AN0070
应用笔记

AT32F421 CMP 使用指南

前言

这篇应用笔记描述了怎么使用AT32F421xx的比较器(CMP)。AT32F421系列内置一个超低功耗比较器CMP，它可用作独立器件（I/O上提供了全部接口），也可以与定时器结合使用。

参考资料：

- AT32F421_Firmware_Library_V2.x.x\project\at_start_f421\examples\cmp
- RM_AT32F421 文档的比较器 CMP 章节

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	AT32F421 系列

目录

表目录

图目录

1 CMP 特性

- 比较器迟滞程度可配
- 定时器输出作为比较器消隐源
- 比较器输出极性可配
- 比较器输出速度可配
- 比较器同相和反相输入源可选：
  - I/O引脚
  - 内部参考电压和三个系数分压值(1/4，1/2，3/4)
- 支持输出重定向功能：
  - 普通I/O
  - 定时器断路输入TMRx_BRK
  - 定时器输入捕获TMR_CH
  - 定时器输出比较参考值清零TMR_CH_CLR
- 结合EXINT产生中断，从低功耗模式唤醒

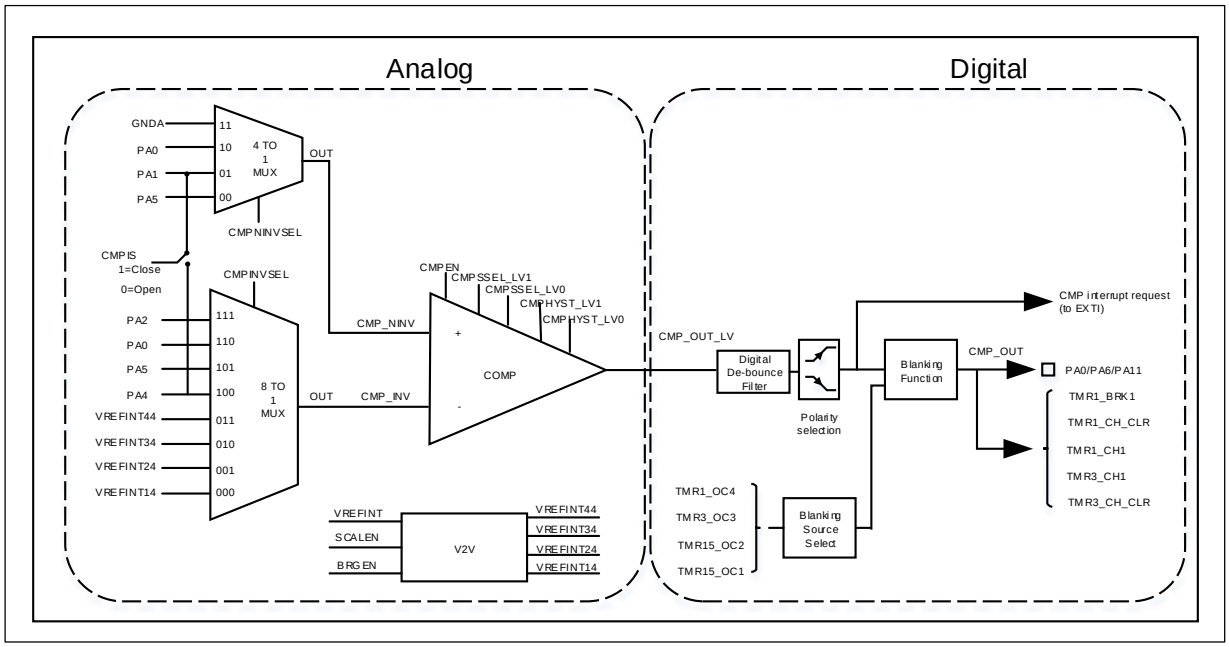
CMP可用于多种功能，包括：

- 由模拟信号触发从低功耗模式唤醒
- 模拟信号调节
- 与定时器的PWM输出结合使用时，组成逐周期的电流控制环路

2 CMP 功能介绍

2.1 CMP 功能框图

图 1. CMP 功能框图



2.2 迟滞

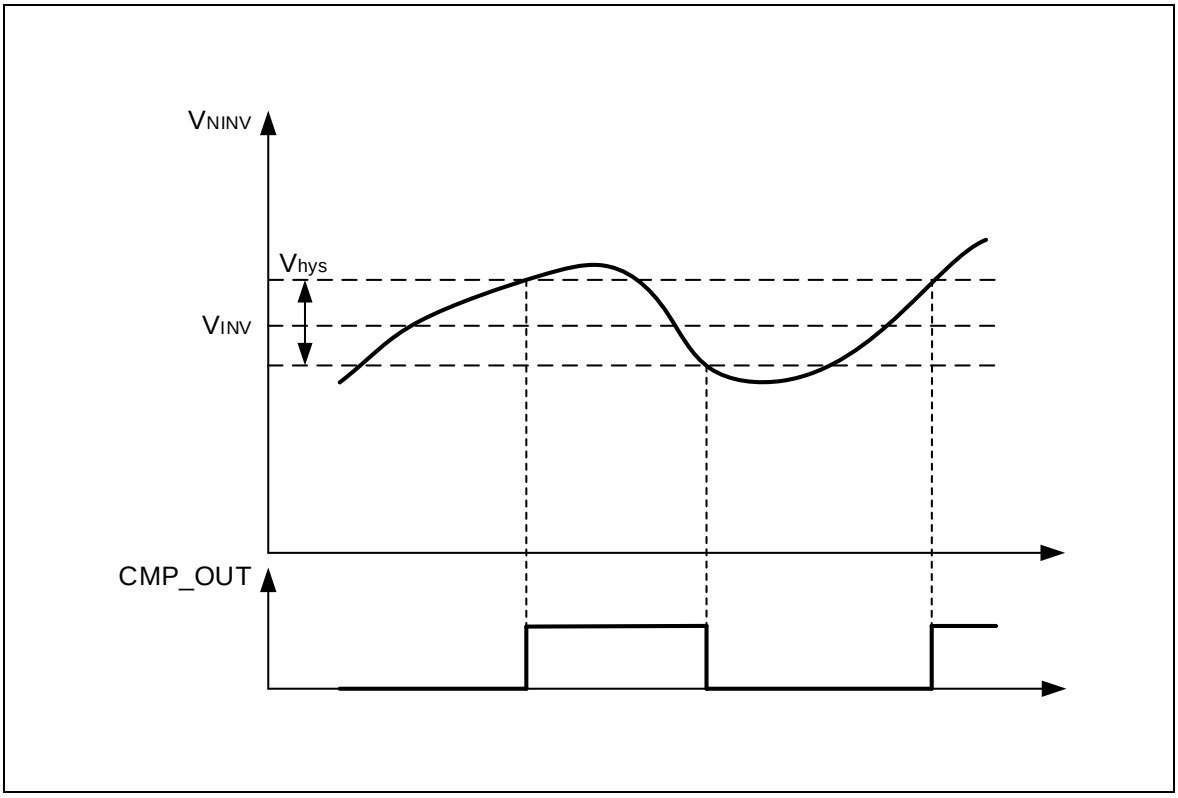
寄存器CMP_CTRLSTS的CMPHYST[1：0]控制比较器迟滞输出，该功能可避开噪声信号带来的虚假传输信号，如果不需要迟滞，可以关闭掉。比较器迟滞特性的典型值如下：

表 1. 迟滞电压典型值

符号	参数	条件	最小值(1)	典型值	最大值(1)	单位
V_{hys}	迟滞电压	无迟滞	-	0	1	mV
		低迟滞	5	8	17	
		中迟滞	10	18	37	
		高迟滞	18	38	70	

(1) 由综合评估保证，不在生产中测试。

图 2. 比较器迟滞



2.3 速率/功耗配置

通过对比较器的比较速率和功耗进行配置，可以调整比较器的传播延迟。当比较速率越低（传播延迟加大）相应的功耗就越低。用户可根据自己的需求进行配置，以平衡速率和功耗。比较器速率/功耗配置的典型值如下：

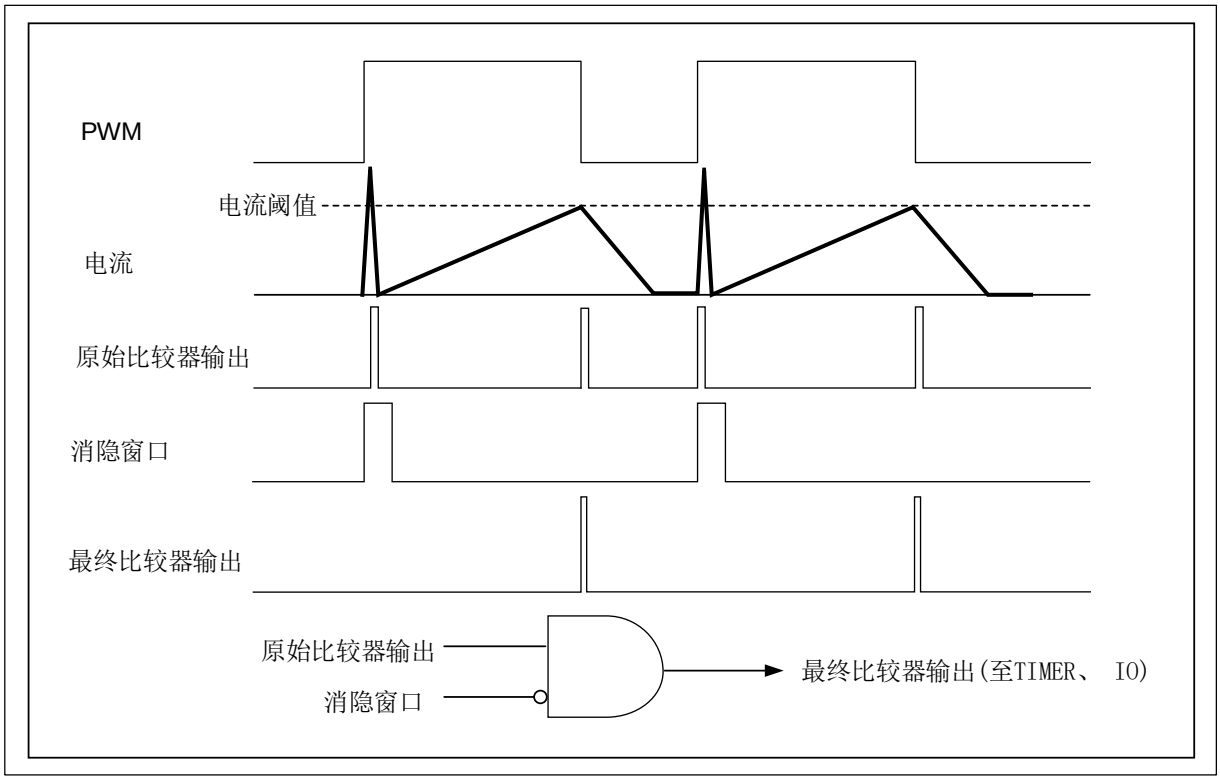
表 2. 传播延迟典型值

符号	参数	条件	最小值(1)	典型值	最大值(1)	单位
t_{START}	启动时间	高速模式	-	1.0	3.5	μs
		中速模式	-	2.8	5	
		低功耗模式	-	8	13	
		超低功耗模式	-	12	18	
t_D	200 mV步进，100 mV超载的传播延迟	高速模式	-	40	100	ns
		中速模式	-	240	320	
		低功耗模式	-	500	820	
		超低功耗模式	-	800	1800	

2.4 输出消隐功能

寄存器CMP_CTRLSTS的CMPBLANKING[2：0]位用于选择比较器消隐窗口的来源，该功能可以用于防止电流调节在PWM起始时刻产生的尖峰电流。

图 3. 比较器输出消隐

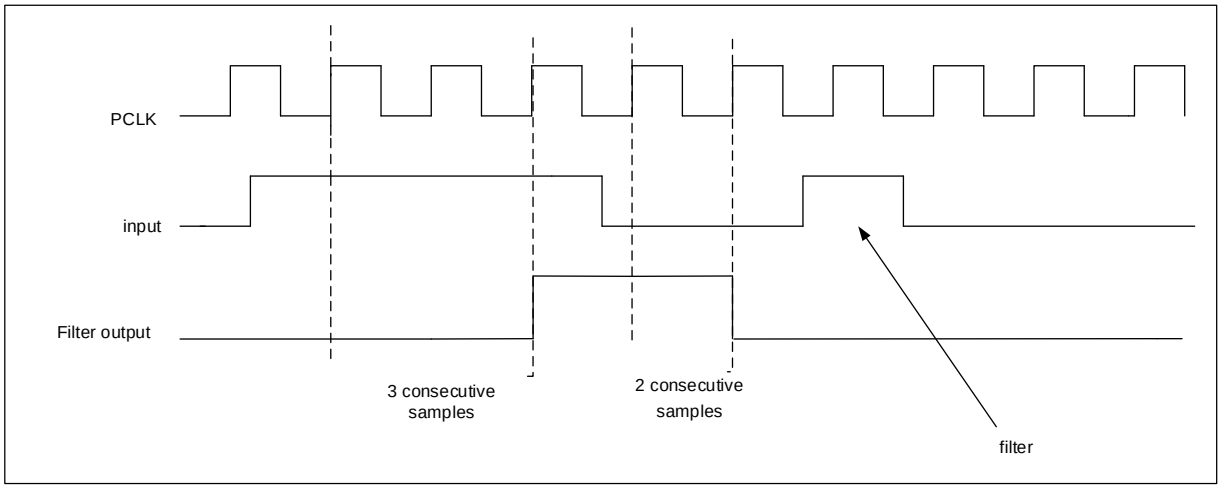


2.5 干扰滤波器

干扰滤波器可以用来滤除毛刺和噪声干扰。

滤波器的敏感性由 H_PULSE_CNT 和 L_PULSE_CNT 位控制。滤波器的敏感性会影响相同的连续采样的数量，在滤波器输入上检测到此类连续采样时，才能将某信号电平变化视为有效切换。例如 H_PULSE_CNT=2 和 L_PULSE_CNT =1 时干扰滤波器时序图，如下：

图 4. 干扰滤波器时序图



注：因为滤波器采样数据需要时钟，系统在深度睡眠模式下关闭比较器时钟，因此，要让比较器在深度睡眠模式下工作，必须在进入深度睡眠模式前关闭滤波器（G_FILTER_EN 寄存器中的GFE=0）。

2.6 CMP 中断

比较器输出从内部连接到扩展中断和事件控制器，能够产生中断或事件。该机制还可以用于退出低功耗模式。

比较器通过EXINT线21来产生中断或事件。

3 应用实例

下面介绍了比较器的四个应用实例，分别是：

- 逐周期电流控制
- 输出消隐功能
- 干扰滤波功能
- 深度睡眠模式唤醒

实例演示了比较器的用途，并介绍了它们与定时器等外设联合工作的方式，为了方便用户快速入门使用AT32F421xx的比较器，本文档介绍的应用实例的工程代码可以在BSP软件包的AT32F421_Firmware_Library_V2.x.x\project\at_start_f421\examples\cmp中获取到。

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 逐周期电流控制

3.1.1 功能简介

将比较器输出重定向至定时器可以实现逐周期的电流控制功能，其原理是通过对电路中的电流值进行比较，产生比较输出后接入定时器来控制PWM输出，从而抑制电路中的电流过大。一种可能的实现方式如下：

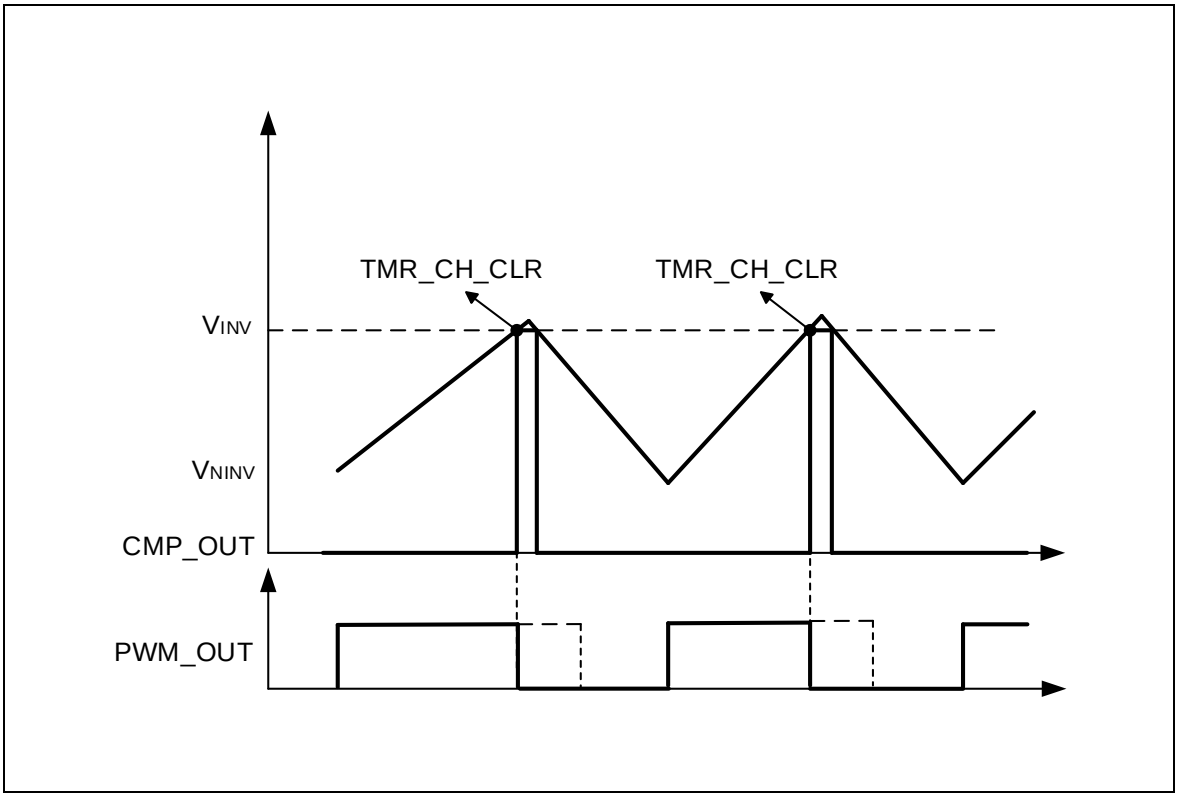
1) 将电流传感器的输出接入比较器的正相输入端；
2) 根据电流限制值设置比较器的反相输入端；
3) 将比较器输出重定向至定时器的通道清除（TMRx_CH_CLR）；

完成以上设置后，当电流传感器输出大于设定的电流限制值时，比较器将输出高电平给定时器TMRx_CH_CLR，用于清除定时器的PWM输出，从而达到限制电路中的电流的目的。在下一次溢出事件之后如果电流低于限制值，则会恢复定时器的PWM输出。

逐周期电流控制示意图如下，其中

1) V_{NINV}为比较器正相输入
2) V_{INV}为比较器反相输入
3) PWM_OUT实线为有电流控制时的输出
4) PWM_OUT虚线部分为无电流控制时的输出

图 5. 逐周期电流控制



3.1.2 资源准备

1) 硬件环境:

AT-START-F421 BOARD

2) 软件环境

AT32F421_Firmware_Library_V2.x.x\project\at_start_f421\examples\cmp\cycle_by_cycle_control

3.1.3 软件设计

1) 配置流程

- 比较器输入引脚配置，输入引脚需要配置为模拟模式
- 比较器输出引脚配置，输出引脚需要配置为复用模式
- 比较器参数初始化，包括反相端选择、输出重映射选择、输出极性等
- 定时器输出引脚配置，用于PWM输出
- 定时器基本参数初始化，包括计数值、分频值等
- 定时器PWM参数配置
- 开启通道清除功能，并将通道清除输入映射至CMP_OUT

2) 代码介绍

- 比较器配置函数代码描述

```
void cmp_config(void)
```

```
{
  cmp_init_type cmp_init_struct;
  /* gpioa peripheral clock enable */
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  /* configure pa1: pa1 is used as cmp1 non inveting input */
  gpio_init_struct.gpio_pins = GPIO_PINS_1;
  gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_init_struct.gpio_pins = GPIO_PINS_0;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_7);

  /* cmp peripheral clock enable */
  crm_periph_clock_enable(CRM_CMP_PERIPH_CLOCK, TRUE);

  /* cmp1 init: pa1 is used cmp1 inverting input */
  cmp_default_para_init(&cmp_init_struct);
  cmp_init_struct.cmp_inverting = CMP_INVERTING_VREFINT;
  cmp_init_struct.cmp_output = CMP_OUTPUT_TMR1CHCLR;
  cmp_init_struct.cmp_polarity = CMP_POL_NON_INVERTING;
  cmp_init_struct.cmp_speed = CMP_SPEED_FAST;
  cmp_init_struct.cmp_hysteresis = CMP_HYSTERESIS_NONE;
  cmp_init(CMP1_SELECTION, &cmp_init_struct);

  cmp_scal_brg_config(CMP_SCAL_BRG_11);

  /* enable cmp1 */
  cmp_enable(CMP1_SELECTION, TRUE);
}
```

- 定时器配置函数代码描述

```
void tmr1_init(void)
{
  tmr_output_config_type tmr_output_structure;

  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE8, GPIO_MUX_2);
  gpio_default_para_init(&gpio_init_struct);
  gpio_init_struct.gpio_pins = GPIO_PINS_8;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOA, &gpio_init_struct);

  nvic_irq_enable(TMR1_CH_IRQn, 0, 1);

  /* tmr1 time base configuration */
  tmr_base_init(TMR1, 9999, 0);
  tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
  tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV1);

  /* channel 4 configuration in pwm mode */
  tmr_output_default_para_init(&tmr_output_structure);

  tmr_output_structure.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
  tmr_output_structure.oc_idle_state = FALSE;
  tmr_output_structure.occ_idle_state = FALSE;
  tmr_output_structure.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
  tmr_output_structure.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
  tmr_output_structure.oc_output_state = TRUE;
  tmr_output_structure.occ_output_state = FALSE;
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_structure);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 5000);

  tmr_output_channel_switch_select(TMR1, TMR_CH_SWITCH_SELECT_CXORAW_OFF);
  tmr_output_channel_switch_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);

  /* tmr1 output enable */
```

```
    tmr_output_enable(TMR1, TRUE);


    /* tmr1 counter enable */
    tmr_counter_enable(TMR1, TRUE);
}
```

3.1.4 实验效果

当电流传感器输出大于设定的电流限制值时，清除定时器的PWM输出，限制电路中的电流。在下一次溢出事件之后如果电流低于限制值，则会恢复定时器的PWM输出。

3.2 输出消隐功能

3.2.1 功能简介

前面2.4 输出消隐功能提到了输出消隐的功能。下面来看一下这个功能的具体用法，首先我们需要在TMR1/3/15中选择一个TMR_CHx来作为比较器的消隐窗口，这里需要相应的配置好CMP_CTRLSTS中的CMPBLANKING位来进行选择。由于没有单独的使能位，只要此位非0就表示开启了blanking功能。

3.2.2 资源准备

1) 硬件环境:

AT-START-F421 BOARD

2) 软件环境

AT32F421_Firmware_Library_V2.x.x\project\at_start_f421\examples\cmp\blanking

3.2.3 软件设计

1) 配置流程

- 比较器输入引脚配置，输入引脚需要配置为模拟模式
- 比较器输出引脚配置，输出引脚需要配置为复用模式
- 比较器参数初始化，包括反相端选择、输出重映射选择、输出极性等
- 定时器输出引脚配置，用于PWM输出
- 定时器基本参数初始化，包括计数值、分频值等
- 定时器PWM参数配置

2) 代码介绍

- 比较器配置函数代码描述

```
void cmp_config(void)
```

```
{
  cmp_init_type cmp_init_struct;
  gpio_init_type gpio_init_struct;

  /* gpioa peripheral clock enable */
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  /* configure pa1: pa1 is used as cmp1 non inveting input */
  gpio_init_struct.gpio_pins = GPIO_PINS_1;
  gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_init_struct.gpio_pins = GPIO_PINS_0;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_7);

  /* cmp peripheral clock enable */
  crm_periph_clock_enable(CRM_CMP_PERIPH_CLOCK, TRUE);

  /* cmp1 init: pa1 is used cmp1 inverting input */
  cmp_default_para_init(&cmp_init_struct);
  cmp_init_struct.cmp_inverting = CMP_INVERTING_1_4VREFINT;
  cmp_init_struct.cmp_output = CMP_OUTPUT_NONE;
  cmp_init_struct.cmp_polarity = CMP_POL_NON_INVERTING;
  cmp_init_struct.cmp_speed = CMP_SPEED_FAST;
  cmp_init_struct.cmp_hysteresis = CMP_HYSTERESIS_NONE;

  cmp_init(CMP1_SELECTION, &cmp_init_struct);

  cmp_scal_brg_config(CMP_SCAL_BRG_11);

  cmp_blanking_config(CMP_BLANKING_TMR1_CH4);
```

```
  /* enable cmp1 */
  cmp_enable(CMP1_SELECTION, TRUE);
}
```

■ 定时器配置函数代码描述

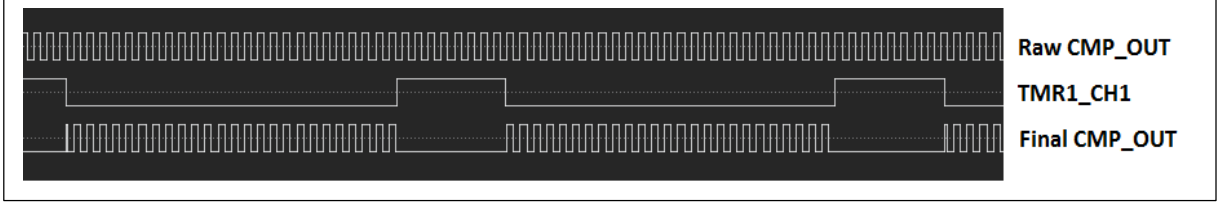
```
void tmr1_pwm_init(void)
{
  gpio_init_type gpio_init_struct;
  tmr_output_config_type tmr_oc_init_structure;

  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE11, GPIO_MUX_2);

  gpio_default_para_init(&gpio_init_struct);
  gpio_init_struct.gpio_pins = GPIO_PINS_11;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOA, &gpio_init_struct);

  /* tmr1 time base configuration */
  tmr_base_init(TMR1, 400, 99);
  tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
  tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV1);

  /* channel 4 configuration in pwm mode */
  tmr_output_default_para_init(&tmr_oc_init_structure);
  tmr_oc_init_structure.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
  tmr_oc_init_structure.oc_idle_state = FALSE;
  tmr_oc_init_structure.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
  tmr_oc_init_structure.oc_output_state = TRUE;
  tmr_oc_init_structure.occ_idle_state = TRUE;
  tmr_oc_init_structure.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
  tmr_oc_init_structure.occ_output_state = TRUE;
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_4, &tmr_oc_init_structure);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_4, 100);
  tmr_output_channel_buffer_enable(TMR1, TMR_SELECT_CHANNEL_4, TRUE);
```

```
    /* tim1 counter enable */
    tmr_counter_enable(TMR1, TRUE);

    /* output enable */
    tmr_output_enable(TMR1, TRUE);
}
```

3.2.4 实验效果

实验选择TMR1_CH1的输出比较PWM模式1来作为CMP的消隐窗口，外部输入100kHz的方波来模拟CMP同相输入端。则可以得到如下波形：

图 6. 输出消隐波形



3.3 干扰滤波功能

3.3.1 功能简介

对于干扰滤波功能也使用TMR来做一个辅助测试，使用TMR1的PWM模式来模拟CMP同相输入信号，然后完成对CMP比较后的输出波形的滤波。

3.3.2 资源准备

1) 硬件环境:

AT-START-F421 BOARD

2) 软件环境

AT32F421_Firmware_Library_V2.x.x\project\at_start_f421\examples\cmp\glitch_filter

3.3.3 软件设计

1) 配置流程

- 比较器输入引脚配置，输入引脚需要配置为模拟模式
- 比较器输出引脚配置，输出引脚需要配置为复用模式
- 比较器参数初始化，包括反相端选择、输出重映射选择、输出极性等
- 比较器滤波配置，H_PULSE_CNT=63，L_PULSE_CNT =0，即滤掉64个PCLK的高电平

- 定时器输出引脚配置，用于PWM输出
- 定时器基本参数初始化，包括计数值、分频值等
- 定时器PWM参数配置
- 连接TMR1_CH1(PA8)与CMP_NINV(PA1)

2) 代码介绍

- 比较器配置函数代码描述

```
void cmp_config(void)
{
  cmp_init_type cmp_init_struct;
  gpio_init_type gpio_init_struct;

  /* gpioa peripheral clock enable */
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  /* configure pa1: pa1 is used as cmp1 non inveting input */
  gpio_init_struct.gpio_pins = GPIO_PINS_1;
  gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_init_struct.gpio_pins = GPIO_PINS_0;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_7);

  /* cmp peripheral clock enable */
  crm_periph_clock_enable(CRM_CMP_PERIPH_CLOCK, TRUE);

  /* cmp1 init: pa1 is used cmp1 inverting input */
  cmp_default_para_init(&cmp_init_struct);
  cmp_init_struct.cmp_inverting = CMP_INVERTING_1_4VREFINT;
  cmp_init_struct.cmp_output = CMP_OUTPUT_NONE;
  cmp_init_struct.cmp_polarity = CMP_POL_NON_INVERTING;
  cmp_init_struct.cmp_speed = CMP_SPEED_FAST;
```

```
  cmp_init_struct.cmp_hysteresis = CMP_HYSTERESIS_NONE;

  cmp_init(CMP1_SELECTION, &cmp_init_struct);

  cmp_scal_brg_config(CMP_SCAL_BRG_11);

  cmp_blanking_config(CMP_BLANKING_TMR1_CH4);

  /* enable cmp1 */
  cmp_enable(CMP1_SELECTION, TRUE);
}
```

■ 定时器配置函数代码描述

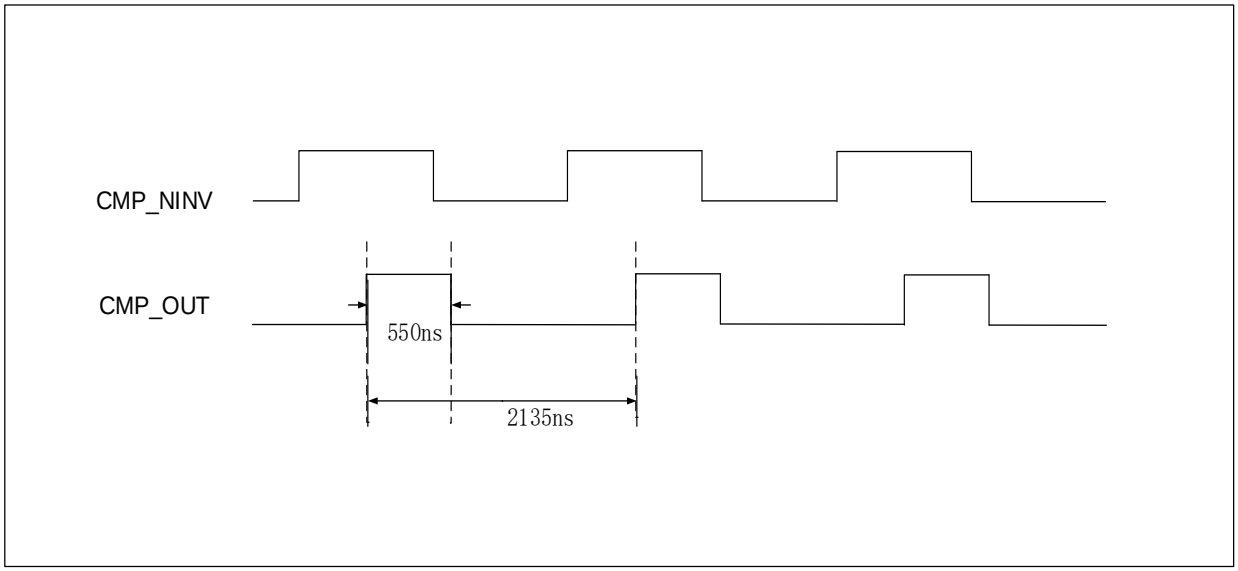
```
void tmr1_pwm_init(void)
{
  gpio_init_type gpio_init_struct;
  tmr_output_config_type tmr_oc_init_structure;

  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE11, GPIO_MUX_2);

  gpio_default_para_init(&gpio_init_struct);
  gpio_init_struct.gpio_pins = GPIO_PINS_11;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOA, &gpio_init_struct);

  /* tmr1 time base configuration */
  tmr_base_init(TMR1, 400, 99);
  tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
  tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV1);

  /* channel 4 configuration in pwm mode */
  tmr_output_default_para_init(&tmr_oc_init_structure);
  tmr_oc_init_structure.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
  tmr_oc_init_structure.oc_idle_state = FALSE;
```

```
  tmr_oc_init_structure.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
  tmr_oc_init_structure.oc_output_state = TRUE;
  tmr_oc_init_structure.occ_idle_state = TRUE;
  tmr_oc_init_structure.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
  tmr_oc_init_structure.occ_output_state = TRUE;
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_4, &tmr_oc_init_structure);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_4, 100);
  tmr_output_channel_buffer_enable(TMR1, TMR_SELECT_CHANNEL_4, TRUE);

  /* tim1 counter enable */
  tmr_counter_enable(TMR1, TRUE);

  /* output enable */
  tmr_output_enable(TMR1, TRUE);
}
```

3.3.4 实验效果

在主频120mHz时，每个PCLK周期为8.33ns，64个PCLK就是533.33ns，CMP滤波前高电平时长为1066.66ns，则CMP滤波后高电平占空比减半，即为25%占空比。但是，以上仅为理论值，实际测试发现最终输出会有误差，这是由于CMP输出有一定的偏移，在无滤波的情况下实际CMP输出的高电平长度会比输入多26ns，偏移量会随着输入频率的增加而增加，最多26ns。实测得到的波形如下：

图 7. 干扰滤波波形



3.4 深度睡眠模式唤醒

3.4.1 功能简介

在ADC采集模拟电压的应用中，由于ADC采样频率极高，长时间工作在运行状态会导致系统功耗大幅增加。而如果只需在超过预定义阈值时测量模拟电压，则能够有效的降低系统功耗。利用比较器的以下两个特性：

- CMP极性选择逻辑和输出端口的重定向工作独立于PCLK时钟
- CMP输出可以连接到EXINT线21

可以将MCU从深度睡眠模式唤醒，从而实现降低系统功耗的目的。开启方式只需要在使能CMP的基础上，将EXINT 21配置为中断或事件模式即可。需要注意的是在退出深度睡眠模式后需要重新配置系统时钟。

3.4.2 资源准备

1) 硬件环境:

AT-START-F421 BOARD

2) 软件环境

AT32F421_Firmware_Library_V2.x.x\project\at_start_f421\examples\cmp\deep_sleep_mode

3.4.3 软件设计

1) 配置流程

- 比较器输入引脚配置，输入引脚需要配置为模拟模式
- 比较器输出引脚配置，输出引脚需要配置为复用模式
- 比较器参数初始化，包括反相端选择、输出重映射选择、输出极性等
- 比较器外部中断配置，中断模式、中断线极性、NVIC配置等
- 进入深度睡眠模式，等待CMP唤醒MCU

2) 代码介绍

- main函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;

  /* enable pwc and bpr clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* congfig the voltage regulator mode.only used with deep sleep mode */
  pwc_voltage_regulate_set(PWC_REGULATOR_EXTRA_LOW_POWER);
```

```
  system_clock_config();

  at32_board_init();

  uart_print_init(115200);

  /* cmp1 configuration */
  cmp_config();

  while (1)
  {
    printf("\r\nenter deep sleep mode...\r\n");
    /* enter deep sleep mode */
    pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

    /* wait clock stable */
    for(index = 0; index < 600; index++)
    {
      __NOP();
    }

    sysclk_config_deep_sleep();
    printf("\r\nwakeup from deep_sleep mode by cmp, mcu is running...\r\n");
    delay_sec(1);
  }
}
```

■ 比较器配置函数代码描述

```
void cmp_config(void)
{
  cmp_init_type cmp_init_struct;
  /* gpioa peripheral clock enable */
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  /* configure pa1: pa1 is used as cmp1 non inveting input */
  gpio_init_struct.gpio_pins = GPIO_PINS_1;
  gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG;
```

```
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_init_struct.gpio_pins = GPIO_PINS_6;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE6, GPIO_MUX_7);

  /* cmp peripheral clock enable */
  crm_periph_clock_enable(CRM_CMP_PERIPH_CLOCK, TRUE);

  /* cmp1 init: pa1 is used cmp1 inverting input */
  cmp_default_para_init(&cmp_init_struct);
  cmp_init_struct.cmp_inverting = CMP_INVERTING_VREFINT;
  cmp_init_struct.cmp_output = CMP_OUTPUT_NONE;
  cmp_init_struct.cmp_polarity = CMP_POL_NON_INVERTING;
  cmp_init_struct.cmp_speed = CMP_SPEED_FAST;
  cmp_init_struct.cmp_hysteresis = CMP_HYSTERESIS_NONE;
  cmp_init(CMP1_SELECTION, &cmp_init_struct);

  cmp_scal_brg_config(CMP_SCAL_BRG_11);

  /* enable cmp1 */
  cmp_enable(CMP1_SELECTION, TRUE);

  cmp_exint_init();
}
```

■ 比较器外部中断配置函数代码描述

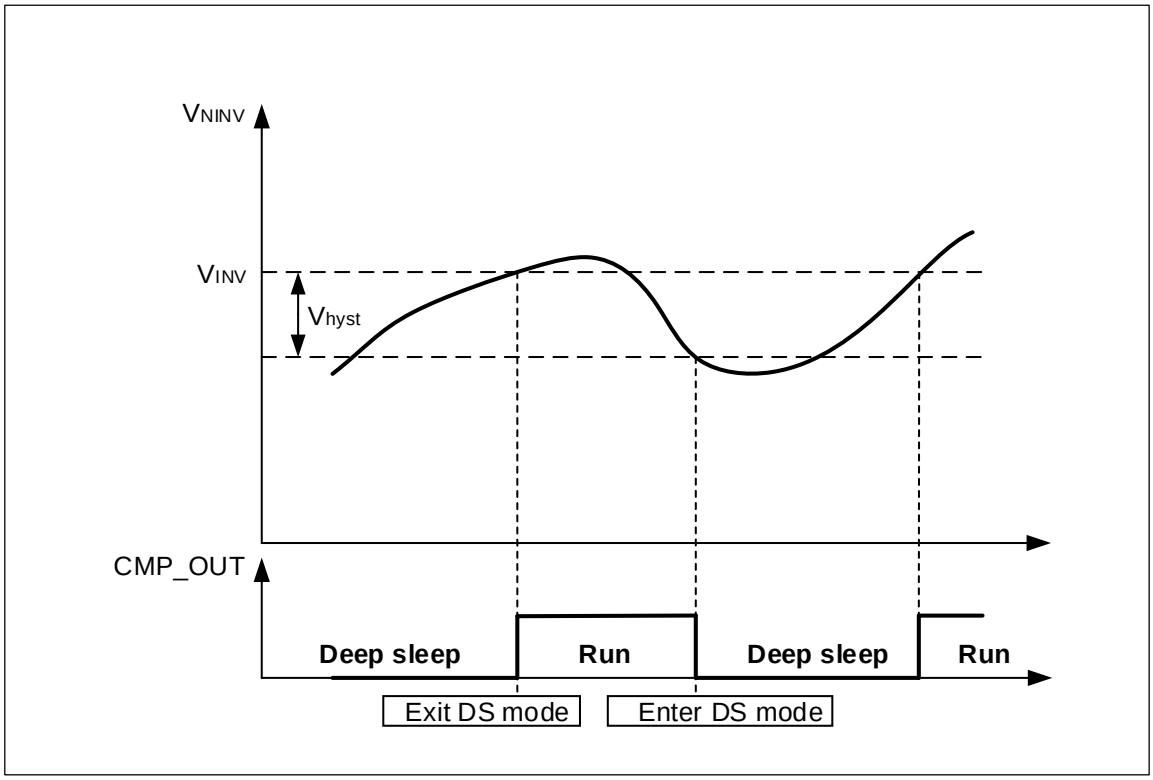
```
void cmp_exint_init(void)
{
  exint_init_type exint_init_struct;

  exint_default_para_init(&exint_init_struct);
  exint_init_struct.line_enable = TRUE;
```

```
    exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
    exint_init_struct.line_select = EXINT_LINE_21;
    exint_init_struct.line_polarity = EXINT_TRIGGER_FALLING_EDGE;
    exint_init(&exint_init_struct);

    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
    nvic_irq_enable(ADC1_CMP_IRQn, 1, 0);
}
```

3.4.4 实验效果

实验效果如下图。

图 8. 深度睡眠模式唤醒



4 版本历史

表 3. 文档版本历史

日期	版本	变更
2021.11.30	2.0.0	最初版本
2022.12.27	2.0.1	修改图片格式
2023.04.11	2.0.2	新增表1、表2以及3.1章节内容

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。