AN0101
应用笔记

AT32 MCU DAC Application Note

前言

本应用入门指南透过 DAC 的特色功能讲解和案列解析，旨在让用户快速使用 AT32 的 DAC 进行项目开发。

数模转换器（DAC）采用 12 位/8 位数字输入，产生 0 至参考电压 V_{REF+}之间的模拟输出。数字部分可以配置为 8 位或者 12 位模式，支持单/双 DAC 的左对齐或者右对齐，同时可以与 DMA 配合使用。

下表 AT32 型号均支持两路 DAC，DAC1/DAC2 各有一个数模转换器，每个 DAC 可以独立地进行数模转换，也可以双 DAC 同时触发进行转换，输入参考电压 V_{REF+}可以使转换结果更加精确。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

支持型号列表：

支持型号	具备 DAC 的型号

目录

表目录

图目录

1 DAC 概述

数模转换器（DAC）采用 12 位/8 位数字输入，产生 0 至参考电压 V_{REF+}之间的模拟输出。

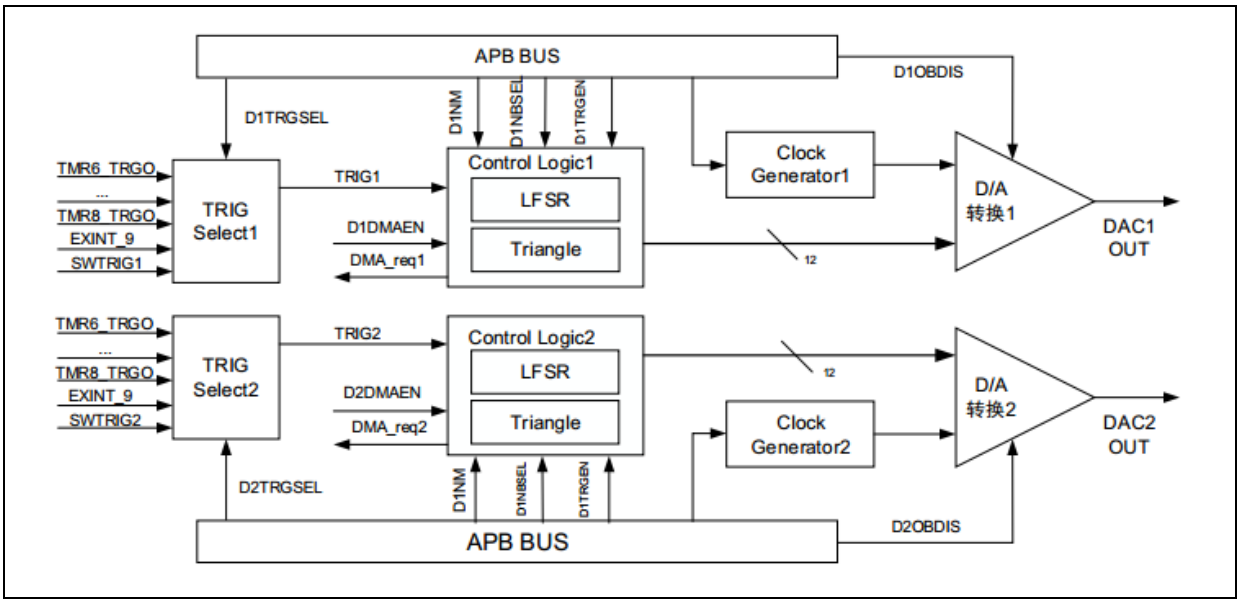
DAC 主要特性：

- 2 个 DAC 转换器：每个转换器对应 1 个输出通道
- 单/双 DAC 8 位或者 12 位数字输入
- 12 位数据支持左对齐或者右对齐模式
- 支持噪声波/三角波产生
- 支持双 DAC 或者单个 DAC1/DAC2 独立转换
- 每个 DAC 支持 DMA 模式
- 软件触发或者外部触发转换
- 支持输入参考电压 V_{REF+}

2 DAC 功能描述

单个 DAC 通道的框图如下图：

图 1.DAC1/DAC2 模块框图



2.1 DAC 引脚说明

数字输入经过 DAC 线性地转换为模拟电压输出，其范围为 0 至 V_{REF+}。

模拟 DAC 模块采用 VDDA 供电，输入正模拟参考电压大小介于 2.0V 与 VDDA 之间，PA4/PA5 作为模拟输出。

一般来说，为保证低电压时 DAC 的高精度，数字电路由 VDD 供电，模拟电路由 VDDA 独立供电，在 64PIN 封装及以下型号中，外部参考电压 V_{REF+}连接至 VDDA 管脚。

任一 DAC 通道上的输出电压满足下面的关系：

$$\text{DAC 输出} = V_{REF+} \times \frac{\text{DAC\_DxODT[11：0]}}{4095}$$

下表给出了 DAC 使用到的引脚说明：

表 1.DAC 引脚说明

名称	型号类型	注释
V_{REF+}	输入，正模拟参考电压	DAC 使用的输入参考电压，$2.0V \le V_{REF+} \le V_{DDA}$
V_{DDA}	输入，模拟电源	模拟电源
V_{SSA}	输入，模拟电源地	模拟电源的地线
DAC_OUTx	模拟输出信号	DAC 通道 x 的模拟输出

注意：一旦使能 DACx 通道，相应的 GPIO 引脚（PA4 或者 PA5）就会自动与 DAC 的模拟输出相连（DACx_OUT）。为了避免寄生的干扰和额外的功耗，引脚 PA4 或者 PA5 在此之前应当设置成模拟模式。

2.2 数据配置

不能直接对数据输出寄存器 DAC_DxODT 写入数据，任何输出到 DAC 通道 x 的数据都必须写入数据保持寄存器（DAC_DxDTH8R、DAC_DxDTH12L、DAC_DxDTH12R、DAC_DDTH8R、DAC_DDTH12L 或者 DAC_DDTH12R 寄存器）。

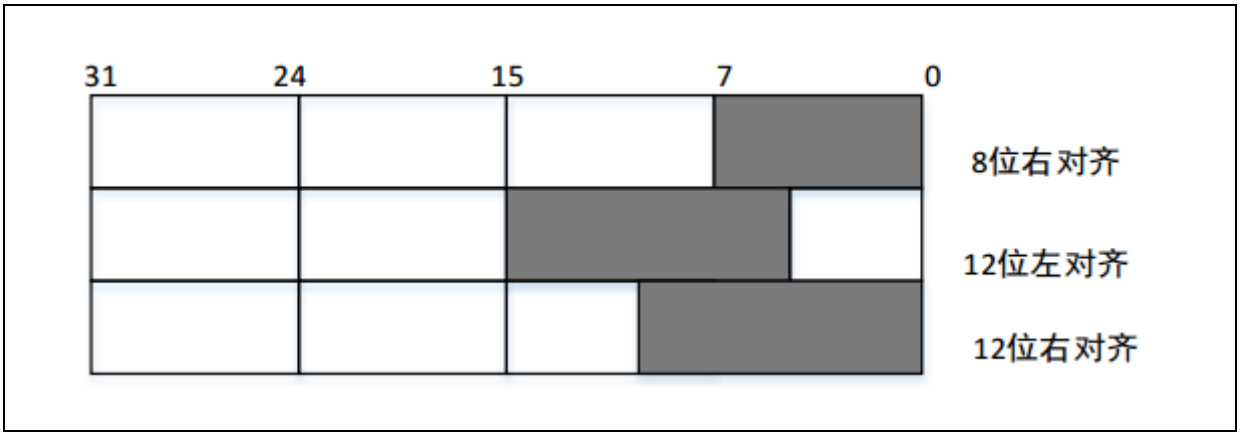
DAC 支持单 DAC 或者双 DAC 模式，根据模式的不同，数据配置有以下方式：

单 DAC 数据配置方式：

- 采用 8 位数据右对齐方式时，对寄存器 DAC_DxDTH8R[7：0]位写入值。
- 采用 12 位数据左对齐方式时，对寄存器 DAC_DxDTH12L[15：4]位写入值。
- 采用 12 位数据右对齐方式时，对寄存器 DAC_DxDTH12R[11：0]位写入值。

根据对 DAC_DxDTHx 寄存器的操作，经过相应的移位元后，写入的资料被转存到 DHRx 寄存器中（DHRx 是内部的数据保存寄存器 x，写入的 8 位数据对应 DHRx[11：4]位，写入的 12 位数据对应 DHRx[11：0]位）。随后，DHRx 寄存器的内容或被自动地传送到 DAC_DxODT 寄存器，或通过软件触发，或通过外部事件触发被传送到 DAC_DxODT 寄存器。

图 2.单 DAC 通道模式的数据寄存器

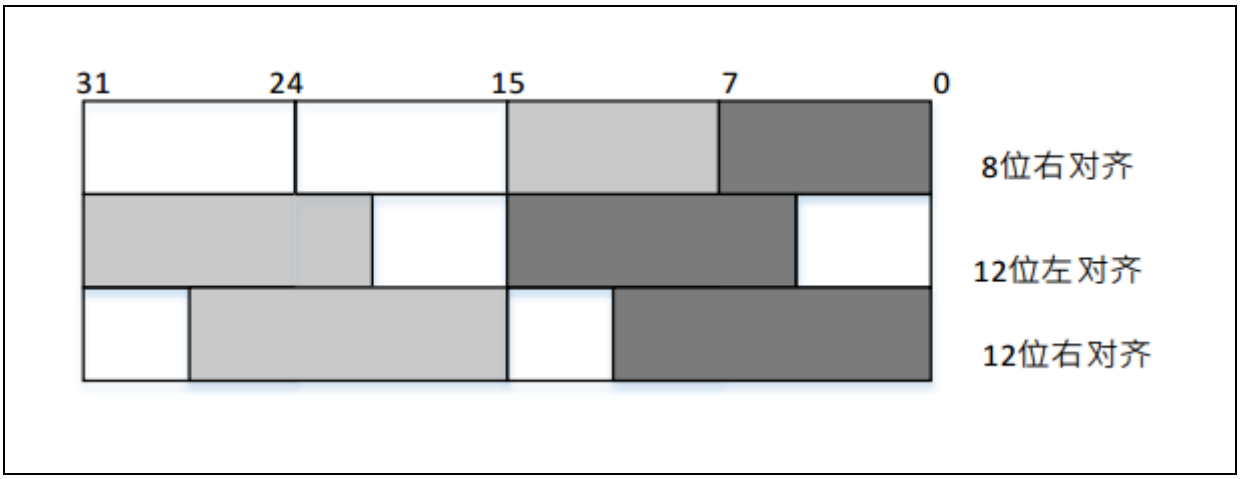


双 DAC 数据配置方式：

- 采用 8 位数据右对齐方式时，对双 DAC 的 8 位右对齐数据保持寄存器 DAC_DDTH8R[7：0]和双 DAC 的 8 位右对齐数据保持寄存器 DAC_DDTH8R[15：8]位写入值。
- 采用 12 位数据左对齐方式时，对双 DAC 的 12 位左对齐数据保持寄存器 DAC_DDTH12L[15：4]和双 DAC 的 12 位左对齐数据保持寄存器 DAC_DDTH12L[31：20]位写入值。
- 采用 12 位数据右对齐方式时，对双 DAC 的 12 位右对齐数据保持寄存器 DAC_DDTH12R[11：0]和双 DAC 的 12 位右对齐数据保持寄存器 DAC_DDTH12R[27：16]位写入值。

根据对 DAC_DDTHx 寄存器的操作，经过相应的移位元后，写入的资料被转存到 DHRx 寄存器中（DHRx 是内部的数据保存寄存器 x，写入的 8 位数据对应 DHRx[11：4]位，写入的 12 位数据对应 DHRx[11：0]位）。随后，DHR1 和 DHR2 寄存器的内容或被自动地传送到 DAC_DxODT 寄存器，或通过软件触发，或通过外部事件触发被传送到 DAC_DxODT 寄存器。

图 3.双 DAC 通道模式的数据寄存器



2.3 触发事件选择

如果 DAC 控制寄存器（DAC_CTRL）的 DxTRGEN 位被置 1，DAC 转换可以由某外部事件（定时器计数器、外部中断线）或者软件触发，触发事件源由 DxTRGSEL[2：0]进行选择。

注意：不能在 DxEN 位为 1 时改变 DxTRGSEL[2：0]位。

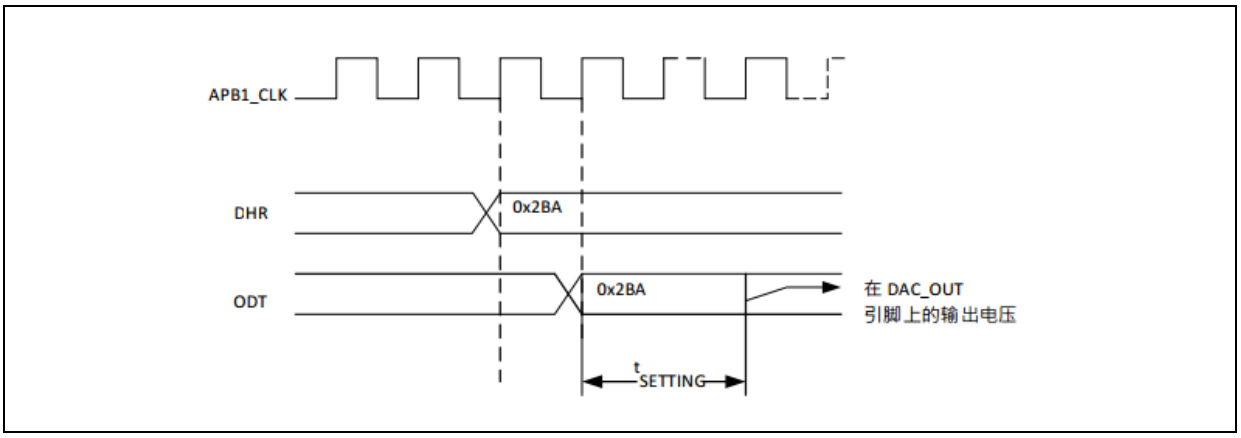
表 2.DAC 触发源选择

触发源	DxTRGSEL [2：0]	说明
TMR6_TRGOUT	000	片上信号
TMR3_TRGOUT	001	
TMR7_TRGOUT	010	
TMR9_TRGOUT	011	
TMR2_TRGOUT	100	
TMR4_TRGOUT	101	
EXINT_9	110	外部信号
DxSWTRG	111	软件触发

- 当 DxTRGEN 位被置 1 时，每次 DAC 侦测到有效的的触发事件（定时器 TRGO 输出或外部中断线 9 的上升沿），转换即开始，存放在 DHRx 中的数据就会被传到 DACx 数据输出寄存器（DAC_DxODT）中。在 3 个 APB1 时钟周期后，DAC_DxODT 寄存器更新为新值。
- 当 DxTRGEN 位被置 1 且选择软件触发时，一旦 DxSWTRG 被软件置 1 后，转换即开始，在 1 个 APB1 时钟周期后，DAC_DxODT 寄存器更新为新值。在数据传送到 DACx 数据输出寄存器（DAC_DxODT）后，DxSWTRG 位由硬件自动清零。
- 当 DxTRGEN 位被清 0 时，每次写入数据保持寄存器值，转换即开始，无需等待触发事件。在 1 个 APB1 时钟周期后，数据即被传送到 DACx 数据输出寄存器（DAC_DxODT）中。

一旦数据从 DHRx[11：0]寄存器装入 DAC_DxODT 寄存器，在经过时间 $t_{SETTLING}$ 之后，输出即有效，这段时间的长短根据电源电压和模拟输出负载的不同会有所变化。

图 4.DxTRGEN=0 触发禁止时转换的时间框图



2.4 DMA

任一 DAC 支持 DMA 功能，通过设置 DAC 控制寄存器（DAC_CTRL）的 DxDMAEN 位使能 DMA 请求。当触发使能位 DxTRGEN 有效，触发信号有效时，即产生 DMA 请求，然后 DHRx 寄存器的数据被传送到 DAC_DxODT 寄存器。

在双 DAC 模式下，如果 2 个通道的 DxDMAEN 位都为 1，则会产生 2 个 DMA 请求。如果实际只需要一个 DMA 传输，则应只选择其中一个 DxDMAEN 位置 1。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。

注意：DAC 的 DMA 请求不会累计，未来得及处理的 DMA 请求将被忽略，也不会产生错误信息。因此如果第 2 个外部触发发生在响应第 1 个外部触发之前，则不能处理第 2 个 DMA 请求。

2.5 噪声/三角波生成

有噪声波和三角波两种波形可以叠加到 DAC 输出：分别利用线性回馈移位寄存器 LFSR（LENinear Feedback Shift Register）产生幅度变化的伪噪声和通过三角波发生器（Triangle）产生三角波。当设置 DAC 控制寄存器（DAC_CTRL）的 DxNM[1：0]位为‘01’使能 LFSR，输出幅度变化的伪噪声；当设置 DxNM[1：0]位为‘1x’使能三角波发生器，输出三角波。

LFSR 原理

寄存器 LFSR 的预装入值为 0xAAA，按照特定算法，在每次触发事件之后更新该寄存器的值。设置 DAC 控制寄存器（DAC_CTRL）的 DxNBSEL[3：0]位可以屏蔽部分或者全部 LFSR 的数据，这样得到的 LSFR 值与 DHRx 的数值相加，去掉溢出位之后即被写入 DACx 数据输出寄存器（DAC_DxODT）。

将 DxNM[1：0]位置’00’可以关掉 LFSR 功能，同时复位 LFSR 波形的生成算法。

三角波原理

三角波的幅度由 DAC 控制寄存器（DAC_CTRL）的 DxNBSEL[3：0]位设置，内部的三角波计数器每检测到一次触发事件累加 1，当达到 DxNBSEL[3：0]位定义的最大幅度时，则计数器开始递减，达到 0 后再开始累加，周而复始。同时，计数器的值与 DHRx 寄存器的数值相加并丢弃溢出位后写入 DACx 数据输出寄存器（DAC_DxODT）。

将 DxNM[1：0]设置‘00’可以关掉三角波生成，同时复位三角波计数器。

注意 1：为了产生噪声/三角波，必须使能 DAC 触发，即设 DAC_CTRL 寄存器的 DxTRGEN 位为 1。

注意 2：DxNBSEL[3：0]位必须在使能 DAC 之前设置， 否则其值不能修改。

2.6 DAC 程序设计

- DACx 触发源配置
  - 设置 DACx 通道的触发使能位 DxTRGEN；
  - 设置 DACx 的触发事件选择位 DxTRGSEL 为相应值，选择触发源。
- DACx 噪声/三角波生成选择
  - 如果需要生成噪声，则将 DAC 控制寄存器（DAC_CTRL）的 DxNM[1：0]位置‘01’，并根据需要配置 DxNBSEL[3：0]选择屏蔽位；
  - 如果需要生成三角波，则将 DAC 控制寄存器（DAC_CTRL）的 DxNM[1：0]位置‘1x’，并根据需要配置 DxNBSEL[3：0]选择波形的幅值；
  - 如果不使用该功能，则将 DxNM[1：0]位置’00’。
- 将输出到 DACx 的数据写入 DAC_DxDTHx/DAC_DDTHx 数据保持寄存器
  - DACx 的 8 位右对齐数据保持寄存器（DAC_DxDTH8R）；
  - DACx 的 12 位左对齐数据保持寄存器（DAC_DxDTH12L）；
  - DACx 的 12 位右对齐数据保持寄存器（DAC_DxDTH12R）；
  - 双 DAC 的 8 位右对齐数据保持寄存器（DAC_DDTH8R）；
  - 双 DAC 的 12 位左对齐数据保持寄存器（DAC_DDTH12L）；
  - 双 DAC 的 12 位右对齐数据保持寄存器（DAC_DDTH12R）。
- DMA 配置
  - 根据需要进行 DMA 配置。
- DACx 输出缓存配置
  - DAC 集成了输出增益，可以用来减少输出阻抗，无需外部运放即可直接驱动外部负载。每个 DAC 的输出增益可以通过设置 DAC 控制寄存器（DAC_CTRL）的 DxOBDIS 位来使能或关闭。
- DACx 使能
  - DACx 的模拟部分由 DAC 控制寄存器（DAC_CTRL）的 DxEN 位控制开启，数字部分则不受该位控制。

3 DAC 案例

注：所有 project 都是基于 keil 5 而建立，若用户需要在其他编译环境上使用，请参考 AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates 中各种编译环境（例如 IAR6/7,keil 4/5）进行简单修改即可。

3.1 案例 1--使用 DMA 的双 DAC 正弦波输出

3.1.1 功能简介

在 TMR2 TRGO 事件启动 DAC 转换，利用 DMA 将数组中数据搬至双 DAC 的 12 位右对齐数据保持寄存器（DAC_HDR12RD），实现 PA4/PA5 输出正弦波。

3.1.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\dac\double_mode_dma_sinewave\mdk_v5

3.1.3 软件设计

1) 配置流程

- 配置时钟及 DAC 对应的 GPIO；
- 配置 TMR/DAC/DMA；

2) 代码介绍

- 时钟配置代码描述

```
/* 使能 DMA1/DAC/TMR2/GPIOA 时钟 */
crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_DAC_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

- GPIO 配置代码描述

```
/*一旦使能 DACx 通道，相应的 GPIO 引脚（PA4/PA5）就会自动与 DAC 的模拟输出相连
（DACx_OUT）。为了避免寄生的干扰和额外的功耗，引脚 PA4/PA5 在此之前设置成模拟模式 */
gpio_init_struct.gpio_pins = GPIO_PINS_4 | GPIO_PINS_5; /*选取 pin4 和 pin5*/
gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG; /*配置 GPIO 为模拟*/
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL; /*配置 GPIO 为推挽模式*/
gpio_init_struct.gpio_pull = GPIO_PULL_NONE; /*配置 GPIO 无上下拉*/
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER; /*配置 GPIO 驱动能力为强*/
gpio_init(GPIOA, &gpio_init_struct); /*按以上配置设置 PA4/PA5*/
```

■ TMR 配置代码描述

```
/* 获取系统时钟，用于配置 TMR 触发频率 */
crm_clocks_freq_get(&crm_clocks_freq_struct);
/* TMR2 初始化配置，向上计数，触发频率 10kHz== (systemclock/(systemclock/1000000))/100 */
tmr_base_init(TMR2, 100-1, (crm_clocks_freq_struct.sclk_freq/1000000 - 1));
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);
/* TMR2 主定时器输出信号选择更新 OVERFLOW */
tmr_primary_mode_select(TMR2, TMR_PRIMARY_SEL_OVERFLOW);
/*使能 TMR2*/
tmr_counter_enable(TMR2, TRUE);
```

■ DAC 配置代码描述

```
/* DAC1/DAC2 触发事件源选择 TMR2 TRGOUT 事件触发*/
dac_trigger_select(DAC1_SELECT, DAC_TMR2_TRGOUT_EVENT);
dac_trigger_select(DAC2_SELECT, DAC_TMR2_TRGOUT_EVENT);
/* DAC1/DAC2 触发使能*/
dac_trigger_enable(DAC1_SELECT, TRUE);
dac_trigger_enable(DAC2_SELECT, TRUE);
/* DAC1/DAC2 噪声/三角波生成功能关闭*/
dac_wave_generate(DAC1_SELECT, DAC_WAVE_GENERATE_NONE);
dac_wave_generate(DAC2_SELECT, DAC_WAVE_GENERATE_NONE);
/* DAC1/DAC2 输出缓存关闭*/
dac_output_buffer_enable(DAC1_SELECT, FALSE);
dac_output_buffer_enable(DAC2_SELECT, FALSE);
/* DAC1/DAC2 DMA 使能*/
dac_dma_enable(DAC1_SELECT, TRUE);
dac_dma_enable(DAC2_SELECT, TRUE);
/* DAC1/DAC2 使能*/
dac_enable(DAC1_SELECT, TRUE);
dac_enable(DAC2_SELECT, TRUE);
```

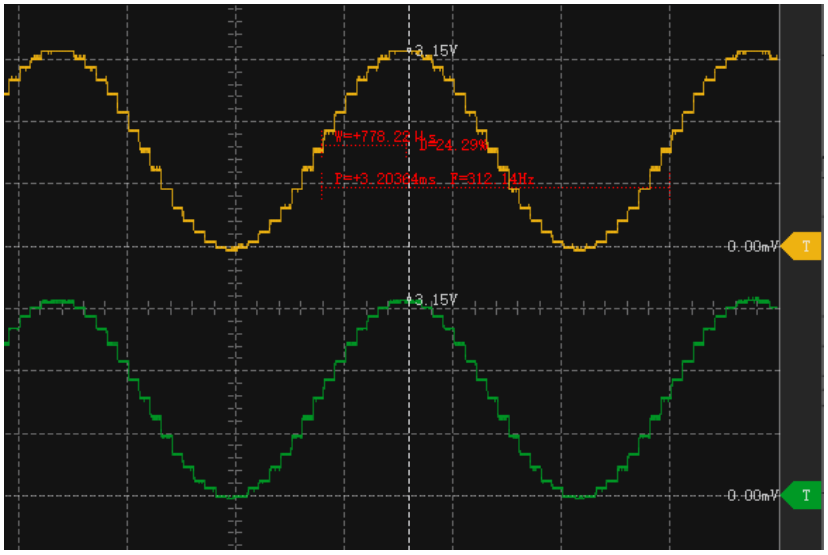
■ DMA 配置代码描述

```
/* 正弦波数组数据*/
const uint16_t sine12bit[32] = {2047, 2447, 2831, 3185, 3498, 3750, 3939, 4056,
                                4095, 4056, 3939, 3750, 3495, 3185, 2831, 2447,
                                2047, 1647, 1263, 909,  599,  344,  155,  38,
                                0,    38,   155,  344,  599,  909,  1263, 1647};
uint32_t dualsine12bit[32];
uint32_t idx = 0;
```

```
/*数组数据与双 DAC 的 12 位右对齐数据保持寄存器（DAC_HDR12RD）对齐*/
for(idx = 0; idx < 32; idx++)
{
  dualsine12bit[idx] = (sine12bit[idx] << 16) + (sine12bit[idx]);
}
/* DMA1 CHANNEL1 配置 */
dma_reset(DMA1_CHANNEL1); /*复位 DMA1 channel1，使 channel1 处于默认配置*/
dma_init_struct.buffer_size = 32; /*设置 DMA buffer 长度：和数组长度一致*/
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL; /*数据传输方向：从内存到外设*/
dma_init_struct.memory_base_addr = (uint32_t)dualsine12bit; /*内存地址*/
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_WORD; /*数据宽度 WORD*/
dma_init_struct.memory_inc_enable = TRUE; /*内存地址自增，发一个数据后，内存地址加一*/
dma_init_struct.peripheral_base_addr = (uint32_t)0x40007420; /*外设地址 DAC_HDR12RD 寄存器*/
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_WORD; /*数据宽度*/
dma_init_struct.peripheral_inc_enable = FALSE; /*外设地址自增关闭，一直是 DAC_HDR12RD*/
dma_init_struct.priority = DMA_PRIORITY_MEDIUM; /*DMA 优先级中等*/
dma_init_struct.loop_mode_enable = TRUE; /*循环模式开启*/
dma_init(DMA1_CHANNEL1, &dma_init_struct); /*将 DMA1 channel1 设置为以上配置*/
/*DMA1 CHANNEL1 弹性映射配置至 DAC2 */
dma_flexible_config(DMA1, FLEX_CHANNEL1, DMA_FLEXIBLE_DAC2);
/*使能 DMA1 CHANNEL1 */
dma_channel_enable(DMA1_CHANNEL1, TRUE);
```

3.1.4 实验效果

图 5.double_mode_dma_sinewave 实验效果



由上图可知，正弦波幅值约为 3.15V，周期约为 3.2ms（32*1/10kHz），基本符合程序设计预期。

3.2 案例 2--使用 DMA 的双 DAC 方波输出

3.2.1 功能简介

每隔 100us，软件触发启动 DAC 转换，利用 DMA 将数组中数据（0xfff 和 0x0）搬至双 DAC 的 12 位右对齐数据保持寄存器（DAC_HDR12RD），实现 PA4/PA5 输出方波。

3.2.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\dac\double_mode_dma_squarewave\mdk_v5

3.2.3 软件设计

1) 配置流程

- 配置时钟及 DAC 对应的 GPIO；
- 配置 DAC/DMA；

2) 代码介绍

- 时钟配置代码描述

```
/* 使能 DMA1/DAC/ GPIOA 时钟 */
crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_DAC_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

- GPIO 配置代码描述

```
/*一旦使能 DACx 通道，相应的 GPIO 引脚（PA4/PA5）就会自动与 DAC 的模拟输出相连
（DACx_OUT）。为了避免寄生的干扰和额外的功耗，引脚 PA4/PA5 在此之前设置成模拟模式 */
gpio_init_struct.gpio_pins = GPIO_PINS_4 | GPIO_PINS_5; /*选取 pin4 和 pin5*/
gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG; /*配置 GPIO 为模拟*/
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL; /*配置 GPIO 为推挽模式*/
gpio_init_struct.gpio_pull = GPIO_PULL_NONE; /*配置 GPIO 无上下拉*/
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER; /*配置 GPIO 驱动能力为强*/
gpio_init(GPIOA, &gpio_init_struct); /*按以上配置设置 PA4/PA5*/
```

- DAC 配置代码描述

```
/* DAC1/DAC2 触发事件源选择软件触发*/
dac_trigger_select(DAC1_SELECT, DAC_SOFTWARE_TRIGGER);
```

```
dac_trigger_select(DAC2_SELECT, DAC_SOFTWARE_TRIGGER);
/* DAC1/DAC2 触发使能*/
dac_trigger_enable(DAC1_SELECT, TRUE);
dac_trigger_enable(DAC2_SELECT, TRUE);
/* DAC1/DAC2 噪声/三角波生成功能关闭*/
dac_wave_generate(DAC1_SELECT, DAC_WAVE_GENERATE_NONE);
dac_wave_generate(DAC2_SELECT, DAC_WAVE_GENERATE_NONE);
/* DAC1/DAC2 输出缓存关闭*/
dac_output_buffer_enable(DAC1_SELECT, FALSE);
dac_output_buffer_enable(DAC2_SELECT, FALSE);
/* DAC1/DAC2 DMA 使能*/
dac_dma_enable(DAC1_SELECT, TRUE);
dac_dma_enable(DAC2_SELECT, TRUE);
/* DAC1/DAC2 使能*/
dac_enable(DAC1_SELECT, TRUE);
dac_enable(DAC2_SELECT, TRUE);
```

- DMA 配置代码描述

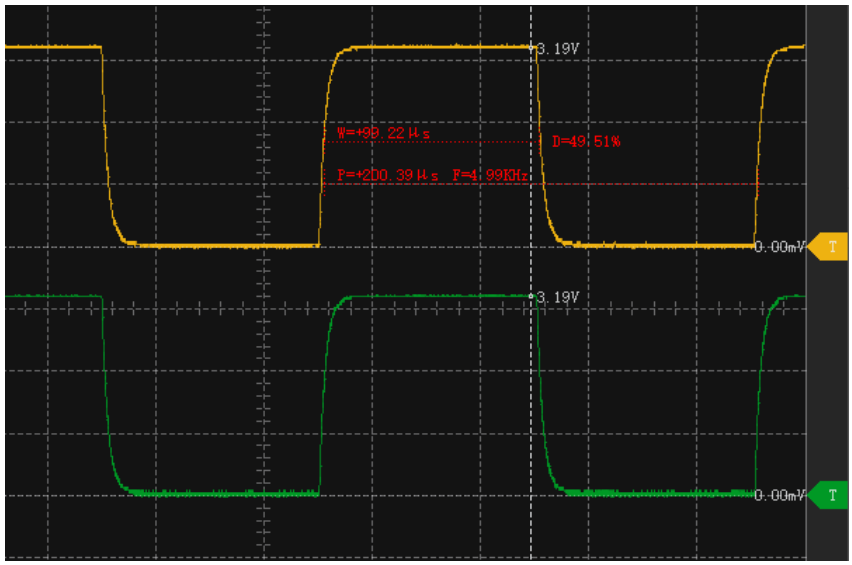
```
/* 方波数组数据，与双 DAC 的 12 位右对齐数据保持寄存器（DAC_HDR12RD）对齐*/
uint32_t dualsquare12bit[2] = {(0xfff | (0xfff << 16)), 0};
/* DMA1 CHANNEL1 配置 */
dma_reset(DMA1_CHANNEL1); /*复位 DMA1 channel1，使 channel1 处于默认配置*/
dma_init_struct.buffer_size = 2; /*设置 DMA buffer 长度：和数组长度一致*/
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL; /*数据传输方向：从内存到外设*/
dma_init_struct.memory_base_addr = (uint32_t)dualsquare12bit; /*内存地址*/
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_WORD; /*数据宽度 WORD*/
dma_init_struct.memory_inc_enable = TRUE; /*内存地址自增，发一个数据后，内存地址加一*/
dma_init_struct.peripheral_base_addr = (uint32_t)0x40007420; /*外设地址 DAC_HDR12RD 寄存器*/
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_WORD; /*数据宽度*/
dma_init_struct.peripheral_inc_enable = FALSE; /*外设地址自增关闭，一直是 DAC_HDR12RD*/
dma_init_struct.priority = DMA_PRIORITY_MEDIUM; /*DMA 优先级中等*/
dma_init_struct.loop_mode_enable = TRUE; /*循环模式开启*/
dma_init(DMA1_CHANNEL1, &dma_init_struct); /*将 DMA1 channel1 设置为以上配置*/
/*DMA1 CHANNEL1 弹性映射配置至 DAC2 */
dma_flexible_config(DMA1, FLEX_CHANNEL1, DMA_FLEXIBLE_DAC2);
/*使能 DMA1 CHANNEL1 */
dma_channel_enable(DMA1_CHANNEL1, TRUE);
```

■ DAC 软件触发使能代码描述

```
/* 每隔 100us，DAC1/DAC2 软件触发一次*/
while(1)
{
  delay_us(100);
  dac_software_trigger_generate(DAC1_SELECT);
  dac_software_trigger_generate(DAC2_SELECT);
}
```

3.2.4 实验效果

图 6.double_mode_dma_squarewave 实验效果



由上图可知，方波幅值约为 3.19V，周期约为 200us，基本符合程序设计预期。

3.3 案例 3--使用 DMA 的 DAC 梯形波输出

3.3.1 功能简介

在 TMR2 TRGO 事件启动 DAC 转换，利用 DMA 将数组中数据搬至 DAC1 的 8 位右对齐数据保持寄存器（DAC_D1DTH8R），实现 PA4 输出梯形波。

3.3.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\dac\one_dac_dma_escalator\mdk_v5

3.3.3 软件设计

1) 配置流程

- 配置时钟及 DAC 对应的 GPIO；
- 配置 TMR/DAC/DMA；

2) 代码介绍

- 时钟配置代码描述

```
/* 使能 DMA1/DAC/TMR2/GPIOA 时钟 */
crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_DAC_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

- GPIO 配置代码描述

```
/*一旦使能 DACx 通道，相应的 GPIO 引脚（PA4/PA5）就会自动与 DAC 的模拟输出相连
（DACx_OUT）。为了避免寄生的干扰和额外的功耗，引脚 PA4/PA5 在此之前设置成模拟模式 */
gpio_init_struct.gpio_pins = GPIO_PINS_4; /*选取 pin4*/
gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG; /*配置 GPIO 为模拟*/
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL; /*配置 GPIO 为推挽模式*/
gpio_init_struct.gpio_pull = GPIO_PULL_NONE; /*配置 GPIO 无上下拉*/
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER; /*配置 GPIO 驱动能力为强*/
gpio_init(GPIOA, &gpio_init_struct); /*按以上配置设置 PA4*/
```

- TMR 配置代码描述

```
/* 获取系统时钟，用于配置 TMR 触发频率 */
crm_clocks_freq_get(&crm_clocks_freq_struct);
/* TMR2 初始化配置，向上计数，触发频率 1kHz== (systemclock/(systemclock/1000000))/1000 */
tmr_base_init(TMR2, 1000-1, (crm_clocks_freq_struct.sclk_freq/1000000 - 1));
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);
/* TMR2 主定时器输出信号选择更新 OVERFLOW */
tmr_primary_mode_select(TMR2, TMR_PRIMARY_SEL_OVERFLOW);
/*使能 TMR2*/
tmr_counter_enable(TMR2, TRUE);
```

- DAC 配置代码描述

```
/* DAC1 触发事件源选择 TMR2 TRGOUT 事件触发*/
dac_trigger_select(DAC1_SELECT, DAC_TMR2_TRGOUT_EVENT);
/* DAC1 触发使能*/
```

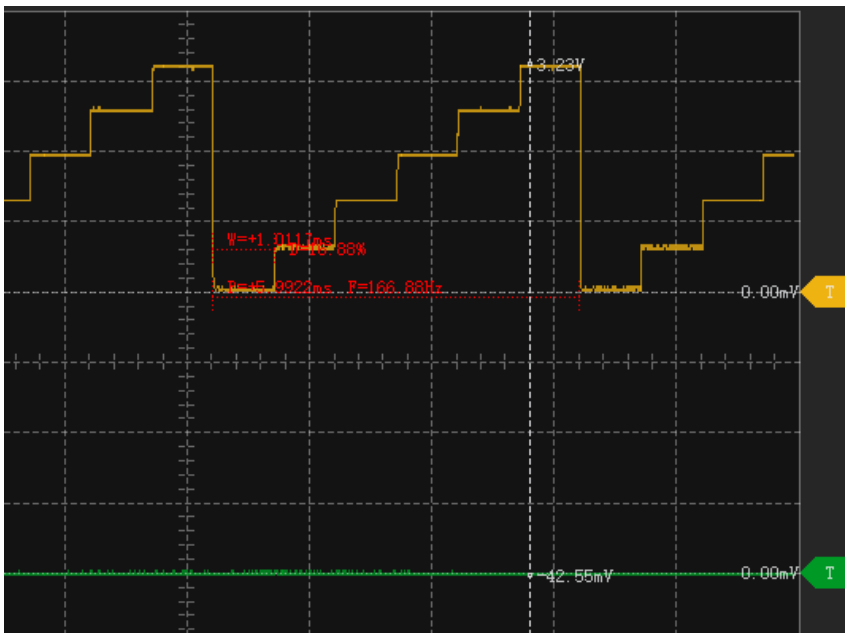
```
dac_trigger_enable(DAC1_SELECT, TRUE);
/* DAC1 噪声/三角波生成功能关闭*/
dac_wave_generate(DAC1_SELECT, DAC_WAVE_GENERATE_NONE);
/* DAC1 输出缓存关闭*/
dac_output_buffer_enable(DAC1_SELECT, FALSE);
/* DAC1 DMA 使能*/
dac_dma_enable(DAC1_SELECT, TRUE);
/* DAC1 使能*/
dac_enable(DAC1_SELECT, TRUE);
```

- DMA 配置代码描述

```
/*  梯形波数组数据*/
const uint8_t escalator8bit[6] = {0x0, 0x33, 0x66, 0x99, 0xCC, 0xFF};
/* DMA1 CHANNEL2 配置  */
dma_reset(DMA1_CHANNEL2); /*复位 DMA1 channel2，使 channel2 处于默认配置*/
dma_init_struct.buffer_size = 6; /*设置 DMA buffer 长度：和数组长度一致*/
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL; /*数据传输方向：从内存到外设*/
dma_init_struct.memory_base_addr = (uint32_t) escalator8bit; /*内存地址*/
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_ BYTE; /*数据宽度 BYTE*/
dma_init_struct.memory_inc_enable = TRUE; /*内存地址自增，发一个数据后，内存地址加一*/
dma_init_struct.peripheral_base_addr = (uint32_t)0x40007410; /*外设地址 DAC_D1DTH8R 寄存器*/
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_ BYTE; /*数据宽度*/
dma_init_struct.peripheral_inc_enable = FALSE; /*外设地址自增关闭，一直是 DAC_D1DTH8R*/
dma_init_struct.priority = DMA_PRIORITY_MEDIUM; /*DMA 优先级中等*/
dma_init_struct.loop_mode_enable = TRUE; /*循环模式开启*/
dma_init(DMA1_CHANNEL2, &dma_init_struct); /*将 DMA1 channel2 设置为以上配置*/
/*DMA1 CHANNEL2 弹性映射配置至 DAC1 */
dma_flexible_config(DMA1, FLEX_CHANNEL2, DMA_FLEXIBLE_DAC1);
/*使能 DMA1 CHANNEL1 */
dma_channel_enable(DMA1_CHANNEL2, TRUE);
```

3.3.4 实验效果

图 7. one_dac_dma_escalator 实验效果



由上图可知，梯形波幅值约为 3.23V，周期约为 6ms（6*1/1kHz），基本符合程序设计预期。

3.4 案例 4-- DAC 噪声输出

3.4.1 功能简介

软件触发启动 DAC 转换，利用线性回馈移位寄存器（LENinear Feedback Shift Register LFSR）产生幅度变化的伪噪声，透过 PA4 输出。本例中，为了生成噪声的输出观测效果连续性，DHRx 寄存器未做配置，即使用默认值 0x0（参见 2.5 噪声/三角波生成相关说明）。

3.4.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\dac\one_dac_noisewave\mdk_v5

3.4.3 软件设计

1) 配置流程

- 配置时钟及 DAC 对应的 GPIO；
- 配置 DAC；

2) 代码介绍

- 时钟配置代码描述

```
/* 使能 DAC/GPIOA 时钟 */
crm_periph_clock_enable(CRM_DAC_PERIPH_CLOCK, TRUE);
```

```
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

■ GPIO 配置代码描述

```
/*一旦使能 DACx 通道，相应的 GPIO 引脚（PA4/PA5）就会自动与 DAC 的模拟输出相连
（DACx_OUT）。为了避免寄生的干扰和额外的功耗，引脚 PA4/PA5 在此之前设置成模拟模式 */
gpio_init_struct.gpio_pins = GPIO_PINS_4; /*选取 pin4*/
gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG; /*配置 GPIO 为模拟*/
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL; /*配置 GPIO 为推挽模式*/
gpio_init_struct.gpio_pull = GPIO_PULL_NONE; /*配置 GPIO 无上下拉*/
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER; /*配置 GPIO 驱动能力为强*/
gpio_init(GPIOA, &gpio_init_struct); /*按以上配置设置 PA4*/
```

■ DAC 配置代码描述

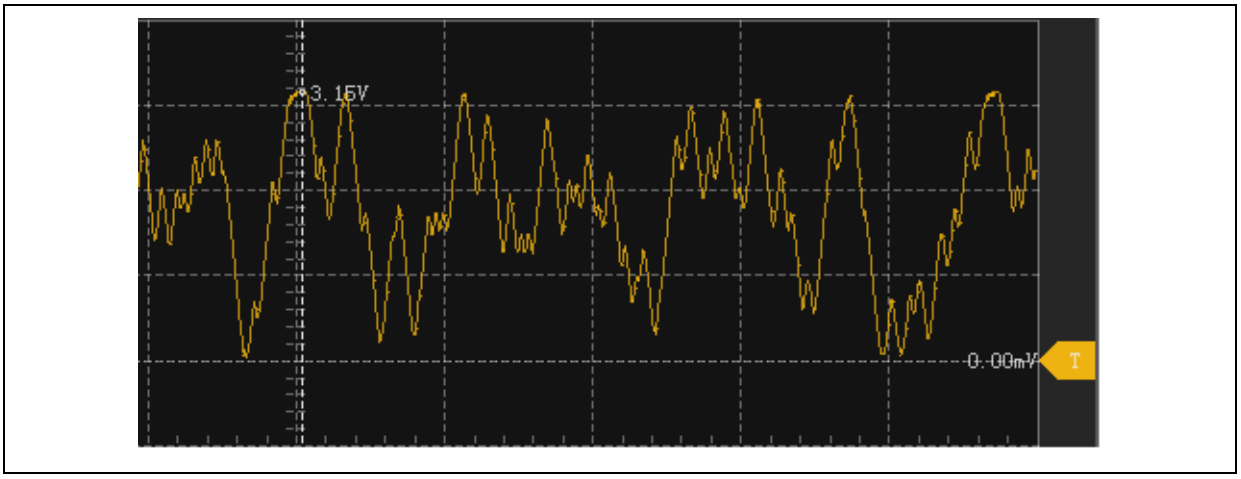
```
/* DAC1 触发事件源选择软件触发*/
dac_trigger_select(DAC1_SELECT, DAC_SOFTWARE_TRIGGER);
/* DAC1 触发使能*/
dac_trigger_enable(DAC1_SELECT, TRUE);
/* DAC1 噪声生成功能使能*/
dac_wave_generate(DAC1_SELECT, DAC_WAVE_GENERATE_NOISE);
/* DAC1 噪声生成模式选择屏蔽位不屏蔽 LSFR 位[11：0] */
dac_mask_amplitude_select(DAC1_SELECT, DAC_LSFR_BITB0_AMPLITUDE_4095);
/* DAC1 输出缓存开启，以及时响应不断产生的噪声*/
dac_output_buffer_enable(DAC1_SELECT, TRUE);
/* DAC1 使能*/
dac_enable(DAC1_SELECT, TRUE);
```

■ DAC 软件触发使能代码描述

```
/* DAC1 不断生成软件触发*/
while(1)
{
  dac_software_trigger_generate(DAC1_SELECT);
}
```

3.4.4 实验效果

图 8. one_dac_noisewave 实验效果



由上图可知，DAC1 随机生成了幅值 0-3.3V 的伪噪声，基本符合程序设计预期。

3.5 案例 5--双 DAC 三角波输出

3.5.1 功能简介

在 TMR2 TRGO 事件启动 DAC 转换，利用三角波发生器（triangle）产生三角波。本例中，DHRx 寄存器值配置为 0x100，与三角波计数器的值相加并丢弃溢出位后写入 DAC1 数据输出寄存器（DAC_D1ODT）（参见 2.5 噪声/三角波生成相关说明），透过 PA4/PA5 输出。

3.5.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\dac\two_dac_trianglewave\mdk_v5

3.5.3 软件设计

1) 配置流程

- 配置时钟及 DAC 对应的 GPIO；
- 配置 TMR/DAC；

2) 代码介绍

- 时钟配置代码描述

```
/* 使能 DMA1/DAC/TMR2/GPIOA 时钟 */
crm_periph_clock_enable(CRM_DAC_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

- GPIO 配置代码描述

```
/*一旦使能 DACx 通道，相应的 GPIO 引脚（PA4/PA5）就会自动与 DAC 的模拟输出相连
（DACx_OUT）。为了避免寄生的干扰和额外的功耗，引脚 PA4/PA5 在此之前设置成模拟模式 */
gpio_init_struct.gpio_pins = GPIO_PINS_4 | GPIO_PINS_5; /*选取 pin4 和 pin5*/
gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG; /*配置 GPIO 为模拟*/
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL; /*配置 GPIO 为推挽模式*/
gpio_init_struct.gpio_pull = GPIO_PULL_NONE; /*配置 GPIO 无上下拉*/
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER; /*配置 GPIO 驱动能力为强*/
gpio_init(GPIOA, &gpio_init_struct); /*按以上配置设置 PA4/PA5*/
```

■ TMR 配置代码描述

```
/* 获取系统时钟，用于配置 TMR 触发频率 */
crm_clocks_freq_get(&crm_clocks_freq_struct);
/* TMR2 初始化配置，向上计数，触发频率 10kHz== (systemclock/(systemclock/1000000))/100 */
tmr_base_init(TMR2, 100-1, (crm_clocks_freq_struct.sclk_freq/1000000 - 1));
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);
/* TMR2 主定时器输出信号选择更新 OVERFLOW */
tmr_primary_mode_select(TMR2, TMR_PRIMARY_SEL_OVERFLOW);
/*使能 TMR2*/
tmr_counter_enable(TMR2, TRUE);
```

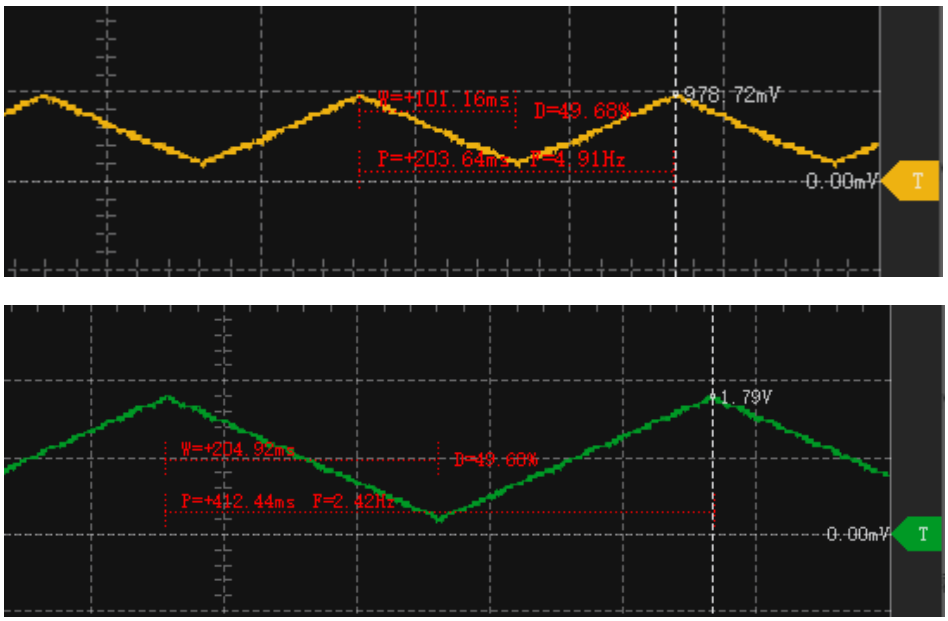
■ DAC 配置代码描述

```
/* DAC1/DAC2 触发事件源选择 TMR2 TRGOUT 事件触发*/
dac_trigger_select(DAC1_SELECT, DAC_TMR2_TRGOUT_EVENT);
dac_trigger_select(DAC2_SELECT, DAC_TMR2_TRGOUT_EVENT);
/* DAC1/DAC2 触发使能*/
dac_trigger_enable(DAC1_SELECT, TRUE);
dac_trigger_enable(DAC2_SELECT, TRUE);
/* DAC1/DAC2 三角波生成功能使能*/
dac_wave_generate(DAC1_SELECT, DAC_WAVE_GENERATE_TRIANGLE);
dac_wave_generate(DAC2_SELECT, DAC_WAVE_GENERATE_TRIANGLE);
/* DAC1 三角波生成模式选择波形的幅值为 1023，换算结果为 3.3V*1023/4095=824.4mV*/
dac_mask_amplitude_select(DAC1_SELECT, DAC_LSFR_BIT90_AMPLITUDE_1023);
/* DAC2 三角波生成模式选择波形的幅值为 2047，换算结果为 3.3V*2047/4095=1649.6mV */
dac_mask_amplitude_select(DAC2_SELECT, DAC_LSFR_BITA0_AMPLITUDE_2047);
/* DAC1/DAC2 输出缓存关闭*/
dac_output_buffer_enable(DAC1_SELECT, FALSE);
dac_output_buffer_enable(DAC2_SELECT, FALSE);
```

```
/* DAC1/DAC2 使能*/
dac_enable(DAC1_SELECT, TRUE);
dac_enable(DAC2_SELECT, TRUE);
/* DHRx 寄存器值配置为 0x100，换算结果为 3.3V*0x100/0xfff=206.3mV*/
dac_dual_data_set(DAC_DUAL_12BIT_RIGHT, 0x100, 0x100);
```

3.5.4 实验效果

图 9. two_dac_trianglewave 实验效果



由上图可知，黄色 DAC1 输出三角波幅值约为 978.72mV（206.3mV+824.4mV），周期约为 204.8ms（1024*2*1/10kHz），绿色 DAC2 输出三角波幅值约为 1.79V（206.3mV+1649.6mV），周期约为 409.6ms（2048*2*1/10kHz），基本符合程序设计预期。

4 文档版本历史

表 3.文档版本历史

日期	版本	变更
2022.01.10	2.0.0	最初版本
2023.03.15	2.0.1	更新AT32F423支持，优化部分排版

重要通知-请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。