AN0085
入门指南

AT32 MCU 定时器入门指南

前言

AT32 的定时器功能比较强大，根据功能不同可以分为定时器、通用定时器、高级控制定时器。本文主要就定时器溢出中断进行基础讲解和案例解析。

支持型号列表：

支持型号	AT32F4 系列 AT32L0 系列

目录

表目录

图目录

1 AT32 定时器概述

定时器种类有基本定时器、通用定时器、高级控制定时器，以 AT32F435xx 举例，下表为各种类型定时器的功能总表。本文主要就定时器溢出中断进行基础讲解和案例解析。

表 1. 各定时器功能总表

Timer 类型	Timer	计数位数	计数方式	重复计数器	预分频系数	DMA 请求产生	捕获/比较通道	PWM 输入模式	ETR 输入	刹车输入
高级控制定时器	TMR1 TMR8	16	向上 向下 向上/向下	8 位	1~65535	支持	4	支持	支持	支持
通用定时器	TMR2 TMR5	16/32	向上 向下 向上/向下	不支持	1~65535	支持	4	支持	支持	不支持
	TMR3 TMR4	16	向上 向下 向上/向下	不支持	1~65535	支持	4	支持	支持	不支持
	TMR9 TMR12	16	向上	不支持	1~65535	不支持	2	支持	不支持	不支持
	TMR10 TMR11 TMR13 TMR14	16	向上	不支持	1~65535	不支持	1	不支持	不支持	不支持
基本定时器	TMR6 TMR7	16	向上	不支持	1~65535	支持	不支持	不支持	不支持	不支持

Timer 类型	Timer	计数位数	计数方式	PWM 输出	单周期输出	互补输出	死区	编码器接口连接	霍尔传感器接口连接	连动外设
高级控制定时器	TMR1 TMR8	16	向上 向下 向上/向下	支持	支持	支持	支持	支持	支持	定时器同步
通用定时器	TMR2 TMR5	16/32	向上 向下 向上/向下	支持	支持	不支持	不支持	支持	支持	定时器同步
	TMR3 TMR4	16	向上 向下 向上/向下	支持	支持	不支持	不支持	支持	支持	定时器同步
	TMR9 TMR12	16	向上	支持	支持	不支持	不支持	不支持	不支持	定时器同步 ADC/DAC

Timer 类型	Timer	计数位数	计数方式	PWM 输出	单周期输出	互补输出	死区	编码器接口连接	霍尔传感器接口连接	连动外设
	TMR10 TMR11 TMR13 TMR14	16	向上	支持	支持	不支持	不支持	不支持	不支持	无
基本定时器	TMR6 TMR7	16	向上	不支持	不支持	不支持	不支持	不支持	不支持	DAC

2 例 定时器溢出中断

2.1 功能简介

定时器溢出中断是定时器最基础功能，进入中断的时间周期可由相关寄存器配置。

- 定时器计数器值 TMRx_CVAL
- 定时器预分频寄存器 TMRx_DIV
- 定时器周期寄存器（TMRx_PR）

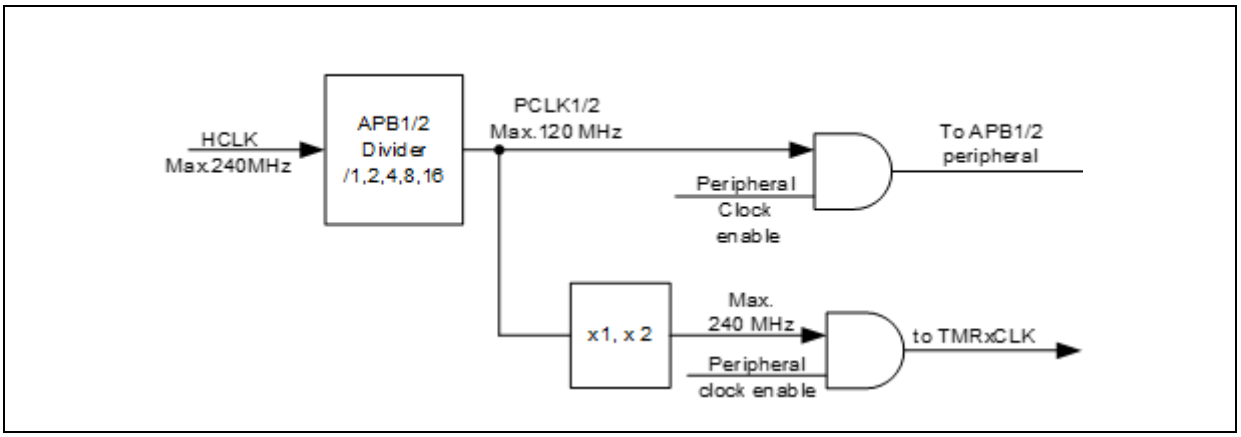
  定时器中断频率计算公式如下

$$F = \frac{TMRxCLK}{(TMRx\_DIV + 1)(TMRx\_PR + 1)}$$

  其中，TMRxCLK 虽然来源于 APB1/2 时钟，但下图时钟架构可以看出，档 APB1/2 Divider 存在非 1 除频时，TMRxCLK 会有 x2。例如 AHB = 240 MHz，APB1/2 = 120MHz，TMRxCLK 的实际频率为 240 MHz.

  以向上计数模式举例，TMREN bit 使能后，TMRx_CVAL 会开始进行累加 1，直到其值等于 TMRx_PR 后，OVFIF 位会置起并且定时器会触发溢出中断事件，（若溢出中断有开启，则会产生溢出中断），同时 TMRx_CVAL 会再次从 0 开始计数，周而复始。

图 1. 定时器时钟源架构



2.2 资源准备

1) 硬件环境:

   对应产品型号的 AT-START BOARD

2) 软件环境

   project\at_start_xxx\examples\tmr\timer_base

2.3 软件设计

1) 配置流程
   - 编写定时器溢出中断函数的应用程序
   - 开启定时器外设时钟
   - 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
   - 配置定时器为向上计数方向

- 开启定时器溢出中断
- 开启 NVIC 溢出中断
- 开启定时器计数

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
/* 系统时钟配置 */
system_clock_config();
/* LED 延时函数等初始化 */
at32_board_init();
/* 获取系统时钟 */
crm_clocks_freq_get(&crm_clocks_freq_struct);
/* 点亮 LED2/LED3/LED4 */
at32_led_on(LED2);
at32_led_on(LED3);
at32_led_on(LED4);
/* 开启 TMR1 时钟 */
crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);

/* 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器 */
/* systemclock/24000/10000 = 1hz */
tmr_base_init(TMR1, 9999, (crm_clocks_freq_struct.ahb_freq / 10000) - 1);
/*配置定时器为向上计数方向，如果选择向上计数也可以不配置该语句，
因为 TMR 默认就是向上计数模式 */
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
/* 开启定时器溢出中断 */
tmr_interrupt_enable(TMR1, TMR_OVF_INT, TRUE);

/* 开启 NVIC 溢出中断 */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
nvic_irq_enable(TMR1_OVF_TMR10_IRQn, 0, 0);

/* 开启定时器计数 */
tmr_counter_enable(TMR1, TRUE);
clkout_config();
```

```
while(1)
{
}
}
```

- TMR1_OVF_TMR10_IRQHandler 中断函数代码描述

```
void TMR1_OVF_TMR10_IRQHandler(void)
{
  /* 判断溢出标志位是否置起 */
  if(tmr_flag_get(TMR1, TMR_OVF_FLAG) == SET)
  {
    /* 增加应用程序 */
    at32_led_toggle(LED3);
    tmr_flag_clear(TMR1, TMR_OVF_FLAG);
  }
}
```

2.4 实验效果

- LED3 每 1 秒翻转一次。

3 例 PWM 输出

3.1 功能简介

定时器的输出部分由比较器和输出控制构成，用于编程输出信号的周期、占空比、极性。高级定时器的输出部分在不同通道上有所不同。

高级定时器在通道 1 到通道 3 上拥有互补输出，且配备死区调节；通道 1 到通道 4 拥有刹车控制。通用定时器的输出部分没有上述功能，只配备了 4 个通道输出。基本定时器、通用定时器和高级定时器的具体功能差异可查看 RM 的 TMR 章节。

如下图为高级定时器通道 1 到 3 输出部分原理图：

图 2. 高级定时器通道 1 到 3 输出部分原理图

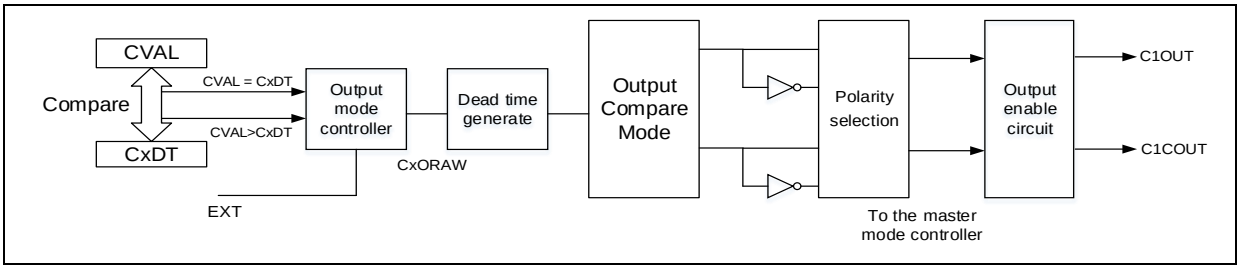
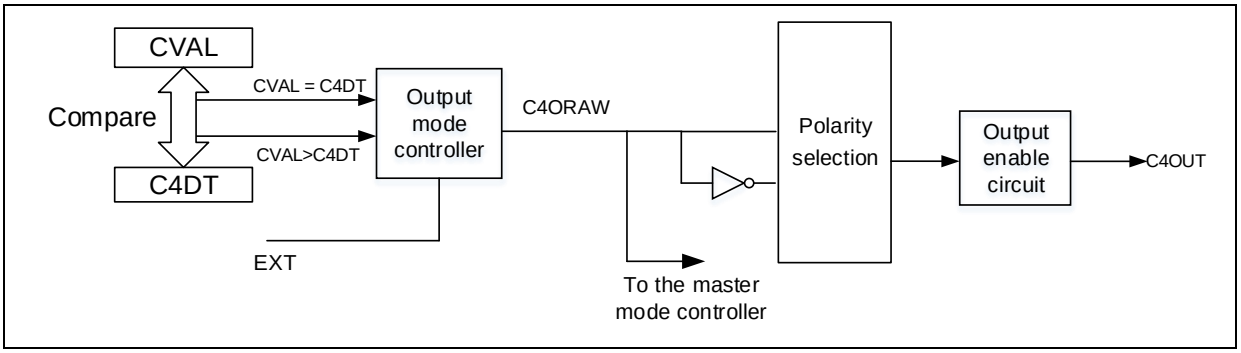


图 3. 高级定时器通道 4 输出部分原理图



PWM 输出是定时器最常用的输出模式，分为 PWM 模式 A 和 PWM 模式 B。其差异在于：

PWM 模式 A：

－OWCDIR=0，若 TMRx_C1DT>TMRx_CVAL 时设置 C1ORAW 为高，否则为低；

－OWCDIR=1，若 TMRx_ C1DT <TMRx_CVAL 时设置 C1ORAW 为低，否则为高。

PWM 模式 B：

－OWCDIR=0，若 TMRx_ C1DT >TMRx_CVAL 时设置 C1ORAW 为低，否则为高；

－OWCDIR=1，若 TMRx_ C1DT <TMRx_CVAL 时设置 C1ORAW 为高，否则为低。

3.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\7_pwm_output

3.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置定时器为向上计数方向
- 配置定时器输出通道为 PWM 模式 B
- 开启定时器计数

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();
  /* 初始化板载设备 */
  at32_board_init();

  /* get system clock */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* turn led2/led3/led4 on */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 打开 tmr1/gpioa/gpiob 时钟 */
  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  /* 配置 TMR1 输出管脚 */
  gpio_init_struct.gpio_pins = GPIO_PINS_8 | GPIO_PINS_9 | GPIO_PINS_10 | GPIO_PINS_11;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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_13 | GPIO_PINS_14 | GPIO_PINS_15;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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(GPIOB, &gpio_init_struct);

gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE8, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE9, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE10, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE11, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE13, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE14, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE15, GPIO_MUX_1);



/* tmr1 初始化 ----------------------------------------------------
  generate 7 pwm signals with 4 different duty cycles:
  prescaler = 0, tmr1 counter clock = apb2_freq *2

  the objective is to generate 7 pwm signal at 17.57 khz:
     - tim1_period = (apb2_freq * 2 / 17570) - 1
  the channel 1 and channel 1n duty cycle is set to 50%
  the channel 2 and channel 2n duty cycle is set to 37.5%
  the channel 3 and channel 3n duty cycle is set to 25%
  the channel 4 duty cycle is set to 12.5%
  the timer pulse is calculated as follows:
     - channelxpulse = dutycycle * (tim1_period - 1) / 100
---------------------------------------------------------------------- */
/* compute the value to be set in arr regiter to generate signal frequency at 17.57 khz */
timerperiod = ((crm_clocks_freq_struct.apb2_freq * 2) / 17570 ) - 1;
/* compute ccr1 value to generate a duty cycle at 50% for channel 1 and 1n */
channel1pulse = (uint16_t) (((uint32_t) 5 * (timerperiod - 1)) / 10);
/* compute ccr2 value to generate a duty cycle at 37.5%  for channel 2 and 2n */
channel2pulse = (uint16_t) (((uint32_t) 375 * (timerperiod - 1)) / 1000);
/* compute ccr3 value to generate a duty cycle at 25%  for channel 3 and 3n */
channel3pulse = (uint16_t) (((uint32_t) 25 * (timerperiod - 1)) / 100);
/* compute ccr4 value to generate a duty cycle at 12.5%  for channel 4 */
```

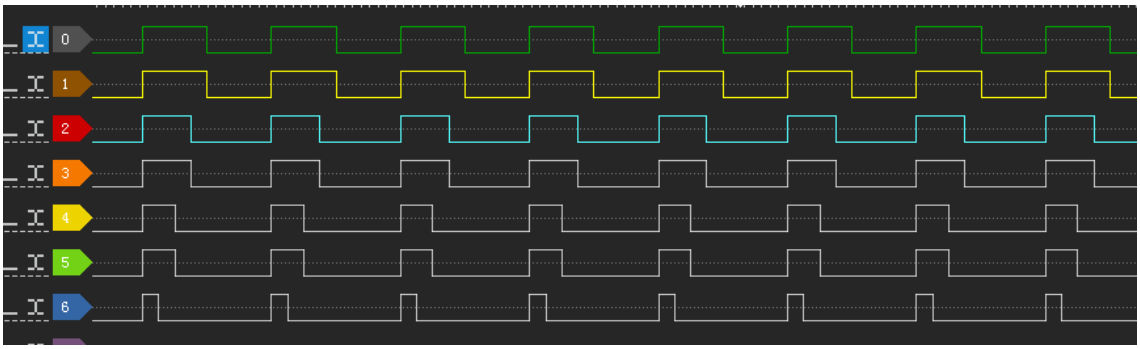
```
channel4pulse = (uint16_t) (((uint32_t) 125 * (timerperiod- 1)) / 1000);
/* 配置 TMR 为向上计数模式 */
tmr_base_init(TMR1, timerperiod, 0);
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);

/* 配置通道 1/2/3/4 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_B;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = TRUE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.occ_idle_state = FALSE;
/* channel 1 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, channel1pulse);
/* channel 2 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_2, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_2, channel2pulse);
/* channel 3 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_3, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_3, channel3pulse);
/* channel 4 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_4, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_4, channel4pulse);

/*TMR1 输出总开关打开 */
tmr_output_enable(TMR1, TRUE);
/*使能 TMR1 */
tmr_counter_enable(TMR1, TRUE);

while(1)
{}
```

3.4 实验效果

- 通过逻辑分析仪或者示波器可将波形打出来。

  如下图：

图 4. 7 路 PWM 输出



图中通道 1 到 4 输出频率相同但占空比不同的波形，互补通道通过输出极性的调节与其对应的通道输出相同的波形。

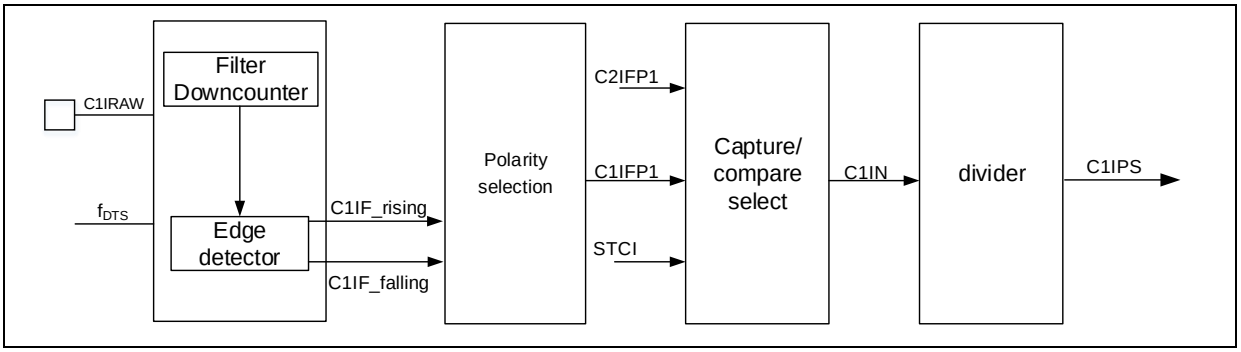
4 例 PWM 输入捕获

4.1 功能简介

定时器的输入部分由专门的捕获电路实现，可用于对输入信号的滤波、选择、分频和输入捕获功能；通过对捕获值的计算，可得到输入波形的频率和占空比。

如下图为输入部分原理图：

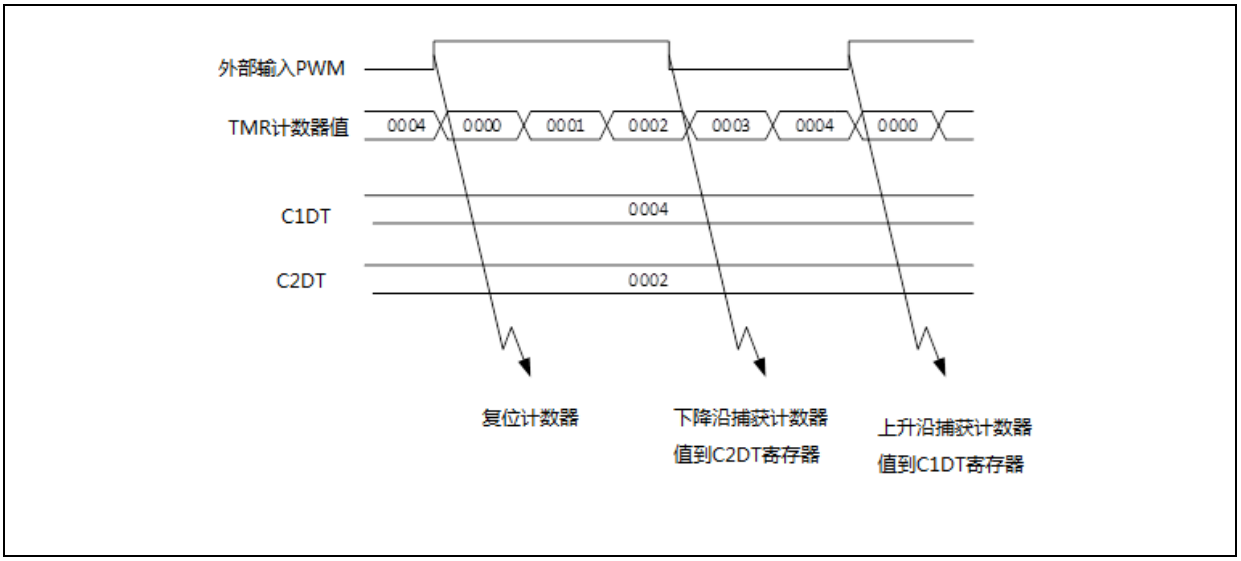
图 5. 定时器输入部分原理图



输入模式下，当选中的触发信号被检测到时，通道寄存器（TMRx_CxDT）会记录当前计数器计数值，并将捕获比较中断标志位（CxIF）置 1，若已使能通道中断（CxIEN）、通道 DMA 请求（CxDEN）则产生相应的中断和 DMA 请求。若在 CxIF 已置 1 后检测到选中的触发信号，则将 CxOF 位置 1。

另外，还提供了更加高效的 PWM 波输入捕获功能，可以更方便的计算出输入波形的频率和占空比。此模式的通过内部两个 CxDT 寄存器实现，输入波形通过定时器的通道 1 或者通道 2 输入即可。

图 6. 定时器捕获 PWM 波原理图



将定时器配置成 PWM 输入模式后，可通过 C1DT 和 C2DT 的值计算出对应 PWM 波形的频率和占空比：

频率 = TMR_CLK / C1DT

占空比 = C2DT / C1DT

4.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\pwm_input

4.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置定时器为向上计数方向
- 配置定时器的 PWM 输入模式
- 开启定时器计数

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();
  /* 系统时钟初始化 */
  crm_configuration();

  /* nvic configuration */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  nvic_irq_enable(TMR3_GLOBAL_IRQn, 1, 0);

  /* gpio configuration */
  gpio_configuration();
  /* 初始化串口，打印计算出的频率和占空比信息 */
  uart_init(115200);
  /* 配置 TMR3 为 PWM 输入模式 */
  tmr_input_default_para_init(&tmr_ic_init_structure);
  tmr_ic_init_structure.input_filter_value = 0;
  tmr_ic_init_structure.input_channel_select = TMR_SELECT_CHANNEL_2;
  tmr_ic_init_structure.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
  tmr_ic_init_structure.input_polarity_select = TMR_INPUT_RISING_EDGE;
```

```
  tmr_pwm_input_config(TMR3, &tmr_ic_init_structure, TMR_CHANNEL_INPUT_DIV_1);

  /* select the tmr3 input trigger: C2IF2 */
  tmr_trigger_input_select(TMR3, TMR_SUB_INPUT_SEL_C2DF2);

  /* select the sub mode: reset mode */
  tmr_sub_mode_select(TMR3, TMR_SUB_RESET_MODE);

  /* enable the sub sync mode */
  tmr_sub_sync_mode_set(TMR3, TRUE);

  /* tmr enable counter */
  tmr_counter_enable(TMR3, TRUE);

  /* enable the c2 interrupt request */
  tmr_interrupt_enable(TMR3, TMR_C2_INT, TRUE);
  while(1)
  {
    /* 打印出信息 */
    printf("Frequency = %dHZ,Dutycycle = %d%%\r\n", frequency, dutycycle);
  }
```

■ 中断函数代码描述

```
/* TMR3 中断处理函数，计算 PWM 波形的频率和占空比 */
void TMR3_GLOBAL_IRQHandler(void)
{
  /* clear tmr3 channel interrupt pending bit */
  tmr_flag_clear(TMR3, TMR_C2_FLAG);

  /* get the input channel data value */
  ic2value = tmr_channel_value_get(TMR3, TMR_SELECT_CHANNEL_2);

  if (ic2value != 0)
  {
    /* duty cycle computation */
    dutycycle = (tmr_channel_value_get(TMR3, TMR_SELECT_CHANNEL_1) * 100) / ic2value;

    /* frequency computation */
    frequency = system_core_clock / ic2value;
```

```
  }
  else
  {
    dutycycle = 0;
    frequency = 0;
  }
}
```

4.4 实验效果

- 从 PA7 灌入 PWM 波形；
- 将串口 1 连接到上位机，然后通过上位机串口工具即可看到打印信息。

5 例 输入捕获

5.1 功能简介

通过对外部信号的上升沿或者下降沿进行捕获可以实现对外部信号输入捕获并计算频率的功能。本例程实现了对外部信号进行捕获并通过串口打印出频率。

5.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\input_capture

5.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置定时器为向上计数方向
- 配置定时器的输入捕获功能
- 开启定时器计数

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 对系统时钟、板载、串口进行初始化 */
  system_clock_config();
  at32_board_init();
  uart_init(115200);

  /* get system clock */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* turn led2/led3/led4 on */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 开启 tmr3/gpioa 时钟 */
```

```
crm_periph_clock_enable(CRM_TMR3_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

/* timer3 输入配置 */
gpio_init_struct.gpio_pins = GPIO_PINS_7;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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_pin_mux_config(GPIOA, GPIO_PINS_SOURCE7, GPIO_MUX_2);

/* tmr3 configuration: input capture mode ----------------------
    the external signal is connected to tmr3 ch2 pin (pa.07)
    the rising edge is used as active edge,
    the tmr3 c2dt is used to compute the frequency value
------------------------------------------------------------ */
/* tmr3 counter mode configuration */
tmr_base_init(TMR3, 0xFFFF, 0);
tmr_cnt_dir_set(TMR3, TMR_COUNT_UP);

/* configure tmr3 channel2 to get clock signal */
tmr_input_config_struct.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_input_config_struct.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_input_config_struct.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_input_channel_init(TMR3, &tmr_input_config_struct, TMR_CHANNEL_INPUT_DIV_1);
/* 使能输入捕获中断 */
tmr_interrupt_enable(TMR3, TMR_C2_INT, TRUE);

/* tmr2 trigger interrupt nvic init */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
nvic_irq_enable(TMR3_GLOBAL_IRQn, 1, 0);

/* 打开 tmr3 */
tmr_counter_enable(TMR3, TRUE);

while(1)
```

```
  {
    delay(10000);
    /* 打印频率 */
    printf("The external signal frequece is : %d\r\n",tmr3freq);
    tmr3freq = 0;
  }
}
```

■ 中断函数代码描述

```
/* TMR3 中断处理函数，计算 PWM 波形的频率和占空比 */
void TMR3_GLOBAL_IRQHandler(void)
{
  if(tmr_flag_get(TMR3, TMR_C2_FLAG) == SET)
  {
    tmr_flag_clear(TMR3, TMR_C2_FLAG);
    if(capturenumber == 0)
    {
      /* get the Input Capture value */
      ic3readvalue1 = tmr_channel_value_get(TMR3, TMR_SELECT_CHANNEL_2);
      capturenumber = 1;
    }
    else if(capturenumber == 1)
    {
      /* get the Input Capture value */
      ic3readvalue2 = tmr_channel_value_get(TMR3, TMR_SELECT_CHANNEL_2);
      /* capture computation */
      if (ic3readvalue2 > ic3readvalue1)
      {
        capture = (ic3readvalue2 - ic3readvalue1);
      }
      else
      {
        capture = ((0xFFFF - ic3readvalue1) + ic3readvalue2);
      }
      /* frequency computation */
      tmr3freq = (uint32_t) crm_clocks_freq_struct.sclk_freq / capture;
      capturenumber = 0;
    }
  }
```

5.4 实验效果

- 从 PA7 灌入 PWM 波形；
- 将串口 1 连接到上位机，然后通过上位机串口工具即可看到打印信息。

6 例 DMA 传输

6.1 功能简介

定时器拥有强大的 DMA 传输能力，基本每个定时器都支持 DMA 请求的产生。这使得应用更加灵活。

本实验将 src_buffer[0]、src_buffer[1] 和 src_buffer[2]数据通过 DMA 传输到 TMR 的 TMRx_C3DT 寄存器。实现了每个周期占空比都发生改变，且占空比在 src_buffer[0]、src_buffer[1] 和 src_buffer[2]的值之间进行有序的切换。

6.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\dma

6.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置定时器为向上计数方向
- 配置 DMA 通道
- 配置定时器的溢出事件产生 DMA 请求
- 开启定时器计数

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 配置系统时钟以及板载外设 */
    system_clock_config();
  at32_board_init();
  /* get system clock */
  crm_clocks_freq_get(&crm_clocks_freq_struct);
  /* turn led2/led3/led4 on */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);
```

```
/* 打开 tmr1/gpioa/gpiob/dma 时钟 */
crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

/* timer1 输出管脚配置 */
gpio_init_struct.gpio_pins = GPIO_PINS_10;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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_15;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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(GPIOB, &gpio_init_struct);

gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE10, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE15, GPIO_MUX_1);

/* tmr1 dma transfer example -----------------------------------------------
tmr1clk = apb2_freq*2, prescaler = 0, tmr1 counter clock = apb2_freq*2

the objective is to configure tmr1 channel 3 to generate complementary pwm
signal with a frequency equal to 17.57 khz:
    - tmr1_period = (apb2_freq * 2 / 17570) - 1
and a variable duty cycle that is changed by the dma after a specific number of
update dma request.

the number of this repetitive requests is defined by the tmr1 repetition counter,
each 3 update requests, the tmr1 channel 3 duty cycle changes to the next new
value defined by the src_buffer .
------------------------------------------------------------------------------*/
/* compute the value to be set in arr regiter to generate signal frequency at 17.57 khz */
```

```
timerperiod = ((crm_clocks_freq_struct.apb2_freq * 2) / 17570 ) - 1;
/* compute c1dt value to generate a duty cycle at 50% */
src_buffer[0] = (uint16_t) (((uint32_t) 5 * (timerperiod - 1)) / 10);
/* compute c1dt value to generate a duty cycle at 37.5% */
src_buffer[1] = (uint16_t) (((uint32_t) 375 * (timerperiod - 1)) / 1000);
/* compute c1dt value to generate a duty cycle at 25% */
src_buffer[2] = (uint16_t) (((uint32_t) 25 * (timerperiod - 1)) / 100);

tmr_base_init(TMR1, timerperiod, 0);
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);

/* 输出通道配置 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_B;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = TRUE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_output_struct.occ_idle_state = FALSE;
/* channel 3 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_3, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_3, src_buffer[0]);

/* 使能 TMR 的溢出事件 DMA 请求 */
tmr_dma_request_enable(TMR1, TMR_OVERFLOW_DMA_REQUEST, TRUE);

/* 配置 DMA */
/* dma1 channel5 configuration */
dma_reset(DMA1_CHANNEL1);
dma_init_struct.buffer_size = 3;
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
dma_init_struct.memory_base_addr = (uint32_t)src_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)0x4001003C;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
dma_init_struct.peripheral_inc_enable = FALSE;
```

```
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = TRUE;
dma_init(DMA1_CHANNEL1, &dma_init_struct);
/*  配置 DMAMUX */
dmamux_enable(DMA1, TRUE);
dmamux_init(DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_TMR1_OVERFLOW);
dma_channel_enable(DMA1_CHANNEL1, TRUE);

/* tmr1 output enable */
tmr_output_enable(TMR1, TRUE);
/* enable tmr1 */
tmr_counter_enable(TMR1, TRUE);

while(1)
{}
```

6.4 实验效果

从 PA10、PB15 输出 PWM 波形；

7 例 burst 传输

7.1 功能简介

高级定时器和通用定时器除了支持常规的 DMA 传输功能，还额外支持 DMA burst 传输功能。在配置为 burst 传输后，当 TMR 产生一个 DMA 请求可连续传输以 TMR 地址为起始地址的多笔数据；传输数据的起始地址和数据量可通过软件配置。

如何配置 burst 传输：

1. 配置 TMRx_DMACTRL 寄存器，此寄存器的 bit0 到 bit4 为 DMA 传输地址偏移，此值决定了 DMA 传输的起始地址；bit8 到 bit12 为 DMA 传输长度配置，此值决定了 DMA 传输的数据笔数。
2. 配置 DMA 通道，此配置流程与常规 DMA 通道配置相同；需要注意的是 DMA 通道的源与目标地址寄存器中的一个必须为 TMRx_DMADT 寄存器地址；具体是源还是目标就由数据传输的方向决定。

本实验将 src_buffer[0] 和 src_buffer[2]数据通过 burst 传输到 TMR 的 TMRx_PR 和 TMRx_C1DT 寄存器。

7.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\dma_burst

7.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置定时器为向上计数方向
- 配置 DMA 通道和定时器的 burst 功能
- 配置 DMAMUX
- 开启定时器计数

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 配置系统时钟以及板载外设 */
  system_clock_config();
  at32_board_init();
```

```
/* get system clock */
crm_clocks_freq_get(&crm_clocks_freq_struct);

/* turn led2/led3/led4 on */
at32_led_on(LED2);
at32_led_on(LED3);
at32_led_on(LED4);

/* 开启 tmr1/gpioa/gpiob/dma 时钟 */
crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

/* 配置 timer1 通道 1 的 pin 脚 */
gpio_init_struct.gpio_pins = GPIO_PINS_8;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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_pin_mux_config(GPIOA, GPIO_PINS_SOURCE8, GPIO_MUX_1);
/* 配置 TMR1 */
tmr_base_init(TMR1, 0xFFFF, 0);
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);

/* 配合 TMR 的通道 1 为输出模式 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_B;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = FALSE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_output_struct.occ_idle_state = FALSE;
/* channel 1 */
```

```
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 0xFFF);

  /* 配置 TMR 的 burst 功能 */
  tmr_dma_request_enable(TMR1, TMR_OVERFLOW_DMA_REQUEST, TRUE);
  tmr_dma_control_config(TMR1, TMR_DMA_TRANSFER_3BYTES, TMR_PR_ADDRESS);

  /* 配置用于 burst 传输的 DMA 通道*/
  /* dma1 channel5 configuration */
  dma_reset(DMA1_CHANNEL1);
  dma_init_struct.buffer_size = 3;
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
  dma_init_struct.memory_base_addr = (uint32_t)src_buffer;
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
  dma_init_struct.memory_inc_enable = TRUE;
  dma_init_struct.peripheral_base_addr = (uint32_t)0x4001004C;
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
  dma_init_struct.peripheral_inc_enable = FALSE;
  dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
  dma_init_struct.loop_mode_enable = FALSE;
  dma_init(DMA1_CHANNEL1, &dma_init_struct);
  /* 配置 DMAMUX 并使能通道 */
  dmamux_enable(DMA1, TRUE);
  dmamux_init(DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_TMR1_OVERFLOW);
  dma_channel_enable(DMA1_CHANNEL1, TRUE);

  /* 打开 TMR 输出总开关 */
  tmr_output_enable(TMR1, TRUE);
  /* 只能 TMR1 */
  tmr_counter_enable(TMR1, TRUE);

  while(1)
  {}
}
```

7.4 实验效果

- 从 PA8 输出 PWM 波形；

8 例 单脉冲输出

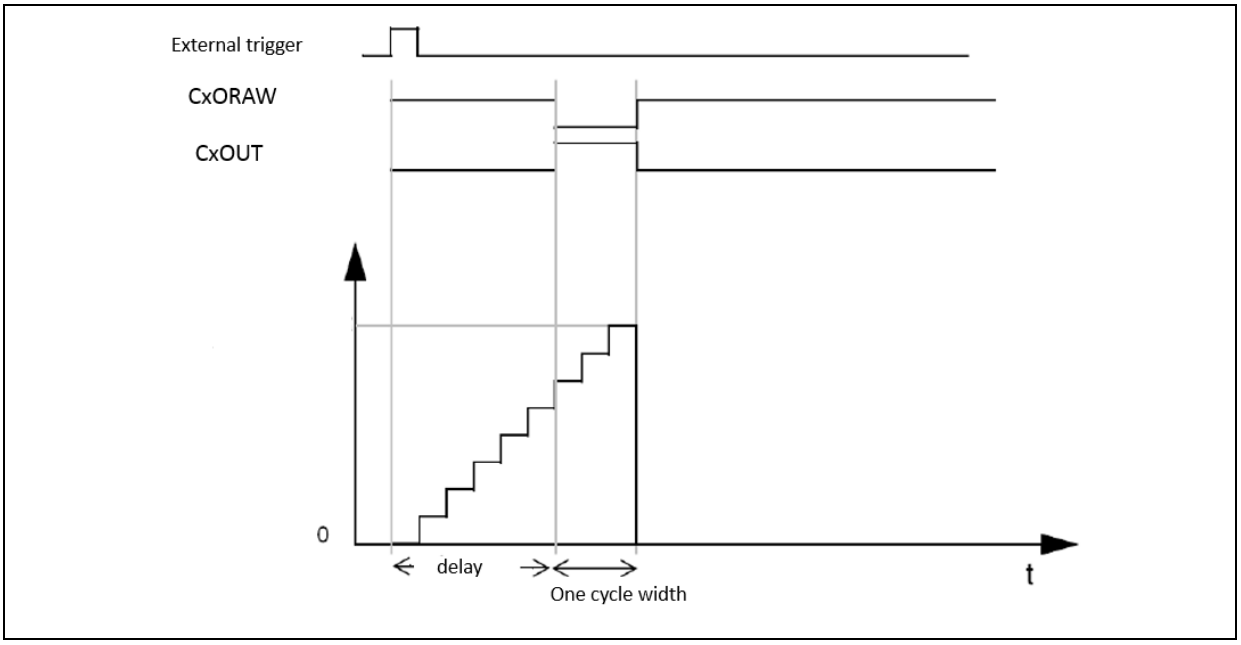
8.1 功能简介

单脉冲输出模式是 PWM 模式的特例，将 OCMEN 位置 1 可开启单周期模式， 此模式下，仅在当前计数周期中进行比较匹配， 完成当前计数后，TMREN 位清 0， 因此仅输出一个脉冲。当配置为向上计数模式时，需要严格配置 CVAL<CxDT≤PR；向下计数时，需严格配置 CVAL>CxDT。

当 TMR 受到外部触发或者软件使能 CNT 时，TMR 开始计数并在此次 overflow 事件时停止计数。此过程中输出也会根据配置产生波形。

单脉冲输出原理如下图：

图 7. 单脉冲输出原理图



图中当受到外部触发后，TMR 开始计数，当 CVAL 等于 CxDT 时，改变输出状态，当 CVAL 溢出时，输出状态再次改变从而达到输出一个单脉冲的目的。

本实验将 TMR4 配置为单脉冲模式。TMR4 的通道 1 配置为输入并充当触发输入的源头，通道 2 配置为输出模式，充当单脉冲输出的端口。

8.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\one_cycle

8.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入、输出管脚

- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置定时器为向上计数方向
- 配置定时器为单脉冲模式
- 配置通道 2 为输出口，并作为 TMR 触发的触发源

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 系统时钟配置 */
  system_clock_config();
  crm_configuration();
  /* get system clock */
  crm_clocks_freq_get(&crm_clocks_freq_struct);
  /* 管脚初始化 */
  gpio_configuration();
  /* compute the prescaler value */
  prescalervalue = (uint16_t) ((crm_clocks_freq_struct.apb1_freq * 2) / 24000000) - 1;
  /* tmr 配置 */
  tmr_base_init(TMR4, 65535, prescalervalue);
  tmr_cnt_dir_set(TMR4, TMR_COUNT_UP);
  tmr_clock_source_div_set(TMR4, TMR_CLOCK_DIV1);
  /* 通道 1 输出配置 */
  tmr_output_default_para_init(&tmr_oc_init_structure);
  tmr_oc_init_structure.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_B;
  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_output_channel_config(TMR4, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
  tmr_channel_value_set(TMR4, TMR_SELECT_CHANNEL_1, 16383);
  /* 通道 2 输入触发配置 */
  tmr_input_default_para_init(&tmr_ic_init_structure);
  tmr_ic_init_structure.input_filter_value = 0;
  tmr_ic_init_structure.input_channel_select = TMR_SELECT_CHANNEL_2;
  tmr_ic_init_structure.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
  tmr_ic_init_structure.input_polarity_select = TMR_INPUT_RISING_EDGE;
  tmr_input_channel_init(TMR4, &tmr_ic_init_structure, TMR_CHANNEL_INPUT_DIV_1);
  /* 使能单脉冲模式 */
```

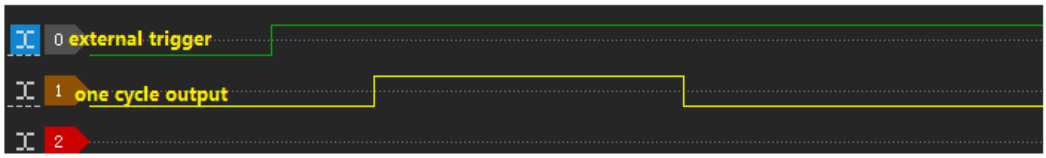
```
    tmr_one_cycle_mode_enable(TMR4, TRUE);
    /* 选择输入触发源 */
    tmr_trigger_input_select(TMR4, TMR_SUB_INPUT_SEL_C2DF2);
    /* 从模式配置 */
    tmr_sub_mode_select(TMR4, TMR_SUB_TRIGGER_MODE);
    while(1)
    {}
}
```

8.4 实验效果

- PB7 输入管脚，外部给上升沿触发即可；
- PB6 为输出管脚，输出单脉冲波形。

使用逻辑分析仪打出波形如下：

图 8. 单脉冲输出结果



9 例 32 位定时器

9.1 功能简介

部分定时器可支持扩展为 32 位定时器（具体请查看 RM）。扩展后，不单 CVAL 寄存器为 32 位，DIV 和 CxDT 都会自动扩展为 32 位。

使能 32 位定时器只需要设置 TMRx_CTRL1 寄存器的 bit10 为 1 即可。

本实验将 TMR2 配置为 32 位定时器模式，然后配置为 PWM 输出模式使其输出 PWM 波形。

9.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\tmr2_32bit

9.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入、输出管脚
- 使能 32 位模式
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置通道输出 PWM 波形

2) 代码介绍

- main 函数代码描述

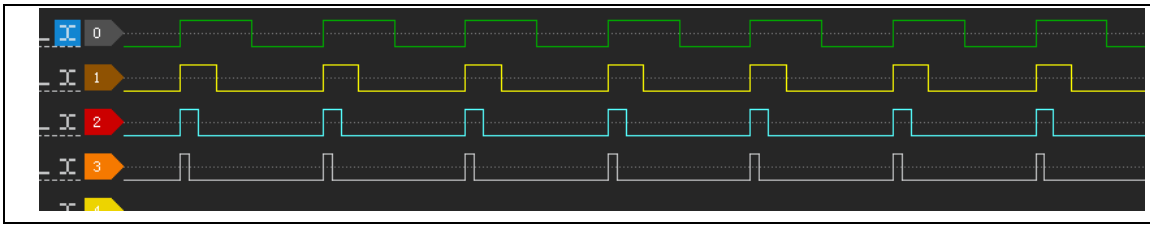
```
int main(void)
{
  /* 配置系统时钟 */
  system_clock_config();
  crm_configuration();
  /* 配置管脚 */
  gpio_configuration();
  /* 使能 32 位定时器 */
  tmr_32_bit_function_enable(TMR2, TRUE);
  /* tmr2 基本配置 */
  tmr_base_init(TMR2, 0xfffff, 0);
  tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);
  tmr_clock_source_div_set(TMR2, TMR_CLOCK_DIV1);
  /* TMR2 通道 1/2/3/4PWM 输出配置 */
```

```
  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_output_channel_config(TMR2, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
  tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_1, ccr1_val);
  tmr_output_channel_buffer_enable(TMR2, TMR_SELECT_CHANNEL_1, TRUE);

  tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_2, &tmr_oc_init_structure);
  tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_2, ccr2_val);
  tmr_output_channel_buffer_enable(TMR2, TMR_SELECT_CHANNEL_2, TRUE);

  tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_3, &tmr_oc_init_structure);
  tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_3, ccr3_val);
  tmr_output_channel_buffer_enable(TMR2, TMR_SELECT_CHANNEL_3, TRUE);

  tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_4, &tmr_oc_init_structure);
  tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_4, ccr4_val);
  tmr_output_channel_buffer_enable(TMR2, TMR_SELECT_CHANNEL_4, TRUE);
  tmr_period_buffer_enable(TMR2, TRUE);
  /* tmr2 定时器使能 */
  tmr_counter_enable(TMR2, TRUE);
  while(1)
  {
  }
}
```

9.4 实验效果

- PA0/1/2/3 输出波形；

图 9. 32 位定时器输出结果



10 例　定时器同步

10.1 功能简介

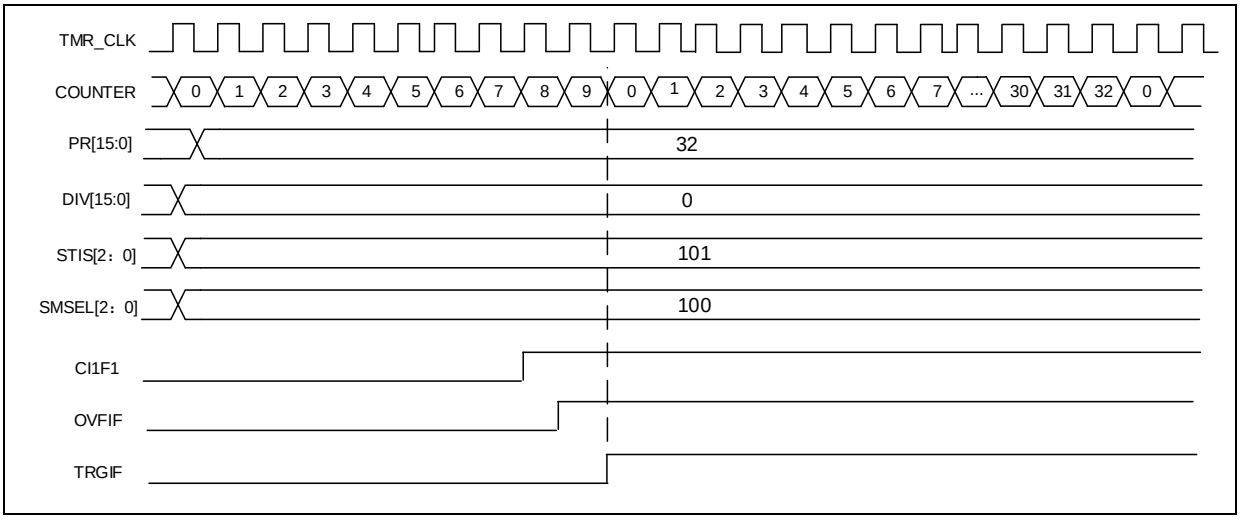
主次定时器之间可由内部连接信号进行同步。主定时器可由 PTOS[2：0]位选择主定时器输出，即同步信息；次定时器由 SMSEL[2：0]位选择从模式，即次定时器的工作模式。

定时器从模式有以下几种：

复位模式：

此模式下，当次定时器收到一个同步信号后，次定时器复位计数器和预分频器，定时器的 CVAL 寄存器变为 0 重新开始计数。若 OVFS 位为 0，将产生一个溢出事件。

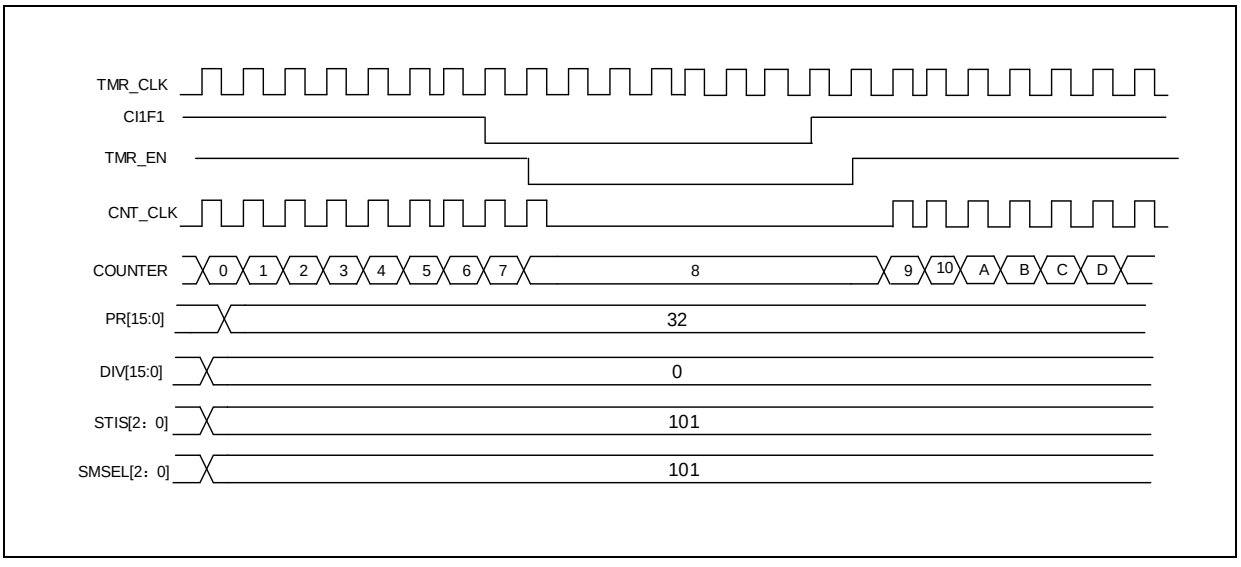
图 10. 复位模式图



挂起模式：

挂起模式下，计数的计数和刹车受选中触发输入信号控制，当触发输入为高电平时计数器开始计数；当为低电平时，计数器暂停计数。

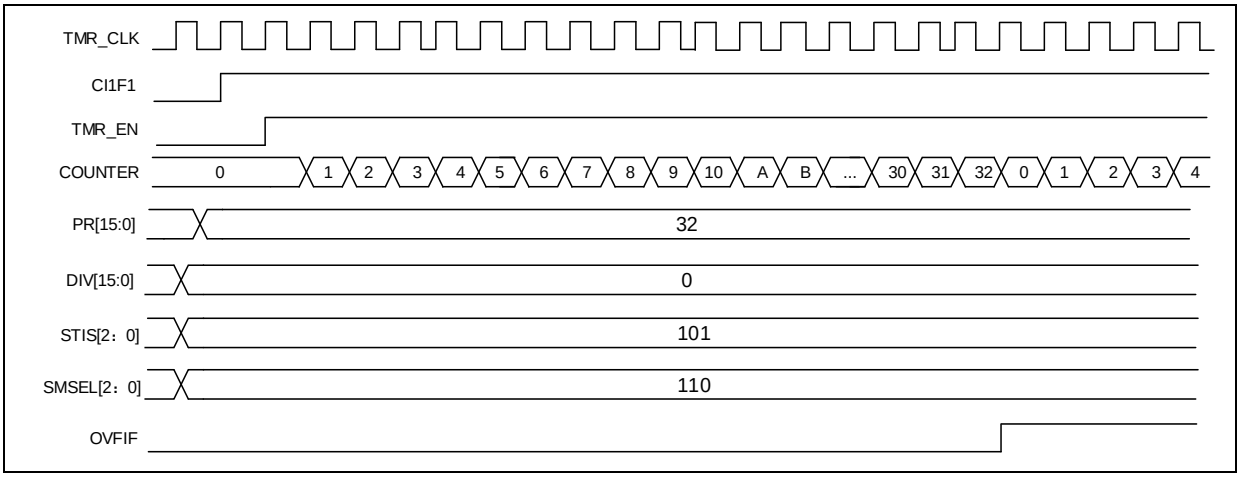
图 11. 挂起模式图



触发模式：

触发模式下，次定时器当受到外部触发信号后，自动启动定时器，即硬件置 TMR_EN 为 1。所以在触发模式下，初始化定时器后软件不需要使能定时器。

图 12. 挂起模式图



本例程实现了 TMR2 同步 TMR3 和 TMR4。主定时器 TMR2 选择溢出事件作为同步信号输出，次定时器 TMR3 和 TMR4 选择挂起模式作为从模式。

10.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\parallel_synchro

10.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入、输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置主模式和从模式
- 配置 PWM 输出模式
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 配置系统时钟 */
   system_clock_config();
  crm_configuration();
```

```
/* get system clock */
crm_clocks_freq_get(&crm_clocks_freq_struct);
/* gpio 初始化 */
gpio_configuration();

/* tmr 基本初始化 */
timerperiod = ((crm_clocks_freq_struct.apb1_freq * 2) / 750000 ) - 1;
tmr_base_init(TMR2, timerperiod, 0);
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);
tmr_clock_source_div_set(TMR2, TMR_CLOCK_DIV1);
tmr_base_init(TMR3, 9, 0);
tmr_cnt_dir_set(TMR3, TMR_COUNT_UP);
tmr_clock_source_div_set(TMR3, TMR_CLOCK_DIV1);
tmr_base_init(TMR4, 4, 0);
tmr_cnt_dir_set(TMR4, TMR_COUNT_UP);
tmr_clock_source_div_set(TMR4, TMR_CLOCK_DIV1);

/* TMR 输出端口初始化 */
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_output_channel_config(TMR2, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_1, timerperiod/5);
tmr_output_channel_config(TMR3, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
tmr_channel_value_set(TMR3, TMR_SELECT_CHANNEL_1, 3);
tmr_output_channel_config(TMR4, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
tmr_channel_value_set(TMR4, TMR_SELECT_CHANNEL_1, 3);
/* 主模式选择: TMR2 */
tmr_sub_sync_mode_set(TMR2, TRUE);
tmr_primary_mode_select(TMR2, TMR_PRIMARY_SEL_OVERFLOW);
/* 从模式选择: TMR3 */
tmr_sub_mode_select(TMR3, TMR_SUB_HANG_MODE);
tmr_trigger_input_select(TMR3, TMR_SUB_INPUT_SEL_IS1);
/*从模式选择: TMR4 */
tmr_sub_mode_select(TMR4, TMR_SUB_HANG_MODE);
tmr_trigger_input_select(TMR4, TMR_SUB_INPUT_SEL_IS1);
```

```
  /* 使能定时器 */
  tmr_counter_enable(TMR2, TRUE);
  tmr_counter_enable(TMR3, TRUE);
  tmr_counter_enable(TMR4, TRUE);

  while(1)
  {}
}
```

10.4 实验效果

- 通过 PA6/PA0/PB6 输出波形，可使用逻辑分析仪抓取波形查看。

11 例 定时器同步-挂起模式

11.1 功能简介

本例程展示了定时器同步功能中的挂起模式，关于定时器的挂起功能介绍可以参考章节“10 例 定时器同步”。

11.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\hang_mode

11.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入、输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置挂起模式
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  at32_board_init();

  /* 获取系统时钟频率 */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* 打开 LED */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 使能 TMR 和 GPIO 时钟 */
  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
```

```
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

/* 引脚配置 */
gpio_init_struct.gpio_pins = GPIO_PINS_8;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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_9;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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_pin_mux_config(GPIOA, GPIO_PINS_SOURCE8, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE9, GPIO_MUX_1);

/* TMR 配置 */
timerperiod = ((crm_clocks_freq_struct.apb2_freq * 2) / 500000 ) - 1;
tmr_base_init(TMR1, timerperiod, 0);
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);

/* 输出配置 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = FALSE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.occ_idle_state = FALSE;
/* 配置通道 1 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, timerperiod/2);
```

```
  /* 配置挂起信号输入 */
  tmr_input_default_para_init(&tmr_input_config_struct);
  tmr_input_config_struct.input_channel_select = TMR_SELECT_CHANNEL_2;
  tmr_input_config_struct.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
  tmr_input_config_struct.input_polarity_select = TMR_INPUT_RISING_EDGE;
  tmr_input_channel_init(TMR1, &tmr_input_config_struct, TMR_CHANNEL_INPUT_DIV_1);

  /* 选择触发输入信号: C2IF2 */
  tmr_trigger_input_select(TMR1, TMR_SUB_INPUT_SEL_C2DF2);

  /* 选择挂起模式 */
  tmr_sub_mode_select(TMR1, TMR_SUB_HANG_MODE);

  /* TMR 输出使能 */
  tmr_output_enable(TMR1, TRUE);

  /* TMR 使能 */
  tmr_counter_enable(TMR1, TRUE);

  while(1)
  {
  }
}
```

11.4 实验效果

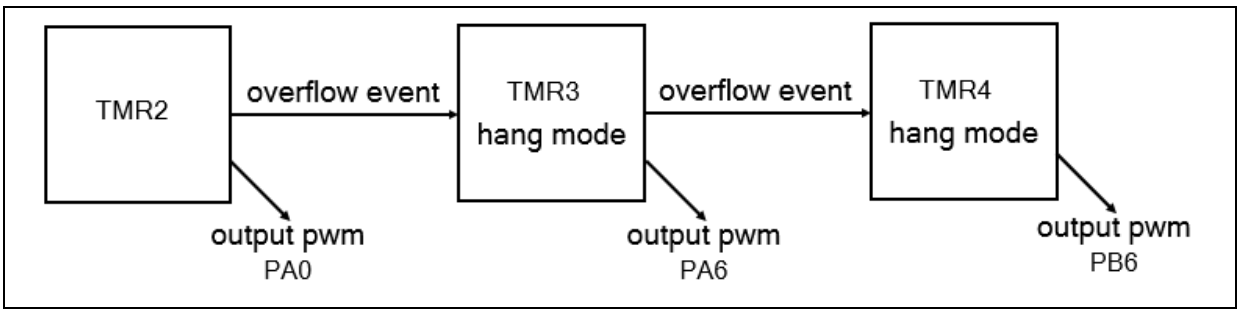
- PA9 用于同步信号输入
- 通过 PA8 输出波形，可使用逻辑分析仪抓取波形查看。

12 例 多定时器级连同步

12.1 功能简介

本例程展示了多个定时器级联同步功能， 关于定时器的同步详细介绍可以参考章节“10 例 定时同步”。

图 13. 级联例程示意图



例程实现了 TMR2 同步 TMR3，TMR3 同步 TMR4。主定时器 TMR2 选择溢出事件作为同步信号输出，次定时器 TMR3 选择挂起模式作为从模式， TMR3 选择溢出事件作为同步信号输出，次定时器 TMR4 选择挂起模式作为从模式。

12.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\cascade_synchro

12.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置主模式和从模式
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  /* 获取系统时钟频率 */
  crm_clocks_freq_get(&crm_clocks_freq_struct);
```

```
at32_board_init();

/* 打开 LED */
at32_led_on(LED2);
at32_led_on(LED3);
at32_led_on(LED4);

/* 使能外设时钟 */
crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_TMR3_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_TMR4_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

/* 引脚配置 */
gpio_init_struct.gpio_pins = GPIO_PINS_6 | GPIO_PINS_0;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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_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(GPIOB, &gpio_init_struct);

gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE6, GPIO_MUX_2);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE6, GPIO_MUX_2);

timerperiod = ((crm_clocks_freq_struct.apb1_freq * 2) / 1000000) - 1;

tmr_base_init(TMR2, timerperiod, 0);
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);
```

```
tmr_base_init(TMR3, 9, 0);
tmr_cnt_dir_set(TMR3, TMR_COUNT_UP);

tmr_base_init(TMR4, 9, 0);
tmr_cnt_dir_set(TMR4, TMR_COUNT_UP);

/* 通道输出配置 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = FALSE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.occ_idle_state = FALSE;
/* TMR2 通道 1 */
tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_1, timerperiod/2);

/* 选择输出到次级定时器信号 */
tmr_primary_mode_select(TMR2, TMR_PRIMARY_SEL_OVERFLOW);
tmr_sub_sync_mode_set(TMR2, TRUE);

/* TMR3 通道 1 */
tmr_output_channel_config(TMR3, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
tmr_channel_value_set(TMR3, TMR_SELECT_CHANNEL_1, 5);

/* 从模式选择 */
tmr_sub_mode_select(TMR3, TMR_SUB_HANG_MODE);
tmr_trigger_input_select(TMR3, TMR_SUB_INPUT_SEL_IS1);

/* 选择输出到次级定时器信号 */
tmr_primary_mode_select(TMR3, TMR_PRIMARY_SEL_OVERFLOW);
tmr_sub_sync_mode_set(TMR3, TRUE);

/* TMR4 通道 1 */
tmr_output_channel_config(TMR4, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
```

```
  tmr_channel_value_set(TMR4, TMR_SELECT_CHANNEL_1, 5);

  /* 从模式选择 */
  tmr_sub_mode_select(TMR4, TMR_SUB_HANG_MODE);
  tmr_trigger_input_select(TMR4, TMR_SUB_INPUT_SEL_IS2);

  /* 使能定时器 */
  tmr_counter_enable(TMR2, TRUE);
  tmr_counter_enable(TMR3, TRUE);
  tmr_counter_enable(TMR4, TRUE);

  while(1)
  {}
}
```

12.4 实验效果

- 通过 PA6/PA0/PB6 输出波形，可使用逻辑分析仪抓取波形查看。

13 例 霍尔信号异或

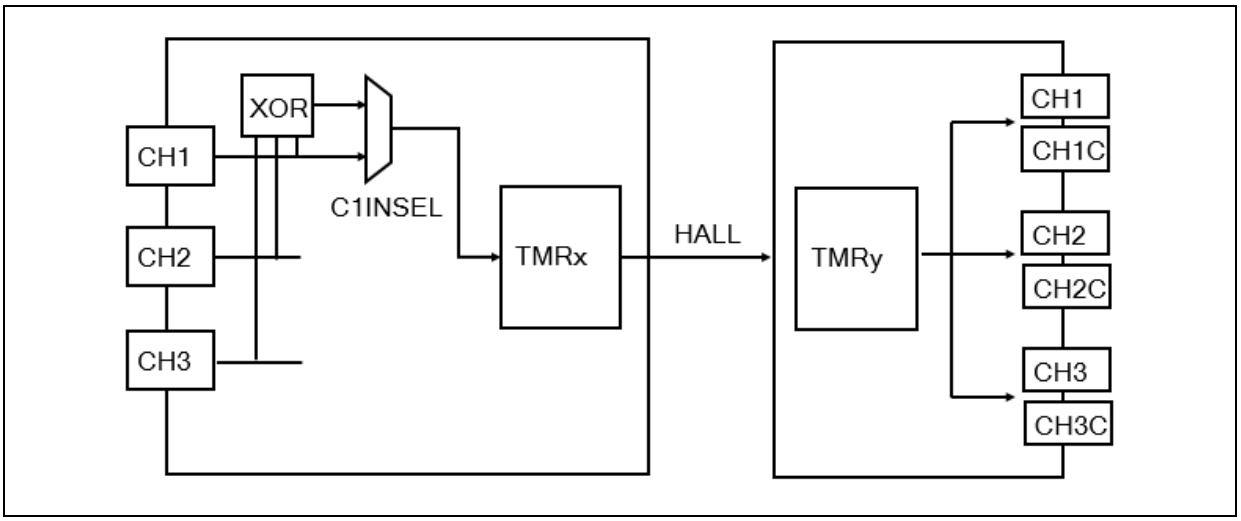
13.1 功能简介

霍尔信号异或功能用于将 3 路霍尔信号异或成 1 路信号，然后将霍尔信号输送到次级定时器 TMRy。TMTy 通过 HALL 信号换相。在 TMRy 中 HALL 信号可以由软件产生。

CH1、CH2、CH3 经异或逻辑后输入主定时器 TMRx，通过分析三路霍尔传感器信号可计算出转子的位置和速度。

当 TMRx_CTRL2 寄存器中的 C1INSEL 位置 1 时开启异或功能。

图 14. 6 异或功能框图



本例程演示了 TMR2 输入 3 路霍尔信号。

13.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\hall_xor_tmr2

13.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入、输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置异或功能
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
```

```
{
  system_clock_config();

  at32_board_init();

  /* 获取系统时钟频率 */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* 打开 LED */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 使能定时器时钟 */
  crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  /* 通道配置 */
  gpio_init_struct.gpio_pins = GPIO_PINS_0 | GPIO_PINS_1 | GPIO_PINS_2;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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);

  /* IO 配置 */
  gpio_init_struct.gpio_pins = GPIO_PINS_3 | GPIO_PINS_6 | GPIO_PINS_7 | GPIO_PINS_8;
  gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
  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_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_1);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE1, GPIO_MUX_1);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE2, GPIO_MUX_1);

  /* TMR 配置 */
```

```
tmr_32_bit_function_enable(TMR2, TRUE);

tmr_base_init(TMR2, 0xFFFFFFFF, 0);
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);

/* 选择输入源 */
tmr_input_config_struct.input_channel_select = TMR_SELECT_CHANNEL_1;
tmr_input_config_struct.input_mapped_select = TMR_CC_CHANNEL_MAPPED_STI;
tmr_input_config_struct.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_input_channel_init(TMR2, &tmr_input_config_struct, TMR_CHANNEL_INPUT_DIV_1);

/* 异或功能使能 */
tmr_channel1_input_select(TMR2, TMR_CHANEL1_2_3_CONNECTED_C1IRAW_XOR);

/* 触发输入选择: C1INC */
tmr_trigger_input_select(TMR2, TMR_SUB_INPUT_SEL_C1INC);

/* 选择从复位模式 */
tmr_sub_mode_select(TMR2, TMR_SUB_RESET_MODE);

tmr_interrupt_enable(TMR2, TMR_TRIGGER_INT, TRUE);

/* 触发中断使能 */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
nvic_irq_enable(TMR2_GLOBAL_IRQn, 1, 0);

/* 使能定时器 */
tmr_counter_enable(TMR2, TRUE);

while(1)
{
  /* 产生霍尔信号 */
  gpio_bits_set(GPIOA, GPIO_PINS_3);
  delay_us(10);
  gpio_bits_set(GPIOA, GPIO_PINS_6);
  delay_us(10);
  gpio_bits_set(GPIOA, GPIO_PINS_7);
  delay_us(10);
```

```
      gpio_bits_reset(GPIOA, GPIO_PINS_3);
      delay_us(10);
      gpio_bits_reset(GPIOA, GPIO_PINS_6);
      delay_us(10);
      gpio_bits_reset(GPIOA, GPIO_PINS_7);
      delay_us(10);
    }
}
```

13.4 实验效果

- PA0 连接 PA3、PA1 连接 PA6、PA2 连接 PA7
- 通过 PA8 输出异或信号，可使用逻辑分析仪抓取波形查看。

14 例　带死区的互补输出

14.1 功能简介

本例程展示了 3 路带死区的互补输出， 关于定时器的 PWM 输出介绍可以参考章节“3 例 PWM 输出”。

死区介绍

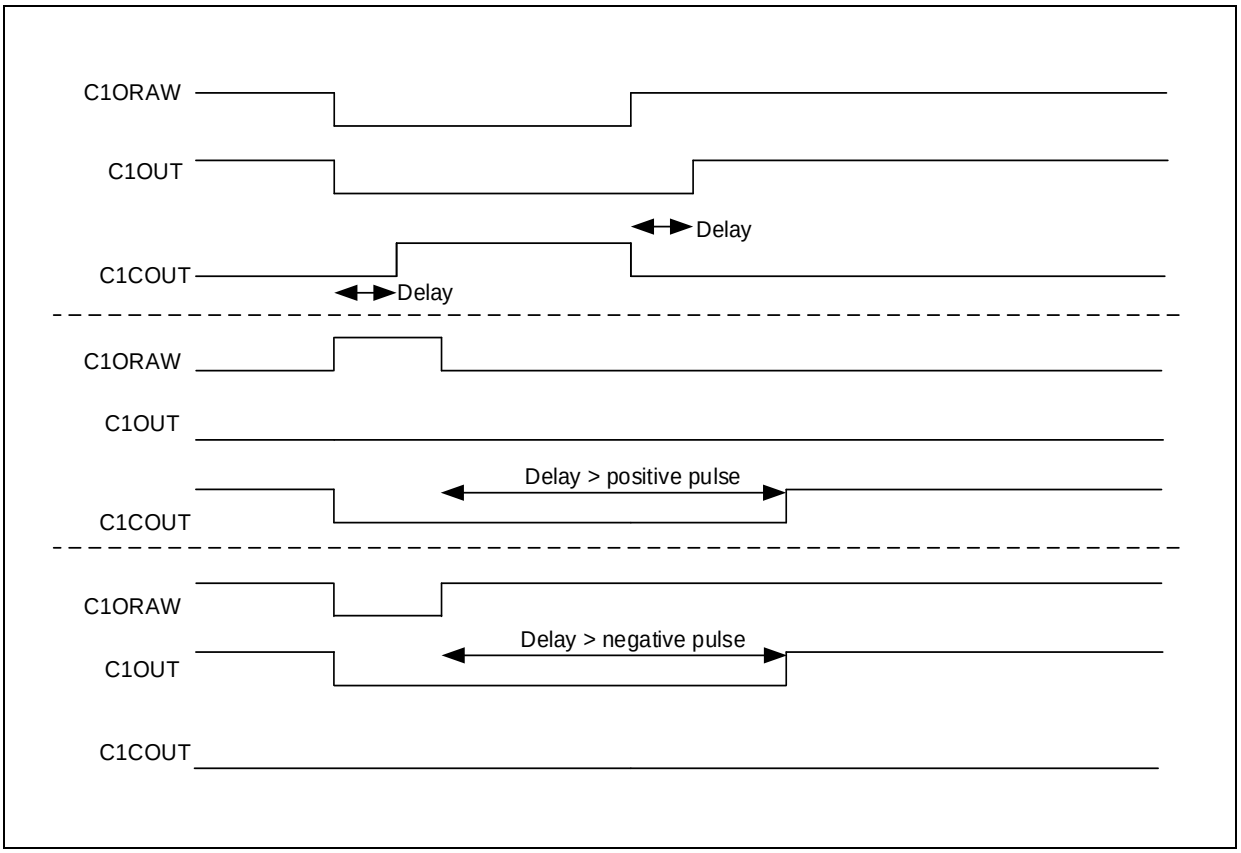
高级定时器通道 1 至 3 包含一组反向通道输出，通过 CxCEN 使能，通过 CxCP 配置极性。

将 CxEN 和 CxCEN 位置 1 后，通过配置 DTC[7：0]死区发生器，可插入不同时长的死区。插入死区后，CxOUT 的上升沿延迟于参考信号的上升沿；CxCOUT 的上升沿延迟于参考信号的下降沿。

如果延迟大于当前有效的输出宽度，C1OUT 和 C1COUT 不会产生相应的脉冲，死区时间应小于有效的输出宽度。

下列图显示了 CxP=0、CxCP=0、OEN=1、CxEN=1 并且 CxCEN=1 时死区插入的例子

图 15. 带死区插入的互补输出



14.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\complementary_signals

14.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置 PWM 互补输出输出
- 配置死区时间
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  at32_board_init();

  /* 获取系统时钟频率 */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* 打开 LED */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 使能外设时钟 */
  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  /* 引脚配置 */
  gpio_init_struct.gpio_pins = GPIO_PINS_8 | GPIO_PINS_9 | GPIO_PINS_10;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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_13 | GPIO_PINS_14 | GPIO_PINS_15;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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(GPIOB, &gpio_init_struct);

gpio_init_struct.gpio_pins = GPIO_PINS_12;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
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(GPIOB, &gpio_init_struct);

gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE8, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE9, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE10, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE13, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE14, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE15, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE12, GPIO_MUX_1);

timerperiod = ((crm_clocks_freq_struct.apb2_freq * 2) / 17570 ) - 1;
channel1pulse = (uint16_t) (((uint32_t) 5 * (timerperiod - 1)) / 10);
channel2pulse = (uint16_t) (((uint32_t) 25 * (timerperiod - 1)) / 100);
channel3pulse = (uint16_t) (((uint32_t) 125 * (timerperiod - 1)) / 1000);

tmr_base_init(TMR1, timerperiod, 0);
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);

/* 通道输出配置 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_B;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = TRUE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_LOW;
```

```
  tmr_output_struct.occ_idle_state = FALSE;
  /* 通道 1 */
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, channel1pulse);
  /* 通道 2 */
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_2, &tmr_output_struct);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_2, channel2pulse);
  /* 通道 3 */
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_3, &tmr_output_struct);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_3, channel3pulse);

  /* 死区相关配置 */
  tmr_brkdt_default_para_init(&tmr_brkdt_config_struct);
  tmr_brkdt_config_struct.brk_enable = TRUE;
  tmr_brkdt_config_struct.auto_output_enable = TRUE;
  tmr_brkdt_config_struct.deadtime = 11;
  tmr_brkdt_config_struct.fcsodis_state = TRUE;
  tmr_brkdt_config_struct.fcsoen_state = TRUE;
  tmr_brkdt_config_struct.brk_polarity = TMR_BRK_INPUT_ACTIVE_HIGH;
  tmr_brkdt_config_struct.wp_level = TMR_WP_LEVEL_3;
  tmr_brkdt_config(TMR1, &tmr_brkdt_config_struct);

  /* TMR 输出使能 */
  tmr_output_enable(TMR1, TRUE);
  /* TMR 使能 */
  tmr_counter_enable(TMR1, TRUE);

  while(1)
  {}
}
```

14.4 实验效果

- 通过 PA8/PA9/PA10、PB13/PB14/PB15 输出波形，可使用逻辑分析仪抓取波形查看。

15 例 6 步方波

15.1 功能简介

6 步方波主要用于控制三相全桥驱动无刷电机旋转，在有感无刷电机中，软件根据由霍尔信号进行换相操作。通常在使用中，霍尔传感器连接 TMRx 的 CH1、CH2、CH3，通过分析三路霍尔传感器信号可计算出转子的位置和速度。经过异或后将 HALL 信号输出给 TMRy，TMTy 通过 HALL 信号换相。此外 HALL 信号可以由软件产生。

图 16. 6 步方波信号流程图

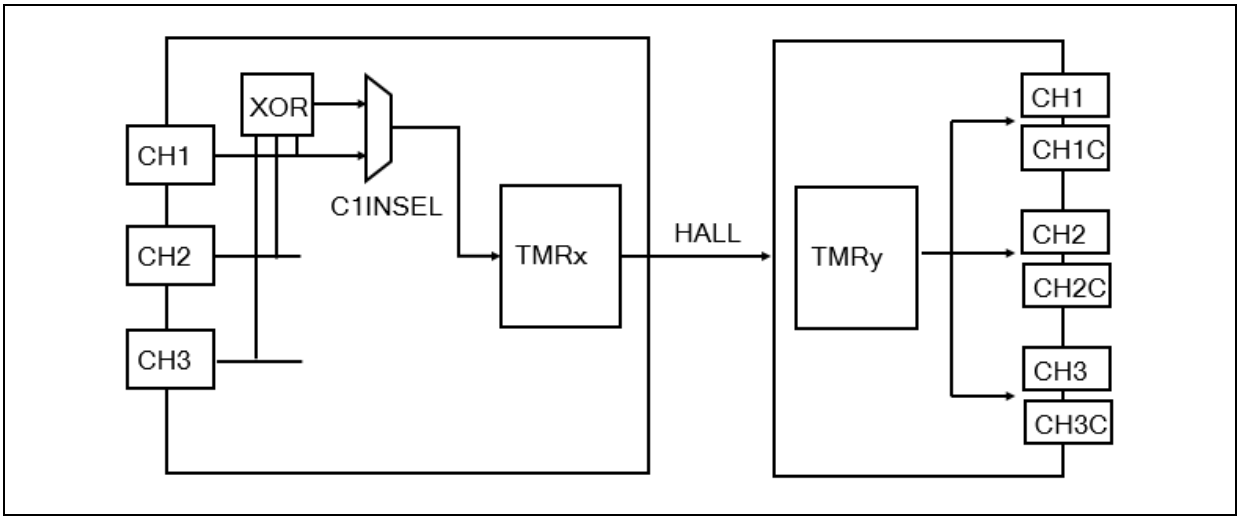


表 2. 换相步骤

	步骤 1	步骤 2	步骤 3	步骤 4	步骤 5	步骤 6
CH1	1	0	0	0	0	1
CH1C	0	0	1	1	0	0
CH2	0	0	0	1	1	0
CH2C	1	1	0	0	0	0
CH3	0	1	1	0	0	0
CH3C	0	0	0	0	1	1

本例程实现了使用 TMR1 通过软件产生 HALL 事件，完成 6 步方波的控制。

15.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\ 6_steps

15.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置互补输出管脚

- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置互补 PWM 输出
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  at32_board_init();

  /* 获取系统时钟频率 */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  SysTick_Configuration();

  /* 打开 LED */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 外设时钟使能 */
  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  /* IO 配置 */
  gpio_init_struct.gpio_pins = GPIO_PINS_8 | GPIO_PINS_9 | GPIO_PINS_10;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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_13 | GPIO_PINS_14 | GPIO_PINS_15;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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(GPIOB, &gpio_init_struct);

gpio_init_struct.gpio_pins = GPIO_PINS_12;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_DOWN;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOB, &gpio_init_struct);

gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE8, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE9, GPIO_MUX_1);
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE10, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE13, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE14, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE15, GPIO_MUX_1);
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE12, GPIO_MUX_1);

/* TMR 基本配置 */
tmr_base_init(TMR1, 4095, 0);
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);

/* 通道输出模式配置 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_OFF;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = TRUE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.occ_idle_state = TRUE;
/* 通道 1 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 2047);
/* 通道 2 */
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_2, &tmr_output_struct);
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_2, 1023);
```

```
  /* 通道 3 */
  tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_3, &tmr_output_struct);
  tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_3, 511);

  /* 死区相关配置 */
  tmr_brkdt_default_para_init(&tmr_brkdt_config_struct);
  tmr_brkdt_config_struct.brk_enable = TRUE;
  tmr_brkdt_config_struct.auto_output_enable = TRUE;
  tmr_brkdt_config_struct.deadtime = 0;
  tmr_brkdt_config_struct.fcsodis_state = TRUE;
  tmr_brkdt_config_struct.fcsoen_state = TRUE;
  tmr_brkdt_config_struct.brk_polarity = TMR_BRK_INPUT_ACTIVE_HIGH;
  tmr_brkdt_config_struct.wp_level = TMR_WP_OFF;
  tmr_brkdt_config(TMR1, &tmr_brkdt_config_struct);
  tmr_channel_buffer_enable(TMR1, TRUE);

  /* 霍尔中断使能 */
  tmr_interrupt_enable(TMR1, TMR_HALL_INT, TRUE);

  /* 霍尔中断 NVIC 使能 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  nvic_irq_enable(TMR1_TRG_HALL_TMR11_IRQn, 1, 0);

  /* TMR 输出使能 */
  tmr_output_enable(TMR1, TRUE);
  /* TMR 使能 */
  tmr_counter_enable(TMR1, TRUE);

  while(1)
  {}
}
```

15.4 实验效果

- 通过 PA8/PA9/PA10、PB13/PB14/PB15 输出波形，可使用逻辑分析仪抓取波形查看。

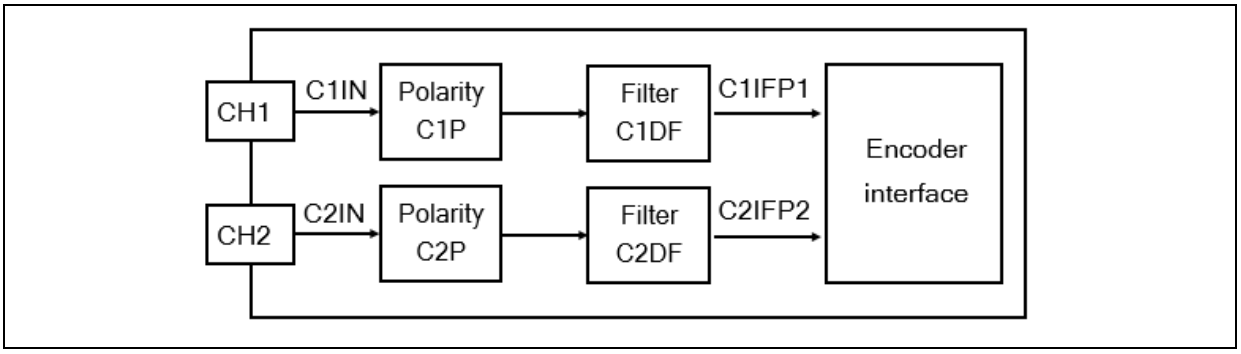
16 例　编码器输入

16.1 功能简介

编码模式下需提供两个输入（C1IN/C2IN），根据一个输入的电平值，计数器将在另一个输入的边沿向上或向下计数。计数方向将由 OWCDIR 值指示，编码器计数模式分为以下几种

- 编码模式 A（SMSEL = 1）：根据 C2IFP2 电平，在 C1IFP1 边沿计数
- 编码模式 B（SMSEL = 2）：根据 C1IFP1 电平，在 C2IFP2 边沿计数
- 编码模式 C（SMSEL = 3）：根据 C2IFP2 电平，在 C1IFP1 边沿计数，根据 C1IFP1 电平，在 C2IFP2 边沿计数

图 17. 编码器接口框图



输入信号极性可以使用 TMRx_CCTRL 的 C1P/C2P 进行配置，滤波器使用 TMRx_CM1 的 C1DF/C2DF 进行配置。

表 3. 计数方向与编码器信号的关系

有效边沿	相对信号的电平（C1IFP1 对应 C2IN，C2IFP2 对应 C1IN）	C1IFP1 信号		C2IFP2 信号	
		上升	下降	上升	下降
仅在 C1IN 计数	高	向下计数	向上计数	不计数	不计数
	低	向上计数	向下计数	不计数	不计数
仅在 C2IN 计数	高	不计数	不计数	向上计数	向下计数
	低	不计数	不计数	向下计数	向上计数
在 C1IN 和 C2IN 上计数	高	向下计数	向上计数	向上计数	向下计数
	低	向上计数	向下计数	向下计数	向上计数

本例程演示了如何使用编码器接口。

16.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\encoder_tmr2

16.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入、输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置编码器功能
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  at32_board_init();

  /* 获取系统时钟频率 */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* 打开 LED */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 使能定时器/gpioa clock */
  crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  /* 引脚配置 */
  gpio_init_struct.gpio_pins = GPIO_PINS_0 | GPIO_PINS_1;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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_2 | GPIO_PINS_3;
  gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
```

```
  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_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_1);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE1, GPIO_MUX_1);

  /* 使能定时器 32bit function */
  tmr_32_bit_function_enable(TMR2, TRUE);

  tmr_base_init(TMR2, 0xFFFFFFFF, 0);
  tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);

  /* 配置编码器模式 */
  tmr_encoder_mode_config(TMR2, TMR_ENCODER_MODE_C, TMR_INPUT_RISING_EDGE,
TMR_INPUT_RISING_EDGE);

  /* 使能定时器 */
  tmr_counter_enable(TMR2, TRUE);

  while(1)
  {
    /* 产生编码器信号 */
    gpio_bits_set(GPIOA, GPIO_PINS_2);
    delay(150);
    gpio_bits_set(GPIOA, GPIO_PINS_3);
    delay(150);
    gpio_bits_reset(GPIOA, GPIO_PINS_2);
    delay(150);
    gpio_bits_reset(GPIOA, GPIO_PINS_3);
    delay(150);

    /* 获取当前计数器值 */
    counter = tmr_counter_value_get(TMR2);
  }
}
```

16.4 实验效果

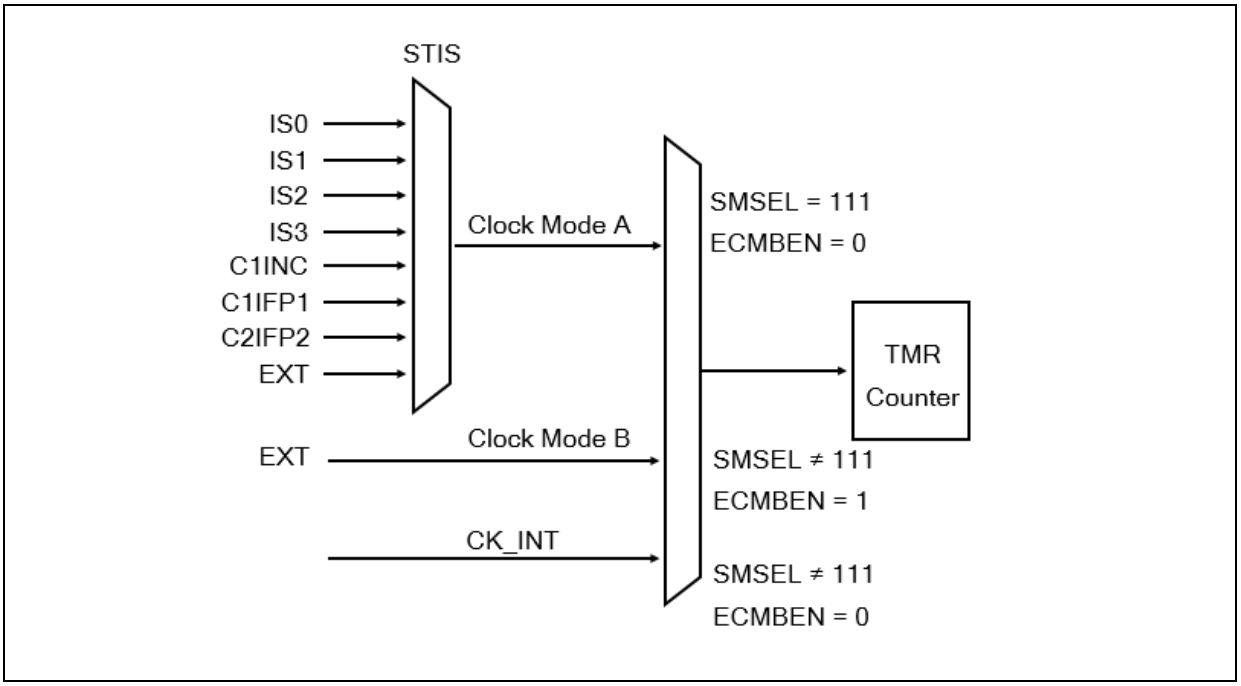
- 连接 PA0 与 PA2、PA1 与 PA3（PA2、PA3 模拟编码器信号，PA0、PA1 为编码器接口）
- 在 DEBUG 模式下，观看 TMR 计数器。

17 例 外部时钟模式 A

17.1 功能简介

定时器计数时钟可从内部时钟（CK_INT）、外部时钟（外部时钟模式 A、B）、内部触发输入（TRGIN）这些时钟源提供。

图 18. 定时器时钟源



本例程实现了 TMR2_CH1 产生 PWM 信号，输入到 TMR3 的 CH2 作为 TMR3 的时钟。

17.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\external_clock

17.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输入、输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置定时器时钟
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  at32_board_init();

  /* 获取系统时钟频率 */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* 打开 LED */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* 使能定时器时钟 */
  crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_TMR3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  /* IO 配置 */
  gpio_init_struct.gpio_pins = GPIO_PINS_0;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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(GPIOB, &gpio_init_struct);

  gpio_init_struct.gpio_pins = GPIO_PINS_7;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  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_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_1);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE7, GPIO_MUX_2);
```

```
gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE0, GPIO_MUX_2);

/* use tmr2 generate clock to drive tmr3----------
tmr2 pwm frequency = 288000000(apb1_freq*2)/240(period+1)/1000(div+1) = 1 khz
tmr3 pwm frequency = 1000(tmr2 pwm frequency)/100(period+1)/1(div+1) = 10 hz
---------- */
/* TMR 配置 */
timerperiod = ((crm_clocks_freq_struct.apb1_freq * 2) / 1000000) - 1;
tmr_base_init(TMR2, timerperiod, 999);
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);

tmr_base_init(TMR3, 99, 0);
tmr_cnt_dir_set(TMR3, TMR_COUNT_UP);

/* 输出配置 */
tmr_output_default_para_init(&tmr_output_struct);
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = FALSE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.occ_idle_state = FALSE;
/* TMR2 通道 1 */
tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_1, timerperiod/2);
/* TMR3 通道 3 */
tmr_output_channel_config(TMR3, TMR_SELECT_CHANNEL_3, &tmr_output_struct);
tmr_channel_value_set(TMR3, TMR_SELECT_CHANNEL_3, 50);

/* 配置 TMR3 通道 2 */
tmr_input_config_struct.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_input_config_struct.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_input_config_struct.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_input_channel_init(TMR3, &tmr_input_config_struct, TMR_CHANNEL_INPUT_DIV_1);

/* 选择触发输入信号: C2IF2 */
tmr_trigger_input_select(TMR3, TMR_SUB_INPUT_SEL_C2DF2);
```

```
  /* 选择外部时钟模式 A*/
  tmr_sub_mode_select(TMR3, TMR_SUB_EXTERNAL_CLOCK_MODE_A);

  /* 使能定时器 */
  tmr_counter_enable(TMR2, TRUE);

  /*使能定时器*/
  tmr_counter_enable(TMR3, TRUE);

  while(1)
  {
  }
}
```

17.4 实验效果

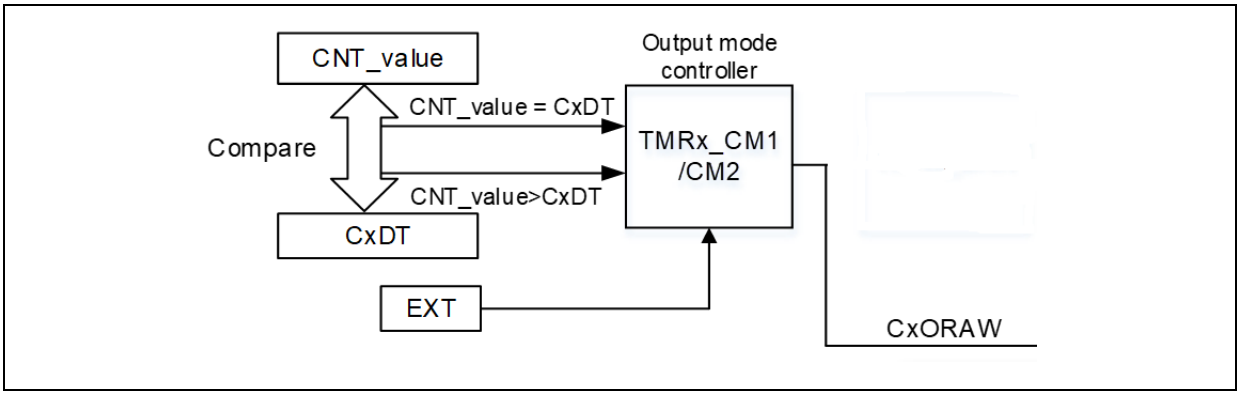
- 连接 PA0 与 PA7，PB0 输出波形，可使用逻辑分析仪抓取波形查看。

18 例 输出比较-匹配时 CxORAW 输出高/低

18.1 功能简介

本例程演示了当 CxDT 值和 CNT_value 匹配时输出高和低。

图 19. 输出比较框图



18.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\oc_high（匹配时输出高，下文用此例介绍）

project\at_start_xxx\examples\tmr\oc_low（匹配时输出低）

18.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置输出比较
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  system_clock_config();

  at32_board_init();
```

```
crm_apb1_div_set(CRM_APB1_DIV_16);

/* 获取系统时钟频率 */
crm_clocks_freq_get(&crm_clocks_freq_struct);

crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);

nvic_irq_enable(TMR2_GLOBAL_IRQn, 0, 1);

gpio_configuration();

/* 计算分频值 */
prescalervalue = (uint16_t) ((crm_clocks_freq_struct.apb1_freq * 2) / 1000) - 1;

/* 定时器基础配置 */
tmr_base_init(TMR2, 0xFFFF, prescalervalue);
tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);
tmr_clock_source_div_set(TMR2, TMR_CLOCK_DIV1);

tmr_output_default_para_init(&tmr_oc_init_structure);
tmr_oc_init_structure.oc_mode = TMR_OUTPUT_CONTROL_HIGH;
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_output_channel_config(TMR2, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_1, ccr1_val);

tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_2, &tmr_oc_init_structure);
tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_2, ccr2_val);

tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_3, &tmr_oc_init_structure);
tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_3, ccr3_val);

tmr_output_channel_config(TMR2, TMR_SELECT_CHANNEL_4, &tmr_oc_init_structure);
tmr_channel_value_set(TMR2, TMR_SELECT_CHANNEL_4, ccr4_val);
```

```
  tmr_period_buffer_enable(TMR2, TRUE);

  /* 定时器使能 */
  tmr_interrupt_enable(TMR2, TMR_C1_INT | TMR_C2_INT | TMR_C3_INT | TMR_C4_INT, TRUE);

  /* 配置 IO */
  gpio_bits_set(GPIOC, GPIO_PINS_6 | GPIO_PINS_7 | GPIO_PINS_8 | GPIO_PINS_9);

  /* 定时器使能 */
  tmr_counter_enable(TMR2, TRUE);

  while(1)
  {
  }
}
```

18.4 实验效果

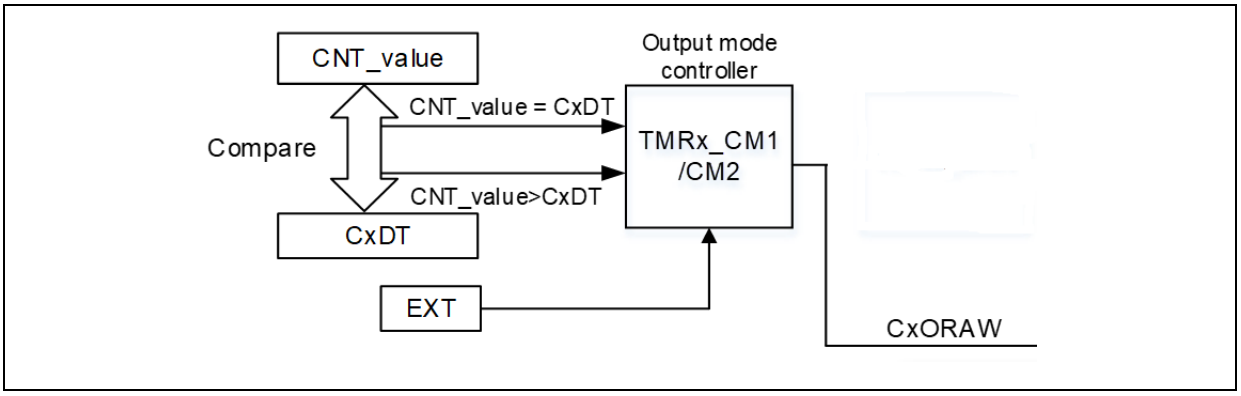
- 通过 PC6/ PC7/ PC8/ PC9 输出波形，可使用逻辑分析仪抓取波形查看。

19 例 输出比较-匹配时 CxORAW 翻转

19.1 功能简介

本例程演示了当 CxDT 值和 CNT_value 匹配时 CxORAW 翻转电平。

图 20. 输出比较框图



19.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_xxx\examples\tmr\oc_toggle_tmr3（TMR3 例程，下文用此例介绍）

project\at_start_xxx\examples\tmr\oc_toggle_tmr9（TMR9 例程）

19.3 软件设计

1) 配置流程

- 开启定时器外设时钟
- 配置输出管脚
- 配置定时器 TMRx_DIV 寄存器和 TMRx_PR 寄存器
- 配置输出比较
- 使能定时器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  crm_configuration();

  /* 获取系统时钟频率 */
```

```
crm_clocks_freq_get(&crm_clocks_freq_struct);

/* NVIC 配置 */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
nvic_irq_enable(TMR3_GLOBAL_IRQn, 1, 0);

/* GPIO 配置 */
gpio_configuration();

/* 计算分频 */
prescalervalue = (uint16_t) ((crm_clocks_freq_struct.apb1_freq * 2) / 24000000) - 1;

/* 定时器基本配置 */
tmr_base_init(TMR3, 65535, prescalervalue);
tmr_cnt_dir_set(TMR3, TMR_COUNT_UP);
tmr_clock_source_div_set(TMR3, TMR_CLOCK_DIV1);

/* 配置比较翻转模式:通道 1 */
tmr_output_default_para_init(&tmr_oc_init_structure);
tmr_oc_init_structure.oc_mode = TMR_OUTPUT_CONTROL_SWITCH;
tmr_oc_init_structure.oc_idle_state = FALSE;
tmr_oc_init_structure.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;
tmr_oc_init_structure.oc_output_state = TRUE;
tmr_output_channel_config(TMR3, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
tmr_channel_value_set(TMR3, TMR_SELECT_CHANNEL_1, ccr1_val);

/* 配置比较翻转模式: 通道 2 */
tmr_output_channel_config(TMR3, TMR_SELECT_CHANNEL_2, &tmr_oc_init_structure);
tmr_channel_value_set(TMR3, TMR_SELECT_CHANNEL_2, ccr2_val);

/* 配置比较翻转模式: 通道 3 */
tmr_output_channel_config(TMR3, TMR_SELECT_CHANNEL_3, &tmr_oc_init_structure);
tmr_channel_value_set(TMR3, TMR_SELECT_CHANNEL_3, ccr3_val);

/* 配置比较翻转模式: 通道 4 */
tmr_output_channel_config(TMR3, TMR_SELECT_CHANNEL_4, &tmr_oc_init_structure);
tmr_channel_value_set(TMR3, TMR_SELECT_CHANNEL_4, ccr4_val);
```

```
  /* 定时器使能 */
  tmr_counter_enable(TMR3, TRUE);

  /* 中断使能 */
  tmr_interrupt_enable(TMR3, TMR_C1_INT | TMR_C2_INT | TMR_C3_INT | TMR_C4_INT, TRUE);

  while(1)
  {}
}
```

19.4 实验效果

- 通过 PB0\PB1\PA6\PA7 输出波形，可使用逻辑分析仪抓取波形查看。

20 文档版本历史

表 4. 文档版本历史

日期	版本	变更
2021.5.20	2.0.0	最初版本
2021.11.30	2.0.1	增加示例
2022.9.2	2.0.2	PWM输入捕获案例，频率和占空比计算方法修改
2022.10.20	2.0.3	增加案例11~19

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。