AN1801
应用笔记

UM324xF DAC 应用手册

版本：V1.0

广芯微电子（广州）股份有限公司

http://www.unicmicro.com

版本修订

版本	日期	描述
V1.0	2022.11.28	初始版

目录

1 DAC 简介

DAC 可以控制数模转换器将 12 位数字量转换为模拟电压输出。支持 8/12 位数字输入，支持两个转换器同时更新转换数据，支持产生三角波和噪声波，支持 DMA 操作，支持外部触发更新转换数据。

可配合《UM324xF 用户手册》和“../Driver&Example/Example/DAC”中的代码用例进行学习。

2 DAC 软件配置

2.1 DAC 宏定义

1. **DAC 通道定义：**

```
#define DAC_CHANNEL_0                   0x0
#define DAC_CHANNEL_1                   0x1
```

从上到下的定义为通道 0 和通道 1。

2. **DAC 使能和失能定义：**

```
#define DAC_DISABLE                     0
#define DAC_ENABLE                      1
```

从上到下的定义为 DAC 失能和 DAC 使能。

3. **DAC 参考电压定义：**

```
#define DAC_REFVOL_VDDA                 1
#define DAC_REFVOL_VREFIN               2
#define DAC_REFVOL_VREFOUT              3
```

从上到下参考电压的定义为 VDDA、VREFIN、VREFOUT。

4. **DAC 触发源定义：**

```
#define DAC_TRIG_TIM5TRGO               0
#define DAC_TRIG_TIM7TRGO               1
#define DAC_TRIG_TIM6TRGO               2
#define DAC_TRIG_TIM4TRGO               3
#define DAC_TRIG_TIM1TRGO               4
#define DAC_TRIG_TIM3TRGO               5
#define DAC_TRIG_EXTINTPIN              6
#define DAC_TRIG_SOFTWARE               7
```

从上到下参考触发源的定义为 TIM5、TIM7、TIM6、TIM4、TIM1、TIM3、EXTIx 和软件触发。

5. **DAC 触发使能和失能定义：**

```
#define DAC_TRIGMODE_DISABLE            DAC_DISABLE
#define DAC_TRIGMODE_ENABLE             DAC_ENABLE
```

从上到下的定义为 DAC 失能触发和 DAC 使能触发。

6. DAC 输出信号类型定义：

```
#define DAC_WAVE_NONE        0
#define DAC_WAVE_NOISE       1
#define DAC_WAVE_TRIANGLE    2
```

从上到下的定义为不产生噪声波或三角波、产生噪声波和产生三角波。

7. DAC 噪声模式下的掩码/三角波模式下的振幅：

```
#define DAC_MASKSORAMPLITUDES_0      0
#define DAC_MASKSORAMPLITUDES_3      1
#define DAC_MASKSORAMPLITUDES_7      2
#define DAC_MASKSORAMPLITUDES_15     3
#define DAC_MASKSORAMPLITUDES_31     4
#define DAC_MASKSORAMPLITUDES_63     5
#define DAC_MASKSORAMPLITUDES_127    6
#define DAC_MASKSORAMPLITUDES_255    7
#define DAC_MASKSORAMPLITUDES_511    8
#define DAC_MASKSORAMPLITUDES_1023   9
#define DAC_MASKSORAMPLITUDES_2047   10
#define DAC_MASKSORAMPLITUDES_4095   11
```

从上到下的定义为输出幅值的有效值范围，即 0-x，其中 x 为 0、3、7、15、31、63、127、255、511、1023、2047 和 4095。

8. DAC Buffer 使能和失能定义：

```
#define DAC_BUFFER_DISABLE    DAC_DISABLE
#define DAC_BUFFER_ENABLE     DAC_ENABLE
```

从上到下的定义为 Buffer 失能和 Buffer 使能。

9. DAC 数据类型定义：

```
#define DAC_DATATYPE_RIGHT_12    0
#define DAC_DATATYPE_LEFT_12     1
#define DAC_DATATYPE_RIGHT_8     2
```

从上到下的定义为 12 位右对齐、12 位左对齐和 8 位右对齐。

2.2 直接输出

DAC 初始化流程如下，下面将以 DAC 通道 0 为例 ：

1. 调用 gpio.h 中的 gpio_set_mux_mode 接口，将 PA4（DAC 通道 0）配置为模拟接口；
2. 调用 dac.h 中的 dac_clock_init 接口使能 DAC 时钟；
3. 调用 dac.h 中的 dac_init 接口，配置输出通道，选择参考电压源，不使能触发，不产生噪声波和三角波，不使能 Buffer，其中触发源选择和噪声模式下的掩码/三角波模式下的振幅选择设置无效；
4. 调用 dac.h 中的 dac_set_channel_data 接口，设置数据类型，此处设置为 12 位右对齐，写入输出电压对应的数据，例如参考电压为 3.3V，若要输出 1.65V 电压，则写入的数据为（1.65/3.3）*4095≈2048。
5. 调用 dac.h 中的 dac_enable_config 接口，使能 DAC 通道 0 输出；

```
gpio_clk_init(GPIOA_HANDLE, GPIO_ENABLE);
gpio_set_mux_mode(GPIOA_HANDLE, PIN4, GPIO_MODE_ANALOG, NULL);//GPIOA4 -> DAC0
dac_clock_init(DAC_HANDLE, 0);
dac_init(DAC_HANDLE, DAC_CHANNEL_0, //通道 1
                        DAC_REFVOL_VDDA, //参考电压：VDDA
                        DAC_TRIG_SOFTWARE, //触发源：无效
                        DAC_TRIGMODE_DISABLE, //失能触发
                        DAC_WAVE_NONE, //输出波形：不产生噪声波/三角波
                        DAC_MASKSORAMPLITUDES_0, //噪声模式下的掩码/三角波模式下的振幅：无效
                        DAC_BUFFER_DISABLE); //输出缓冲器：不使能
dac_set_channel_data(DAC_HANDLE, DAC_CHANNEL_0, DAC_DATATYPE_RIGHT_12, 2048);
dac_enable_config(DAC_HANDLE, DAC_CHANNEL_0, DAC_ENABLE);
```

2.3 硬件触发输出

DAC 初始化流程如下，下面将以 TIM1 定时触发 DAC 通道 1 为例 ：

1. 调用 gtimer.h 中的 gtim_clock_init 接口使能 TIM1 时钟；
2. 调用 gtimer.h 中的 gtim_init 接口，配置自动重载值 arr 和预分频系数 psc，选择计数方向为向上计数，选择计数对齐方式为边沿对齐；

3. 调用 gtimer.h 中的 gtim_master_trgo_config 接口，配置 TRGO 由 UPDATE 更新事件产生；
4. 调用 gtimer.h 中的 gtim_enable_config 接口，使能 TIM1 计数器；
5. 调用 gpio.h 中的 gpio_set_mux_mode 接口，将 PA5（DAC 通道 1）配置为模拟接口；
6. 调用 dac.h 中的 dac_clock_init 接口使能 DAC 时钟；
7. 调用 dac.h 中的 dac_init 接口，配置输出通道，选择参考电压源，选择触发源为 TIM1，使能触发，不产生噪声波和三角波，不使能 Buffer，其中噪声模式下的掩码/三角波模式下的振幅选择设置无效；
8. 调用 dac.h 中的 dac_set_channel_data 接口，设置数据类型，此处设置为 12 位右对齐，写入输出电压对应的数据，例如参考电压为 3.3V，若要输出 1.65V 电压，则写入的数据为（1.65/3.3）*4095≈2048。
9. 调用 dac.h 中的 dac_enable_config 接口，使能 DAC 通道 0 输出；

```
gpio_clk_init(GPIOA_HANDLE, GPIO_ENABLE);
gpio_set_mux_mode(GPIOA_HANDLE, PIN5, GPIO_MODE_ANALOG, NULL);//GPIOA5 -> DAC1
/******************TIM1 COUNT*******************/
gtim_clock_init(TIM1_HANDLE, GTIM_ENABLE);
gtim_init(TIM1_HANDLE, 168000, 1000, GTIM_COUNTER_DIRECTION_UP, GTIM_COUNTER_ALIGNMENT_EDGE);
gtim_master_trgo_config(TIM1_HANDLE, GTIM_TRGO_UPDATE);
gtim_irq_init(TIM1_HANDLE, GTIM_IRQ_UPDATE, GTIM_ENABLE, gtim_counter_pro);
gtim_enable_config(TIM1_HANDLE, GTIM_ENABLE);

dac_clock_init(DAC_HANDLE, 0);
dac_init(DAC_HANDLE, DAC_CHANNEL_1, //通道 1
                                    DAC_REFVOL_VDDA, //参考电压：VDDA
                                    DAC_TRIG_TIM1TRGO, //触发源：TIM1
                                    DAC_TRIGMODE_ENABLE, //使能触发，写入 DACDHR 的数据在触发后的 3 个（硬件触发）/1 个（软件触发）周期后开始转换输出
                                    DAC_WAVE_NONE, //输出波形：不产生噪声波/三角波
                                    DAC_MASKSORAMPLITUDES_0, //噪声模式下的掩码/三角波模式下的振幅：无效
                                    DAC_BUFFER_DISABLE); //输出缓冲器：不使能
dac_set_channel_data(DAC_HANDLE, DAC_CHANNEL_1, DAC_DATATYPE_RIGHT_12, 2048);
dac_enable_config(DAC_HANDLE, DAC_CHANNEL_1, DAC_ENABLE);
```

其中 TIM1 的回调函数实现如下：

```
    void gtim_counter_pro(void)
{
    printfS("tim1\r\n");
    dac_set_channel_data(DAC_HANDLE, DAC_CHANNEL_1,
DAC_DATATYPE_RIGHT_12, 2048);

    gtim_clr_status(TIM1_HANDLE, GTIM_STATUS_UPDATE);
    gtim_enable_config(TIM1_HANDLE, GTIM_DISABLE);
}
```

可根据实际应用场景，在回调函数中更新 DAC 输出。

2.4 DAC DMA 传输

DAC 初始化流程如下，下面将以 TIM1 定时触发 DMA 传输更新 DAC 通道 0 为例 ：

1. 调用 gtimer.h 中的 gtim_clock_init 接口使能 TIM1 时钟。
2. 调用 gtimer.h 中的 gtim_init 接口，配置自动重载值 arr 和预分频系数 psc，选择计数方向为向上计数，选择计数对齐方式为边沿对齐。
3. 调用 gtimer.h 中的 gtim_master_trgo_config 接口，配置 TRGO 由 UPDATE 更新事件产生。
4. 调用 gtimer.h 中的 gtim_enable_config 接口，使能 TIM1 计数器。
5. 调用 gpio.h 中的 gpio_set_mux_mode 接口，将 PA4（DAC 通道 0）配置为模拟接口。
6. 调用 dac.h 中的 dac_clock_init 接口使能 DAC 时钟。
7. 调用 dac.h 中的 dac_init 接口，配置输出通道，选择参考电压源，选择触发源为 TIM1，使能触发，不产生噪声波和三角波，不使能 Buffer，其中噪声模式下的掩码/三角波模式下的振幅选择设置无效。
8. 调用 dac.h 中的 dac_set_channel_data 接口，设置数据类型，此处设置为 12 位右对齐，写入输出电压对应的数据，例如参考电压为 3.3V，若要输出 1.65V 电压，则写入的数据为（1.65/3.3）*4095≈2048。
9. 调用 dac.h 中的 dac_irq_init 接口使能 DAC 中断，用于监测 DMA 请求响应情况。
10. 调用 dac.h 中的 dac_enable_config 接口，使能 DAC 通道 0 输出。
11. 调用 dma.h 中的 dma_init 接口，使能 DMA 时钟。
12. 调用 dma.h 中的 dma_tt_fc_inc_config 接口，配置 DMA 传输方向为内存到外设，源地址和目的地址均不递增。
13. 调用 dma.h 中的 dma_msize_config 接口，设置源和目的的 Burst 传输长度为 1。
14. 调用 dma.h 中的 dma_tr_width_config 接口，设置源和目的的数据宽度为 32bit。

15. 调用 dma.h 中的 dma_hs_sel_config 接口，设置源和目的为硬件握手。
16. 调用 dma.h 中的 dma_per_config 接口，由于传输方向为内存到外设，因此配置目的握手信号 5，源握手信号无效。
17. 调用 dma.h 中的 dma_dma_en_config 接口，使能 DMA 控制器。
18. 调用 dma.h 中的 dma_poll_transfer 接口，设置源地址为 SRAM 中的某一个内存地址，设置目的地址为 DAC0 右对齐 12 位数据寄存器（DAC_DHR12R0）地址，传输 Block 大小设为 1。

```
*(volatile uint32_t *)(0x20000100) = 0x000002ff;

gpio_clk_init(GPIOA_HANDLE, GPIO_ENABLE);
gpio_set_mux_mode(GPIOA_HANDLE, PIN4, GPIO_MODE_ANALOG, NULL);//GPIOA4 ->
DAC0

/******************TIM1 COUNT******************/
gtim_clock_init(TIM1_HANDLE, GTIM_ENABLE);
gtim_init(TIM1_HANDLE, 168000, 1000, GTIM_COUNTER_DIRECTION_UP,
GTIM_COUNTER_ALIGNMENT_EDGE);
gtim_master_trgo_config(TIM1_HANDLE, GTIM_TRGO_UPDATE);
gtim_irq_init(TIM1_HANDLE, GTIM_IRQ_UPDATE, GTIM_ENABLE, gtim_counter_pro);
gtim_enable_config(TIM1_HANDLE, GTIM_ENABLE);

/******************DAC******************/
dac_clock_init(DAC_HANDLE, 2048);
dac_init(DAC_HANDLE, DAC_CHANNEL_0, //通道 0
                                    DAC_REFVOL_VDDA, //参考电压：VDDA
                                    DAC_TRIG_TIM1TRGO, //触发源：TIM1TRGO
                                    DAC_TRIGMODE_ENABLE, //使能触发，写入 DACDHR
的数据在触发后的 3 个（硬件触发）/1 个（软件触发）周期后开始转换输出
                                    DAC_WAVE_NONE, //输出波形：不产生噪声波/三角波
                                    DAC_MASKSORAMPLITUDES_0, //噪声模式下的掩码/
三角波模式下的振幅：无效
                                    DAC_BUFFER_DISABLE); //输出缓冲器：不使能
dac_set_channel_data(DAC_HANDLE, DAC_CHANNEL_0, DAC_DATATYPE_RIGHT_12,
0);
dac_irq_init(DAC_HANDLE, DAC_CHANNEL_0, DAC_ENABLE, dac_dma_pro);//若两次
DMA 请求未响应，则产生该中断
dac_enable_config(DAC_HANDLE, DAC_CHANNEL_0, DAC_ENABLE);
```

```
delay_ms(10);

/*******************DMA*******************/
dma_init(DMA0_HANDLE, DMA_ENABLE);
dma_tt_fc_inc_config(DMA0_HANDLE, DMA_Channel_0, DMA_MEM_TO_PERIP,
DMA_SINC_NOC, DMA_DINC_NOC);
dma_msize_config(DMA0_HANDLE, DMA_Channel_0, DMA_SRC_MSIZE_1,
DMA_SRC_MSIZE_1);
dma_tr_width_config(DMA0_HANDLE, DMA_Channel_0, DMA_SRC_TR_WIDTH_32,
DMA_DST_TR_WIDTH_32);
dma_hs_sel_config(DMA0_HANDLE, DMA_Channel_0, DMA_HW_HS_SRC,
DMA_HW_HS_DST);
dma_per_config(DMA0_HANDLE, DMA_Channel_0,
DMA0_SRC_PER_HANDSHAKING_SIGNAL5,
DMA0_DEST_PER_HANDSHAKING_SIGNAL5);
dma_dma_en_config(DMA0_HANDLE, 1);

dma_poll_transfer(DMA0_HANDLE, DMA_Channel_0, (uint32_t)(0x20000100),
(uint32_t)&(DAC_HANDLE->DHR12R0), 1);
```

2.5 三角波输出

DAC 初始化流程如下，下面将以 DAC 通道 0 为例 ：

1. 调用 gpio.h 中的 gpio_set_mux_mode 接口，将 PA4（DAC 通道 0）配置为模拟接口。
2. 调用 dac.h 中的 dac_clock_init 接口使能 DAC 时钟。
3. 调用 dac.h 中的 dac_init 接口，配置输出通道，选择参考电压源，触发源选择软件触发，使能触发，产生三角波，三角波模式下的振幅为 4095，不使能 Buffer。
4. 调用 dac.h 中的 dac_set_channel_data 接口，设置数据类型，此处设置为 12 位右对齐，写入输出电压对应的数据，此处设为 0，即三角波信号在第一次触发时，将由 0 开始输出，随后的每次触发 DAC 按规律输出三角波信号。
5. 调用 dac.h 中的 dac_enable_config 接口，使能 DAC 通道 0 输出。

```
gpio_clk_init(GPIOA_HANDLE, GPIO_ENABLE);
gpio_set_mux_mode(GPIOA_HANDLE, PIN4, GPIO_MODE_ANALOG, NULL);//GPIOA4 ->
DAC0

dac_clock_init(DAC_HANDLE, 0);
```

```
dac_init(DAC_HANDLE, DAC_CHANNEL_0, //通道 0
                                    DAC_REFVOL_VDDA, //参考电压：VDDA
                                    DAC_TRIG_SOFTWARE, //触发源：软件触发
                                    DAC_TRIGMODE_ENABLE, //使能触发，写入 DACDHR
的数据在触发后的 3 个（硬件触发）/1 个（软件触发）周期后开始转换输出
                                    DAC_WAVE_TRIANGLE, //输出波形：产生三角波
                                DAC_MASKSORAMPLITUDES_4095, //噪声模式下的掩码/三角波
模式下的振幅：4095
                                    DAC_BUFFER_DISABLE); //输出缓冲器：不使能
dac_set_channel_data(DAC_HANDLE, DAC_CHANNEL_0, DAC_DATATYPE_RIGHT_12,
0);
dac_enable_config(DAC_HANDLE, DAC_CHANNEL_0, DAC_ENABLE);
delay_ms(10);
while(1)
{
    //每次触发改变输出
    dac_software_trig(DAC_HANDLE, DAC_CHANNEL_0);
}
```

2.6 噪声输出

DAC 初始化流程如下，下面将以 DAC 通道 0 为例 ：

1. 调用 gpio.h 中的 gpio_set_mux_mode 接口，将 PA4（DAC 通道 0）配置为模拟接口；
2. 调用 dac.h 中的 dac_clock_init 接口使能 DAC 时钟；
3. 调用 dac.h 中的 dac_init 接口，配置输出通道，选择参考电压源，触发源选择软件触发，使能触发，产生噪声波，噪声模式下的掩码为 4095，不使能 Buffer,；
4. 调用 dac.h 中的 dac_set_channel_data 接口，设置数据类型，此处设置为 12 位右对齐，写入输出电压对应的数据，此处设为 0，即噪声波信号在第一次触发时，将由 0 开始输出，随后的每次触发 DAC 按规律输出噪声波信号。
5. 调用 dac.h 中的 dac_enable_config 接口，使能 DAC 通道 0 输出；

```
gpio_clk_init(GPIOA_HANDLE, GPIO_ENABLE);
gpio_set_mux_mode(GPIOA_HANDLE, PIN4, GPIO_MODE_ANALOG, NULL);//GPIOA4 ->
DAC0

dac_clock_init(DAC_HANDLE, 0);
```

```
dac_init(DAC_HANDLE, DAC_CHANNEL_0, //通道 0
                                    DAC_REFVOL_VDDA, //参考电压：VDDA
                                    DAC_TRIG_SOFTWARE, //触发源：软件触发
                                    DAC_TRIGMODE_ENABLE, //使能触发，写入 DACDHR
的数据在触发后的 3 个（硬件触发）/1 个（软件触发）周期后开始转换输出
                                    DAC_WAVE_NOISE, //输出波形：产生噪声波
                DAC_MASKSORAMPLITUDES_4095, //噪声模式下的掩码/三角波模式下的振
幅：4095
                                    DAC_BUFFER_DISABLE); //输出缓冲器：不使能
dac_set_channel_data(DAC_HANDLE, DAC_CHANNEL_0, DAC_DATATYPE_RIGHT_12,
0);
dac_enable_config(DAC_HANDLE, DAC_CHANNEL_0, DAC_ENABLE);
delay_ms(10); //等待 DAC 上电
while(1)
{
    //每次触发改变输出
    dac_software_trig(DAC_HANDLE, DAC_CHANNEL_0);
}
```