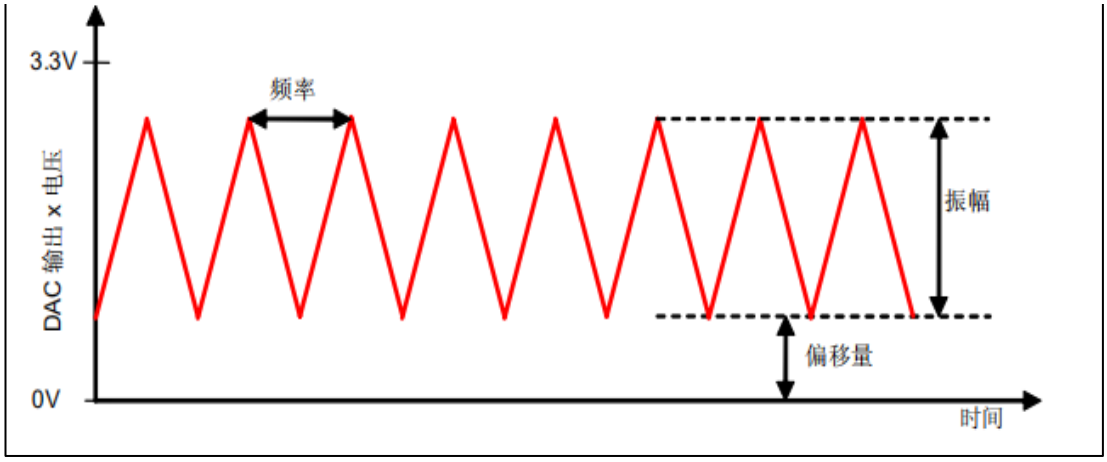
ACM32H5xx 系列芯片的 DAC 为用户提供了具有灵活的偏移量、振幅和频率的三角波形发生器。 理论上说，三角波形是一种由无限组奇次谐波组成的波形，可以使用 DAC_CR 寄存器中的 MAMPx 位修正三角波形的振幅。

图 3-8 DAC 输出三角波



其中，三角波形频率与触发源的频率相关，三角波的幅值与基值以及 MAMPx 的设置有关。

图 3-9 DAC 三角波频率、振幅及基值示意图



3.8.3. 输出锯齿波

ACM32H5xx 系列芯片的 DAC 有自主生成锯齿波的功能，只需要配置如下参数：

- 锯齿波的复位值（也称初始值），每个 step 的增量值及方向
- 锯齿波的触发源，step 触发，reset 触发

- 锯齿波的复位值、增量值及方向通过 DAC_STRx. STRSTDATAx、DAC_STRx. STINCDATAx 和 DAC_STRx. STDIRx（其中 x 代表 0 或 1）来配置
- 锯齿波的增量触发通过 DAC_STMODR. STINCTRIGSELx(x 代表 0 或 1)来配置
- 锯齿波的复位触发通过 DAC_STMODR. STRSTTRIGSELx(x 代表 0 或 1)来配置。

以通道 1 的锯齿波发生器举例说明：锯齿波计数器的值是从 DAC_STR1.STRSTDATA1 的值开始的，每一个增量触发后都会让该计数器的值增加 DAC_STR1.STINCDATA1。当锯齿波的计数器的值达到 0x0000 或者 0xFFFF，就达到了满量程值不会继续变化。锯齿波的复位触发将使得计数器重新加载成复位值。如果在到达满量程值前，还没有锯齿波的复位触发，那么锯齿波将会产生削尖的波形。

图 3-10 DAC 输出锯齿波递增模式（STDIR1 = 1）

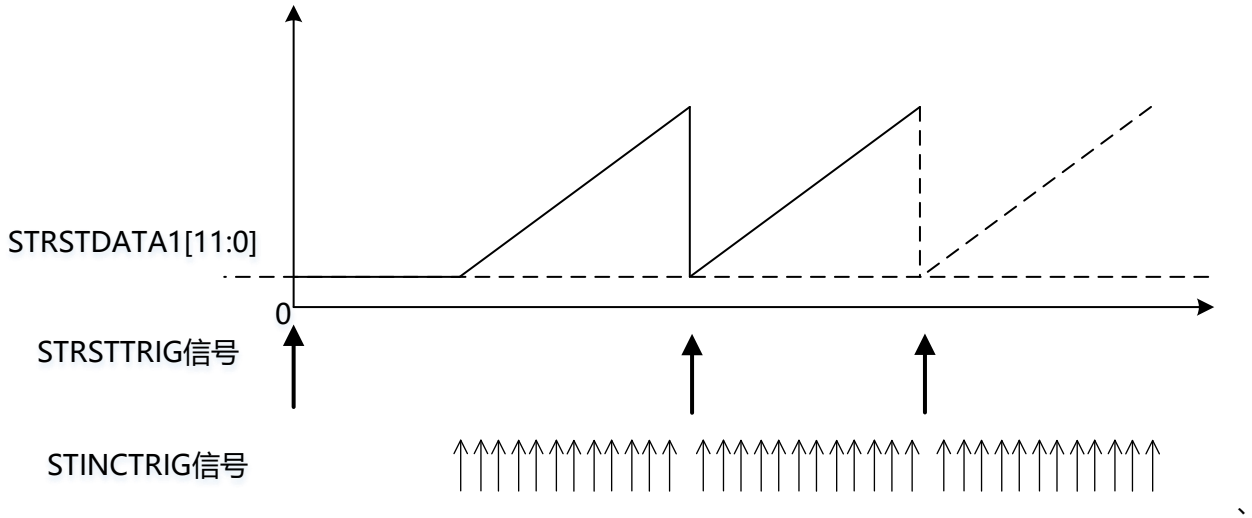
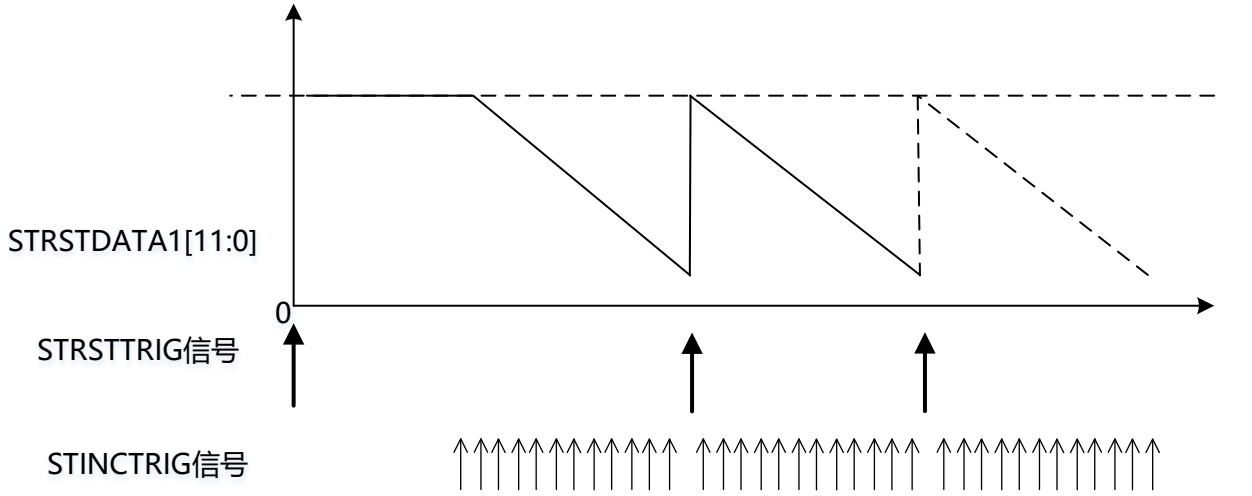


图 3-11 DAC 输出锯齿波递减模式（STDIR1 = 0）

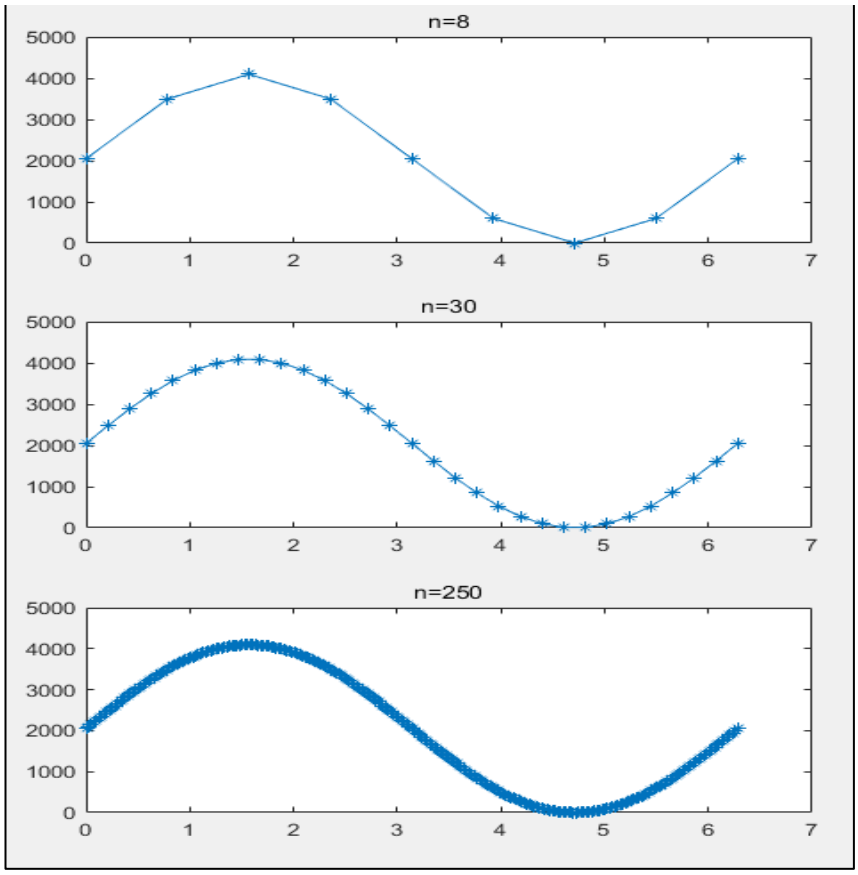


3.8.4. 输出正弦波

■ 生成正弦波原理

要输出正弦波，实质是要控制 DAC 以 V=sin(t)的正弦函数关系输出电压，其中 V 为电压输出，t 为时间。而由于模拟信号连续而数字信号是离散的，所以使用 DAC 产生正弦波时，只能按一定时间间隔输出正弦曲线上的点，在该时间段内输出相同的电压值，若缩短时间间隔，提高单个周期内的输出点数，可以得到逼近连续正弦波的图形，见图，点数越多，越能更好的还原正弦波，若在外部电路加适当的电容滤波，可得到更完美的图形。

图 3-12 DAC 输出不同正弦波的点数示意图



■ 生成正弦波数据表

由于正弦曲线是周期函数，所以只需要得到单个周期内的数据后按周期重复即可，而单个周期内取样输出的点数又是有限的，所以为了得到呈 v=sin(t)函数关系电压值的数据通常不会实时计算获取，而是预先计算好函数单个周期内的电压数据表，并且转化成以 DAC 寄存器表示的值。

sinx 函数值的范围为[-1: +1]，而 DAC 输出电压范围为[0~3.3]V，按 12 位 DAC 分辨率表示的方法，可写入寄存器的最大值为 0xFFF = 4095，即范围为[0:4095]。所以，实际输出时，需进行如下处理：

抬升 sin 函数的输出为正值：V = sin(t)+1 ，此时，V 的输出范围为[0:2]；

扩展输出至 DAC 的全电压范围: V = 3.3*(sin(t)+1)/2 ，此时，V 的输出范围为[0:3.3]，正是 DAC 的电压输出范围，扩展至全电压范围可以充分利用 DAC 的分辨率；

把电压值[0:3.3]转换为[0,4095]以 DAC 寄存器的形式表示：Reg_val = 4095 *V/3.3,此时，存储到 DAC 寄存器的值范围为[0:4095]；

实践证明，在 sin(t)的单个周期内，至少取 30 个点进行电压输出才能较好地还原正弦波形，所以在 t∈[0:2Pi]区间内等间距根据上述 Reg_val 公式运算得到 n 个寄存器值，利用 matlab 或 python 等脚本即可得到正弦波表；

控制 DAC 输出时，每隔一段相同的时间从上述正弦波表中取出一个新数据进行输出，即可输出正弦波。改变间隔时间的单位长度，或正弦波的样本点数，可以改变正弦波曲线的周期。

采样(n)	数字采样值	模拟采样值（电压）
1	2048	1.65

2	3649	2.94
3	4045	3.25
4	2937	2.36
5	1159	0.93
6	51	0.04
7	447	0.36
8	2048	1.65

■ 正弦波频率计算

有了 2.2 中的准备，就可以生成正弦波信号了，只需要在一定时间内，循环的将正弦波数据搬运至 DAC 数据寄存器中，即可生成正弦波信号了。

为了更好的控制正弦波的频率，本应用中采用了 Timer+DMA+DAC 输出正弦波的方案，利用定时器得 TRGO 信号触发 DAC 得 DMA 请求搬运正弦波数据，则正弦波的频率跟定时器的 TRGO 触发时间和正弦波样本数量 n 相关，计算方法如下：

按照默认配置，系统时钟 120M，PCLK=60M，TimCLK=PCLK*2=120M

配置定时器分频系数 Prescaler,则分频后的定时器频率=120M/(Prescaler+1)

配置定时器的计算值 Period，则定时器触发 TRGO 信号的时间

=(Period+1)/（120M/(Prescaler+1)）

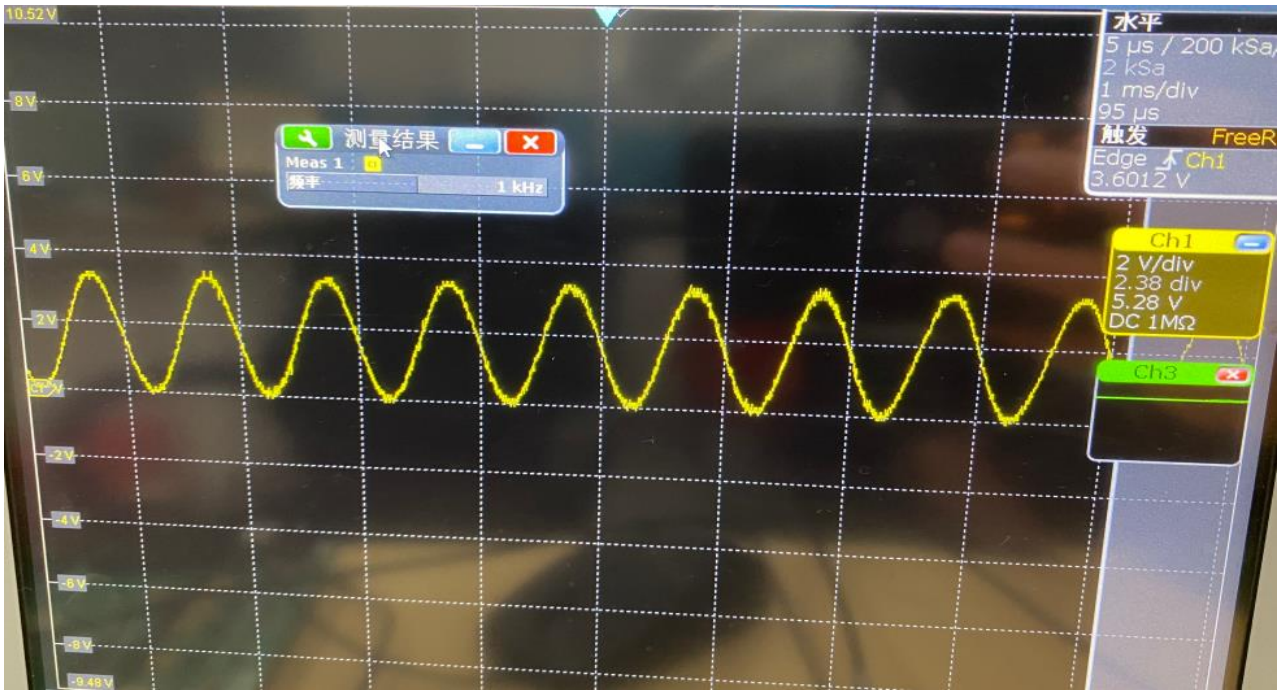
根据正弦波单个周期的样本数 N，可得正弦波得频率为：

F=120M/((Prescaler+1)*(Period+1)*N)

在本应用中，正弦波样本数 N=250 个，Prescaler=1-1（不分频），Period=480-1，故

F=120M/(1*480*250)=1000HZ。

图 3-13 DAC 输出正弦波示波器测量



4. 版本历史

版本	日期	作者	描述
V0.1	2025-03-21	AisinoChip	First release

联系我们

公司：上海航芯电子科技股份有限公司

地址：上海市闵行区合川路 2570 号科技绿洲三期 2 号楼 702 室

邮编：200241

电话：+86-21-6125 9080

传真：+86-21-6125 9080-830

Email： service@aisinochip.com

Website： www.aisinochip.com