32 位微控制器

HC32F4A0 系列的通用定时器

TIMERA

适用对象

F 系列	**HC32F4A0**

目　录

1 摘要

本篇应用笔记主要介绍 HC32F4A0 系列的通用定时器 TimerA 模块的功能原理及基本的应用流程。

2 TimerA 简介

通用定时器 A（TimerA）是一个计数宽度为 16 位的定时器，其基本特性及功能如表 2-1 所示。

波形模式	锯齿波、三角波
时钟源	软件时钟源（PCLK0/PCLK1）、硬件时钟源
基本功能	递加、递减计数方向
	同步启动计数器
	基准值缓存功能
	32 位级联计数
	正交编码计数
	4 路 PWM 输出
	比较匹配事件输出
中断类型	比较匹配中断
	周期匹配中断

表 2-1 TimerA 基本特定及功能

3 TimerA 功能及应用

3.1 引脚描述

HC32F4A0 系列 MCU 的 TimerA 具有如表 2-2 所示的多种端口，通过不同的配置而实现不同的功能，可用于多种不同的应用场景。

端口名称	方向	功能
TIMA_<t>PWMn	输入或输出	输入事件的捕获端口或 PWM 输出端口（n=1~4；t=1~12 为单元编号，下同）
TIMA_<t>CLKA	输入	正交编码计数事件的输入端口
TIMA_<t>CLKB		
TIMA_<t>TRIG	输入	硬件触发启动、停止、清零事件的输入端口

表 2-2 TimerA 端口

在使用这些端口时，应使用 API GPIO_SetFunc() 将端口设置为对应的功能。如，例程 timera_base_timer，将 TIMA_1_TRIG （PC0）的上升沿用作 TimerA 单元 1 的启动条件，将其下降沿用作停止条件，当 PC0 用作 TIMA_1_TRIG 时，其对应的功能号是 6。应当注意的是，不同单元的端口，或者不同功能的端口，其功能号可能不同，具体请参考用户手册“引脚配置及功能”章节的“引脚功能表”。

3.2 计数波形模式

TimerA 有两种基本的计数波形模式，锯齿波和三角波。在实际应用中，根据需求选择合适的计数波形模式，并根据选择的计数波形模式，计算相应的比较值和周期值。

锯齿波模式和向上计数。计数器启动后，若初始值大于周期值寄存器的值，则初始值寄存器的值会被清零。计数器递加计数，当递加到与指定通道的比较值寄存器的值相等时，再递加一次，该通道的比较匹配标志位置位以及产生比较匹配中断（若比较匹配中断被使能）；当递加到与周期值寄存器的值相等时，再递加一次，则单元的计数上溢标志位置位以及产生计数上溢中断（若计数上溢中断被使能），计数值寄存器的值更新为 0。

锯齿波模式和向下计数。计数器启动后，若计数值寄存器的值大于周期值寄存器的值，计数值寄存器的值会被更新为周期值寄存器的值。计数器做递减计数，当递减到与指定通道的比较值寄存器的值相等时，再递减一次，则该通道的比较匹配标志位置位以及产生比较匹配中断（若比较匹配中断被使能）；当递减到 0 时，再递减一次，则单元的计数下溢标志位置位

以及产生计数下溢中断（若计数下溢中断被使能），计数值寄存器的值更新为周期值寄存器的值。

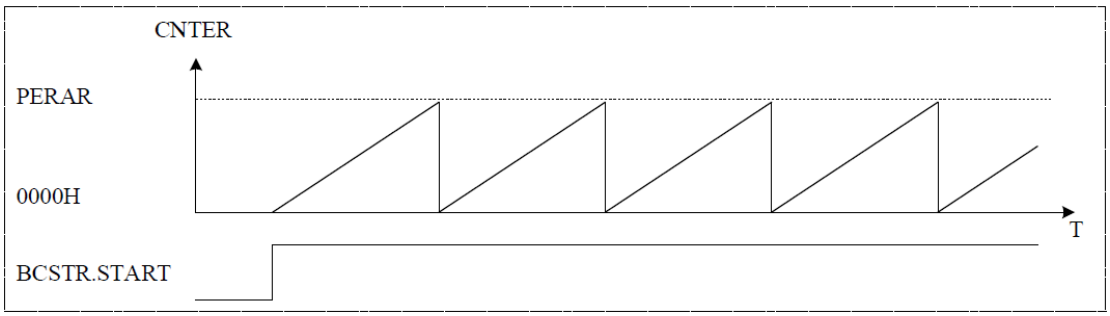


图 3-1 锯齿波波形（向上计数）

在用作基本定时器时，锯齿波模式的周期值计算公式如下：

TmrAPeriodVal=(TmrAPeriod×[TmrAClockSource÷TmrAClockPrescaler])－1

其中，TmrAPeriodVal 为周期值寄存器的初始值，TmrAPeriod 为定时时间（单位微秒），TmrAClockSource 为时钟源频率（单位 MHz），TmrAClockPrescaler 为时钟源分频数。

三角波模式。在三角波模式下，计数方向的设置被忽略，但可通过读取相关寄存器获取当前计数方向。计数器启动后，如果计数值寄存器的初始值大于等于周期值寄存器的值，则先从周期值开始做递减计数；如果计数值寄存器的初始值小于周期值寄存器的值，则从初始值开始做递加计数。以下的说明，基于计数器寄存器的初始值小于周期值寄存器的值。

三角波模式下计数器启动后，计数值寄存器递加，当递加到与指定通道的比较值寄存器的值相等时，再递加一次，该通道的比较匹配标志位置位以及产生比较匹配中断（若比较匹配中断被使能）；当递加到与周期值寄存器的值相等时，再递加一次，则单元的计数上溢标志位置位以及产生计数上溢中断（若计数上溢中断被使能）；此时，计数值寄存器的值因为上溢而更新为周期值寄存器的值减 1，计数器做递减计数，当递减到与指定通道的比较值寄存器的值相等时，再递减一次，则该通道的比较匹配标志位置位以及产生比较匹配中断（若比较匹配中断被使能）；当递减到 0 时，再递减一次，则单元的计数下溢标志位置位以及产生计数下溢中断（若计数下溢中断被使能）；此时，计数值寄存器的值因为下溢而更新为 1；计数器做递加计数。

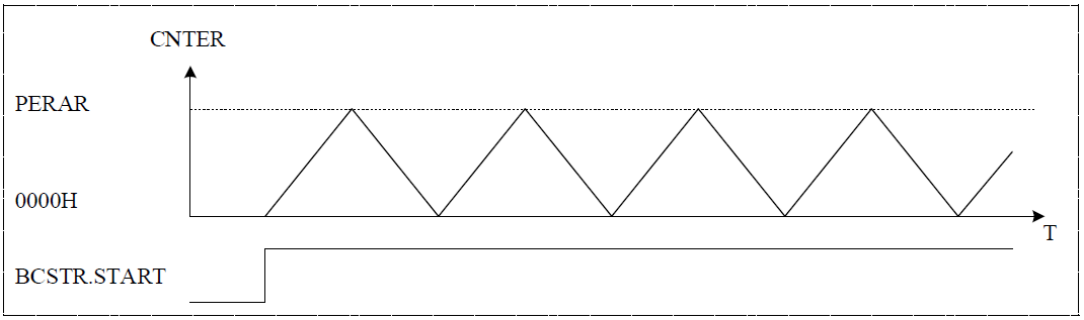


图 3-2 三角波波形

在用作基本定时器时，三角波模式的周期值计算公式如下：

TmrAPeriodVal=(TmrAPeriod×[TmrAClockSource÷TmrAClockPrescaler])÷2

其中，TmrAPeriodVal 为周期值寄存器的初始值，TmrAPeriod 为定时时间（单位微秒），TmrAClockSource 为时钟源频率（单位 MHz），TmrAClockPrescaler 为时钟源分频数。

例程 timera_base_timer 即是将 TimerA 用作基本定时器，实现了 1 毫秒的定时。在该例程中，通过配置，可分别使用锯齿波和三角波计数波形模式。

3.3 时钟源

TimerA 的计数时钟源有如下几种：

a) PCLK 的 1、2、4、8、16、32、64、128、256、512、1024 分频，其中 PCLK0 用于单元 1~单元 4，PCLK1 用于单元 5~单元 12；

b) TIMA_<t>_TRIG 端口输入的上升沿、下降沿；

c) 片上外设产生的事件，如 ADC 转换结束事件、DMA 传输结束事件等；

d) 对称单元的计数上溢事件或计数下溢事件（单元 1 和单元 2 互为对称单元、单元 3 和单元 4 互为对称单元，单元 5 和单元 6 互为对称单元……）；

e) 端口 TIMA_<t>_CLKA 和 TIMA_<t>_CLKB 输入的正交编码信号。

计数时钟源选择 a 时为软件计数模式。当时钟源为 a 时：

1) 当计数波形模式为锯齿波时，可设置计数方向为向上或向下；

2) 当计数波形模式为三角波时，计数方向的设置无效。

计数时钟源选择 b、c、d、e 时为硬件计数模式。当时钟源为 b、c、d、e 时：

1) 时钟源 a 自动无效；

2) 计数波形模式的设置无效；

3) 计数方向的设置无效，但可获取当前的计数方向；

4) 它们相互独立，可多个同时设置为有效。

应用时，根据实际需求选择合适的时钟源。如，将 TimerA 用作基本定时功能时，一般选用 a 作为时钟源；当用作两相正交编码计数时，选择 e 作为时钟源；等等。

3.4 比较输出

比较输出，即计数值寄存器与比较基准值寄存器、周期值寄存器比较匹配后，可从端口 TIMA_<t>_PWMn 输出各自指定的电平。每个 TimerA 单元内部均含有 4 个通道的比较输出端口，均可在比较匹配时输出指定的电平。

TIMA_<t>_PWMn 端口的计数开始时的电平、计数停止时的电平、计数比较匹配时的电平、计数周期匹配时的电平等，可通过端口控制寄存器（PCONRn）的 STAC、STPC、CMPC、PERC、FORC 位设定控制（n=1~4）。图 3-3 为比较输出的时序示例。

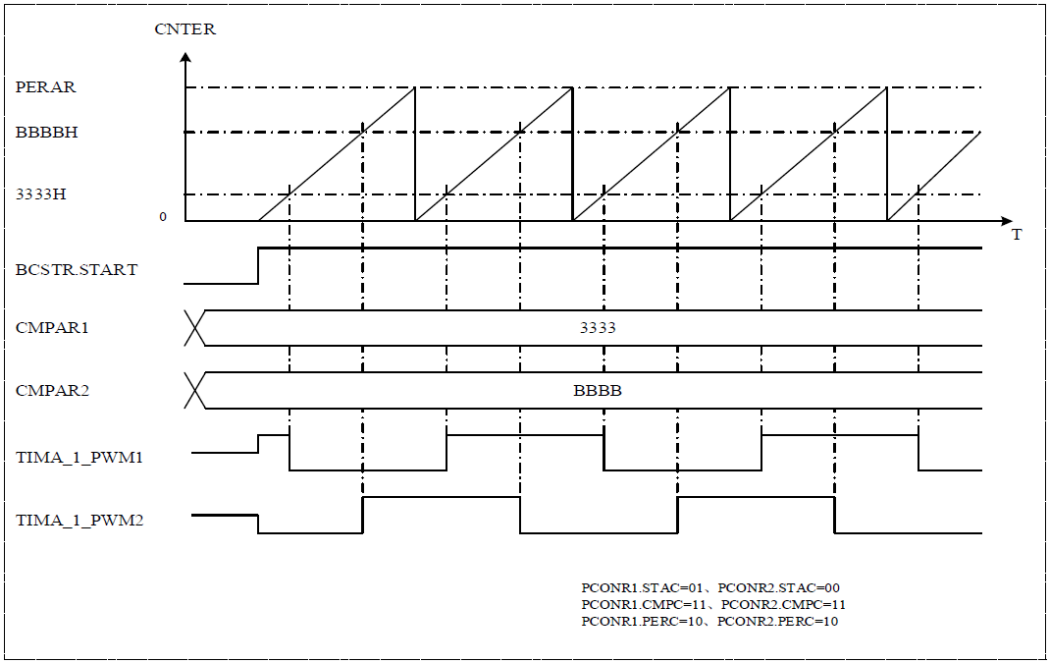


图 3-3 比较输出的时序

比较输出的实现方法，请参考例程 timera_pwm。

3.5 捕获输入

输入捕获，即捕获输入的各种事件。这些事件包括（请参考用户手册关于寄存器 CCONR 的描述）：

1) 端口 TIMA_<t>_PWMn 的上升沿或下降沿；

2) 端口 TIMA_<t>_TRIG 的上升沿或下降沿；

3) 片上外设产生的事件（由寄存器 TMRA_HTSSR 指定）。

当通道指定的捕获事件产生后，当前计数值寄存器的值就被保存到该通道的比较值寄存器中（CMPARn）。图 3-4 为捕获输入的动时序示例。

例程 timera_capture，介绍了 TimerA 捕获输入这个功能的实现方法。在该例程中，通过宏“APP_CAPTURE_CONDITION”可指定不同的捕获事件或同时指定多个捕获事件，当指定事件有一个发生时，即满足捕获条件，TimerA 执行捕获动作，计数值寄存器的值即被保存到指定通道的比较值寄存器中，应用时，可根据需求获取并使用捕获值。

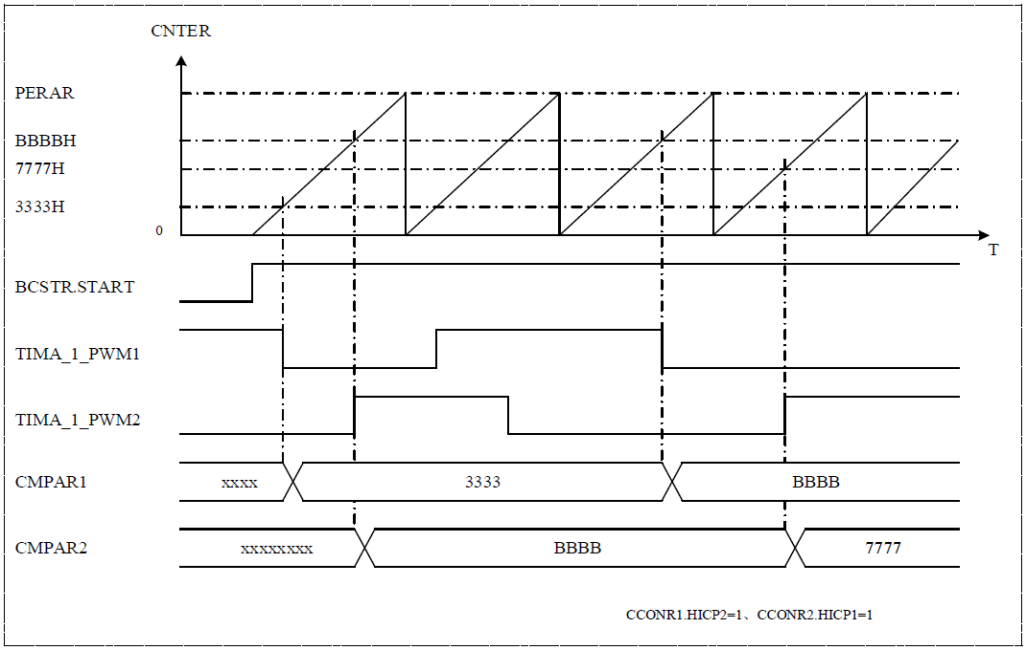


图 3-4 捕获输入的时序

3.6 同步启动

互为对称单元的 TimerA 可以实现软件同步启动或硬件同步启动。当使能对称单元的偶数单元的同步启动功能后，启动奇数单元会同时启动偶数单元。

软件同步启动。对称单元的两个单元均使用 PCLK 作为时钟源，且偶数单元使能同步启动功能，如图 3-5 所示。

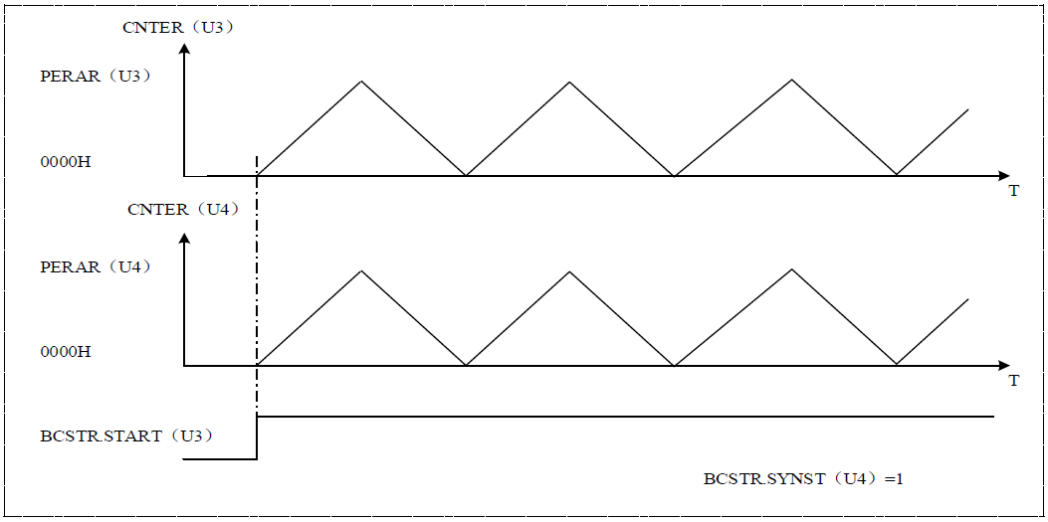


图 3-5 软件同步启动

硬件同步启动。硬件启动条件只支持 TIMA_<t>_TRIG 端口输入的上升沿和下降沿，通过寄存器位 HCONR.HSTA0 和 HCONR.HSTA1 指定（详见用户手册关于该寄存器的说明）。若要使用硬件同步启动，互为对称单元的两个单元的寄存器位 HCONR.HSTA0 和 HCONR.HSTA1 应作相同设置。

在级联计数例程 timera_cascade_count 中，就用到了软件同步启动，详细代码请参考该例程。

3.7 数字滤波

TimerA 的各种输入端口，都有数字滤波功能，这些端口包括：

1) TIMA_<t>_CLKA，TIMA_<t>_CLKB；

2) TIMA_<t>_TRIG；

3) TIMA_<t>_PWMn（用作输入捕获功能时）。

各端口的滤波功能的使能和滤波时钟的选择可通过设定滤波控制寄存器（FCONR）和捕获控制寄存器（CCONR）的对应位来实现，详见用户手册关于寄存器 FCONR 和 CCONR 的描述。

滤波机制。在滤波采样基准时钟采样到端口上 3 次一致的电平时，该电平被当作有效电平传送到模块内部；小于 3 次一致的电平会被当作外部干扰滤掉，不传送到模块内部，如图 3-6 所示。

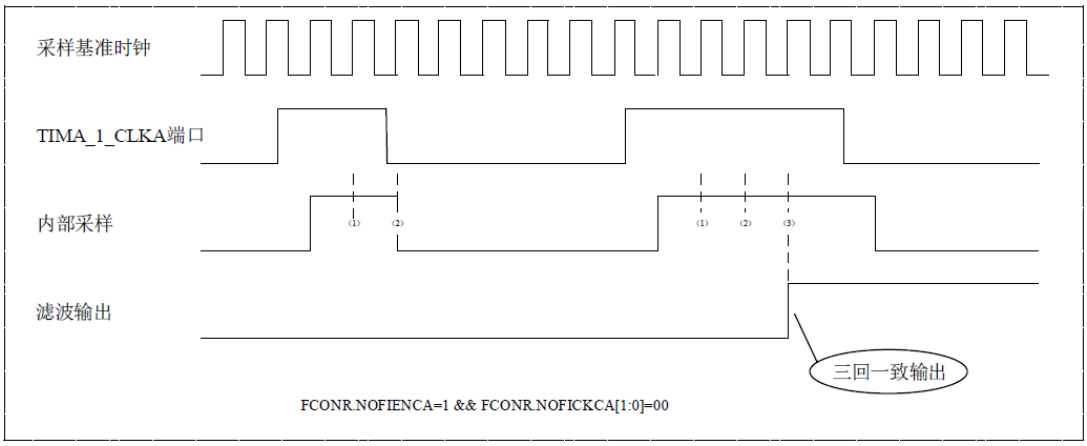


图 3-6 滤波示意图

应用程序应该根据实际情况，设置合适的滤波时钟分频数。在例程 timera_base_timer 中，用到了端口 TIMA_<t>_TRIG 的数字滤波功能。

3.8 比较基准值的缓存

TimerA 的 4 个通道的比较基准寄存器（CMPAR）可以成对（通道 1 与通道 2 一对，通道 3 与通道 4 一对）实现缓存功能。当缓存条件产生后，偶数通道的比较值寄存器的值被缓存至奇数通道的比较值寄存器中，如图 3-7 所示。缓存条件如下：

a) 以 PCLK 作为计数时钟，在三角波计数波形计数模式下，计数到峰点或谷点；

b) 以 PCLK 作为计数时钟，在锯齿波计数波形计数模式下，计数上溢、计数下溢或计数值寄存器发生清零动作；

c) 在硬件计数模式下，计数上溢或计数下溢或计数值寄存器发生清零动作。

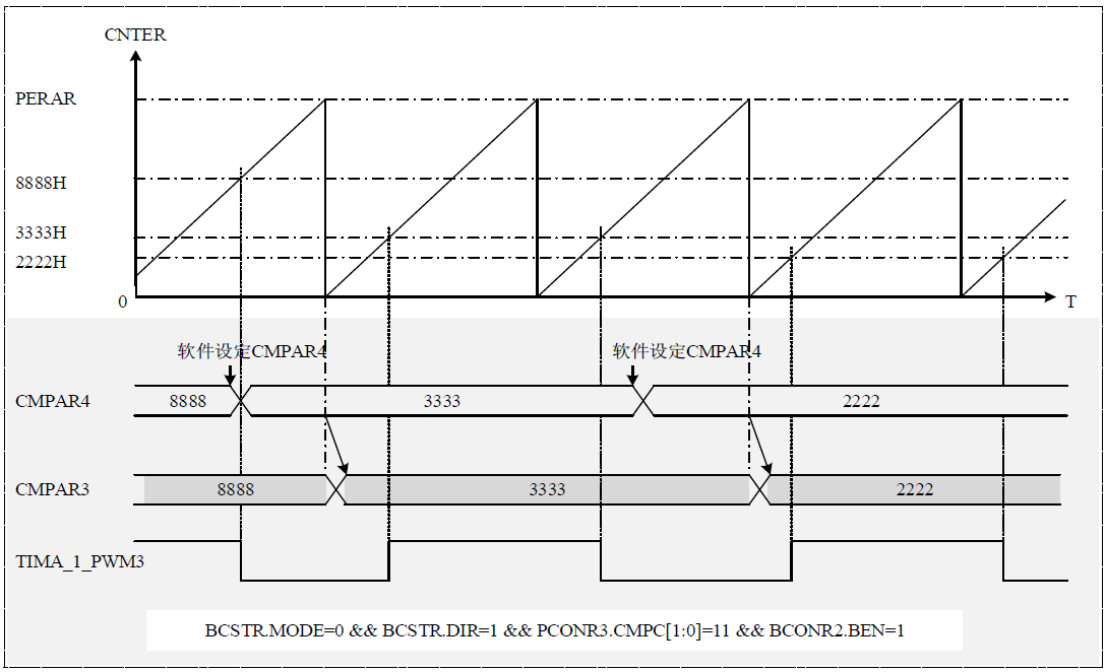


图 3-7 锯齿波模式时缓存时序示例

例程 timera_compare_value_cache 实现了 TimerA 的这个功能。

3.9 级联计数

互为对称单元的两个 TimerA 单元，当偶数单元的计数时钟源选择 d（见 3.3 节）时，可实现级联计数。TimerA 是 16 位计数器，实现级联计数即是实现 32 位计数。在级联计数中，奇数单元的 CNTER 为低 16 位计数器，偶数单元的 CNTER 为高 16 位计数器。

例如，在三角波向上计数模式时，设定单元 1 的计数时钟为 PCLK，设定单元 2 的计数时钟源为单元 1 的计数上溢事件（也可以使用计数下溢事件，或同时使用），启动单元 1、2 计数（先启动单元 2，再启动单元 1）就实现级联计数。单元 1 的 CNTER 位低 16 位计数器，单元 2 的 CNTER 为高 16 位计数器。如图 3-8 所示。

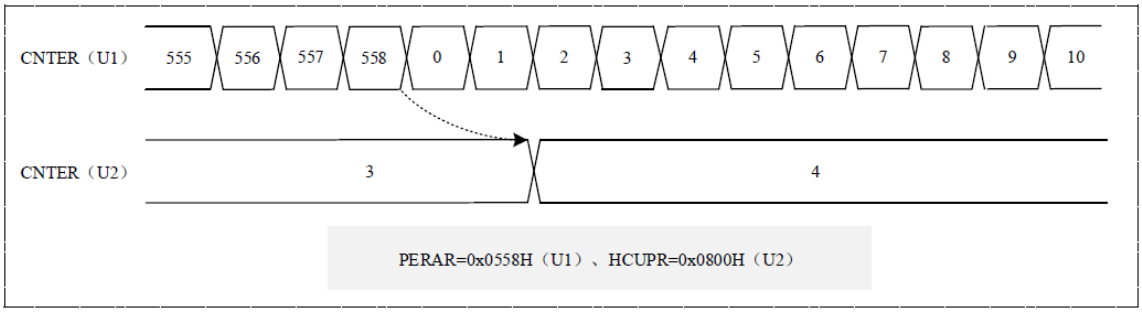


图 3-8 级联计数

在锯齿波计数波形模式下，根据指定的计数方向，来设置偶数单元的计数时钟源。当计数方向向上时，为计数上溢事件，反之则为计数下溢事件。

例程 timera_cascade_count 用 TimerA 的单元 1 和单元 2，用级联计数的方式，实现了 10 毫秒的定时。其中，单元 1 的计数时钟为 PCLK，单元 2 的计数时钟，可根据不同的配置选择对称单元的上溢或下溢事件。

3.10 PWM 输出

HC32F4A0 系列 MCU 搭载 12 个单元的 TimerA，每个单元有 4 个通道，最多可实现 48 路 PWM 输出。通过不同的配置，TimerA 可输出单边对齐 PWM 和双边对称 PWM。

3.10.1 单边对齐 PWM

所谓单边对齐，即是在周期点对齐，从输出的 PWM 波形上看，输出电平在计数值与周期值比较匹配后同时翻转，如图 3-9 所示。TimerA 单元的各个通道共享计数值寄存器和周期值寄存器，可通过如下配置实现单边对齐 PWM 的输出：

1) 设置计数波形模式为锯齿波模式；

2) 选择两个或多个通道，分别设置各自的比较值寄存器；

3) 每个通道设定在比较基准值比较匹配时翻转、在周期基准值比较匹配时翻转。

将例程 timera_pwm 中的宏“APP_FUNC”定义为“APP_FUNC_SINGLE_EDGE_ALIGNED_PWM”，即可输出单边对齐的 PWM。

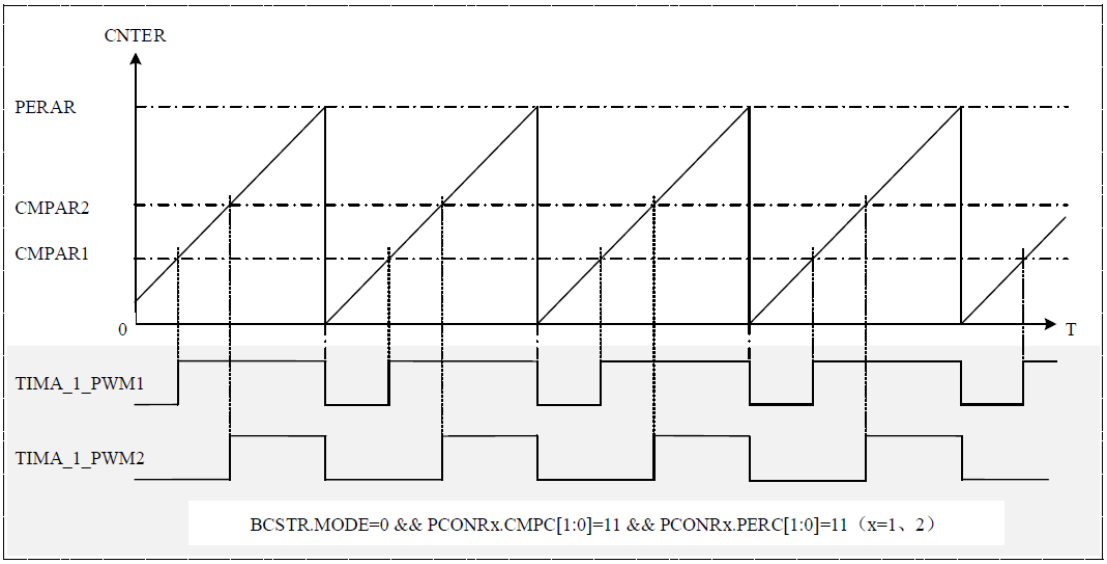


图 3-9 单边对齐 PWM 输出示例

3.10.2 双边对称 PWM

选择三角波计数模式，可以实现双边对称 PWM 输出（以计数峰点对称）。根据通道间的输出关系，可以实现独立的 PWM 输出或互补的 PWM 输出。如图 3-10 所示，是三角波模式

下，通道 1、2、3、4 的双边对称 PWM 输出波形例。其中，通道 1、2 可以作为一对互补 PWM 输出。

将例程 timera_pwm 中的宏“APP_FUNC”定义为“APP_FUNC_TOW_EDGE_SYMMETRIC_PWM”，即可输出双边对称的 PWM，定义为“APP_FUNC_NORMAL_SINGLE_PWM”即输出一路独立的 PWM。

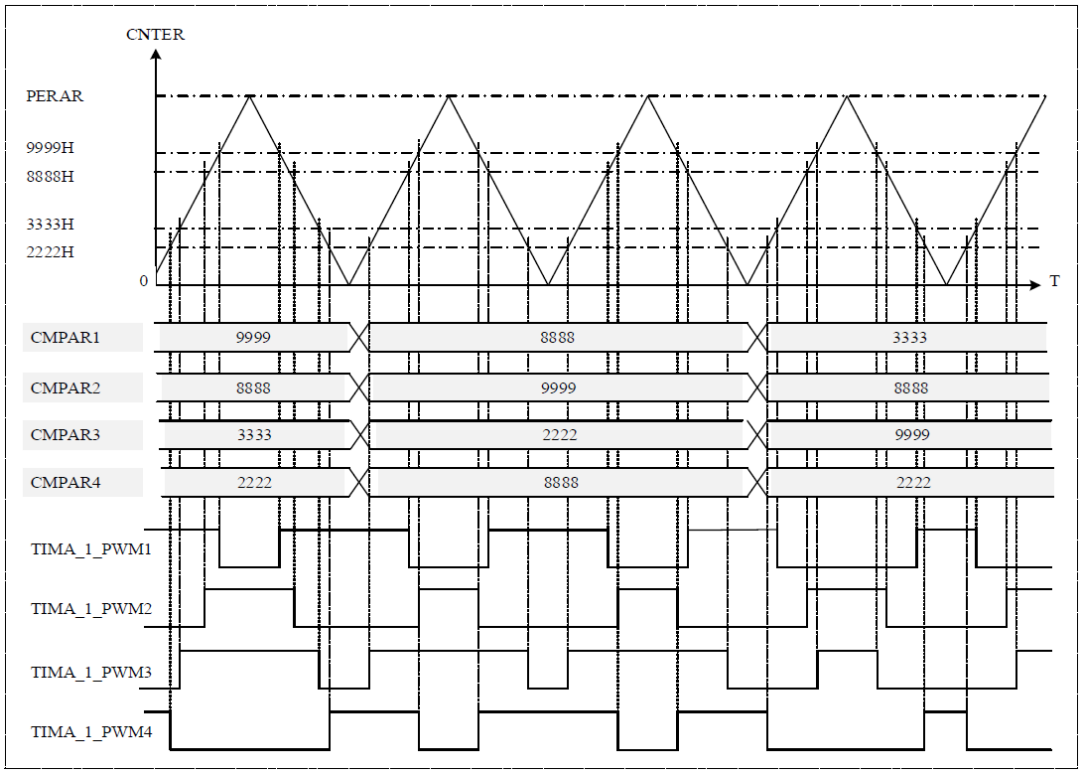


图 3-10 双边对称 PWM 输出示例

3.11 正交编码计数

将 TIMA_<t>_CLKA 输入看作 AIN 输入、TIMA_<t>_CLKB 输入看作 BIN 输入、TIMA_<t>_TRIG 输入看作 ZIN 输入，TimerA 就可以实现三路输入的正交编码计数。

每个单元的 AIN、BIN 单独动作可以实现位置计数模式；两个单元的 AIN、BIN、ZIN 组合动作可以实现公转计数模式，其中用于位置计数的单元称之为位置计数单元、用于公转计数的单元称之为公转计数单元。公转计数模式时，使用对称单元组合，组合内位置计数单元和公转计数单元可以任意指定。

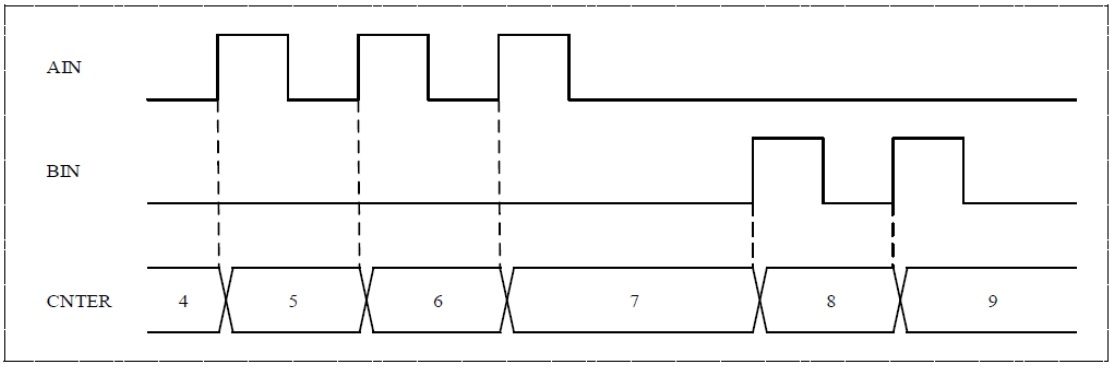
AIN 和 BIN 的计数条件使能通过设定硬件递加事件选择寄存器（HCUPR）和硬件递减事件选择寄存器（HCDOR）中 TIMA_<t>_CLKA 和 TIMA_<t>_CLKB 的正交关系来实现；ZIN 的输入动作通过设定位置计数单元的硬件触发事件选择寄存器（HCONR）的清零使能位实

现位置定时器清零、通过设定公转单元的硬件递加事件选择寄存器（HCUPR）实现公转定时器计数。

3.11.1 位置计数模式

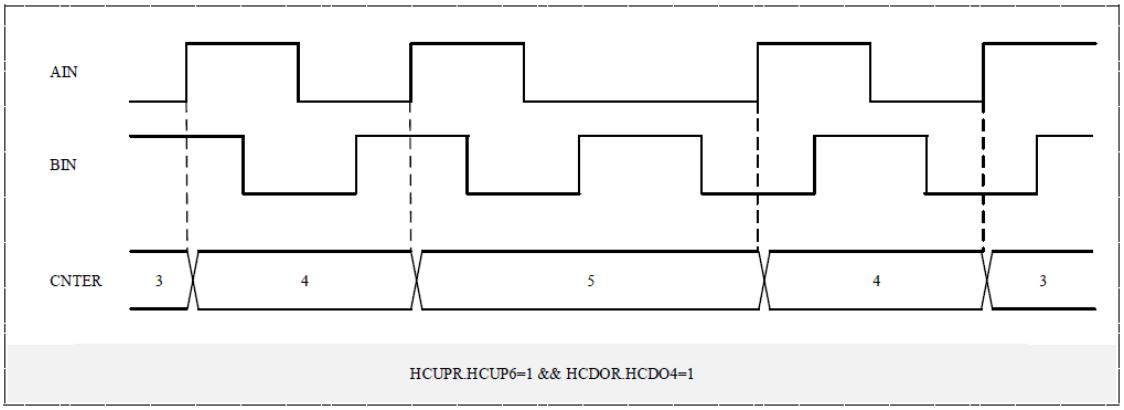
正交编码位置计数模式，是指根据 AIN、BIN 的输入实现基本计数功能、相位差计数功能和方向计数功能。

基本计数。基本计数动作是根据 AIN 或 BIN 端口的输入时钟进行计数，如图 3-11 所示。

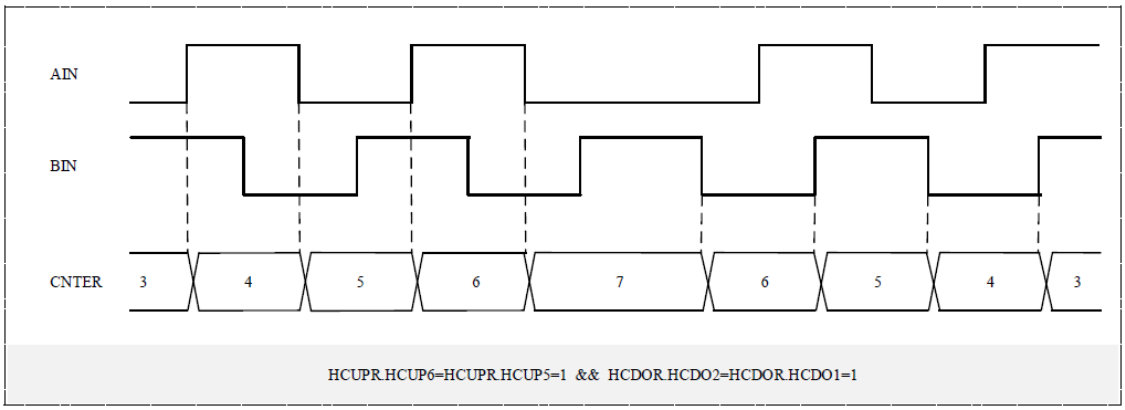


3-11 位置计数模式—基本计数

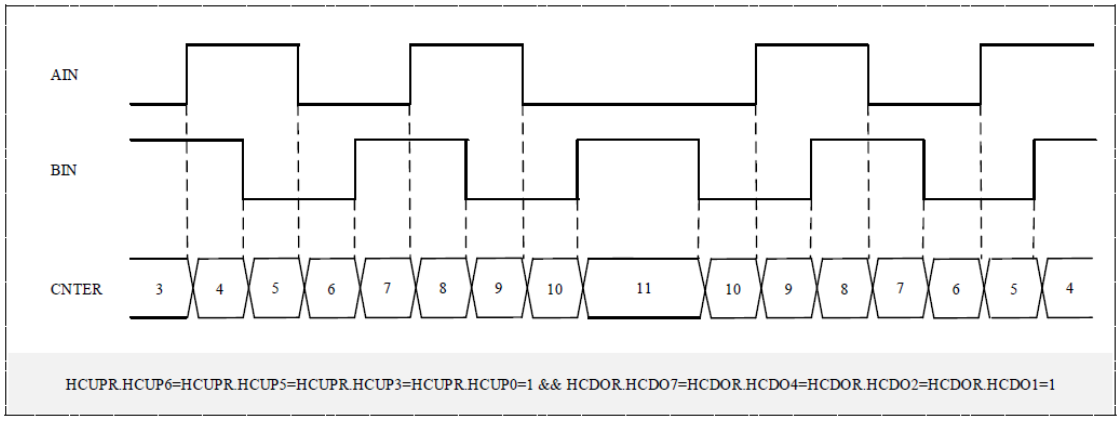
相位差计数。相位差计数是指根据 AIN 和 BIN 的相位关系进行计数。根据设定的不同，可以实现 1 倍计数、2 倍计数、4 倍计数等，如图 3-12、图 3-13 和图 3-14 所示。



3-12 位置计数模式—相位差计数（1 倍计数）

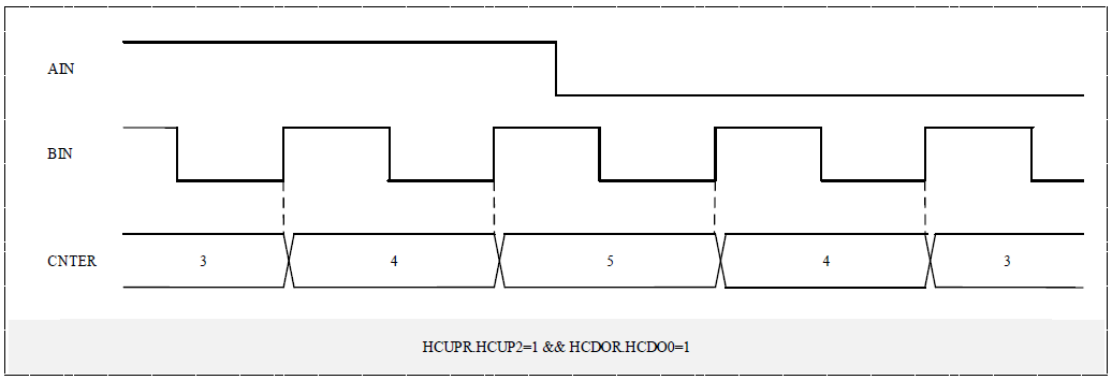


3-13 位置计数模式—相位差计数（2 倍计数）



3-14 位置计数模式—相位差计数（4 倍计数）

方向计数。方向计数是指将 AIN 的输入状态设定为方向控制，将 BIN 的输入作为时钟计数，如图 3-15 所示。当前计数方向可通过读取寄存器 BCSTR.DIR 获得。



3-15 位置计数模式—方向计数

例程 timera_oc_position_count，详细介绍了如何应用位置计数方法测量正交编码器的转速（RPM，转每分钟）。该例程实现的基本原理是，根据相位差计数的倍数，设置正确的硬件计数时钟源，用一分钟内计数器的计数次数，除以正交编码器 PPR（每转输出的脉冲数）与相位差计数倍数的乘积。在应用过程中，须注意设置的单位时间内，计数器是否会溢出。

3.11.2 公转计数模式

正交编码公转计数模式，是指在 AIN、BIN 计数的基础上，加入 ZIN 的输入事件以实现对公转圈数等的判断。公转计数模式时根据公转定时器的计数方式，可实现 Z 相计数功能、位置溢出计数功能和混合计数功能。

Z 相计数。Z 相计数是指根据 ZIN 的输入，公转计数单元进行计数，同时将位置计数单元清零的计数动作，如图 3-16 所示。

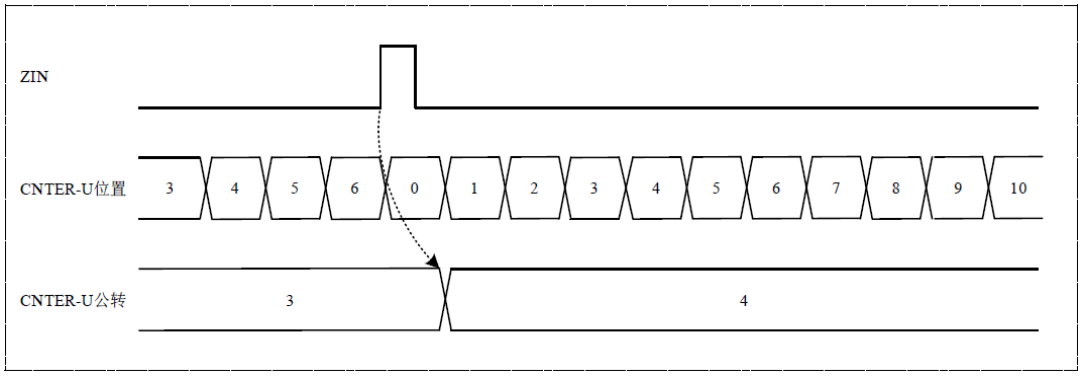


图 3-16 公转计数模式—Z 相计数

位置溢出计数。位置溢出计数是指位置计数单元计数发生上溢或下溢时，产生一个溢出事件，从而触发公转计数单元的定时器进行一次计数（在该计数方式时 ZIN 的输入不进行公转计数单元的计数动作和位置计数单元的清零动作）。

公转计数单元的硬件递加（递减）事件选择寄存器（HCUPR 或 HCDOR）的递加（递减）事件 bit12~11 位使能，位置计数单元的溢出事件就可以触发公转计数单元实现一次计数。如图 3-17 所示。

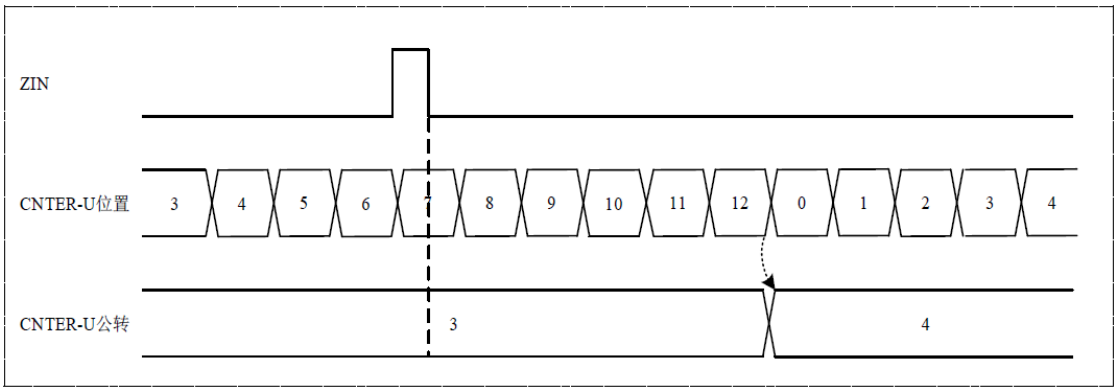


图 3-17 公转计数模式—位置溢出计数

混合计数。混合计数是指上述 Z 相计数和位置溢出计数两种计数方式合并起来的计数动作，其实现方式也是上述两种计数方式的组合，如图 3-18 所示。

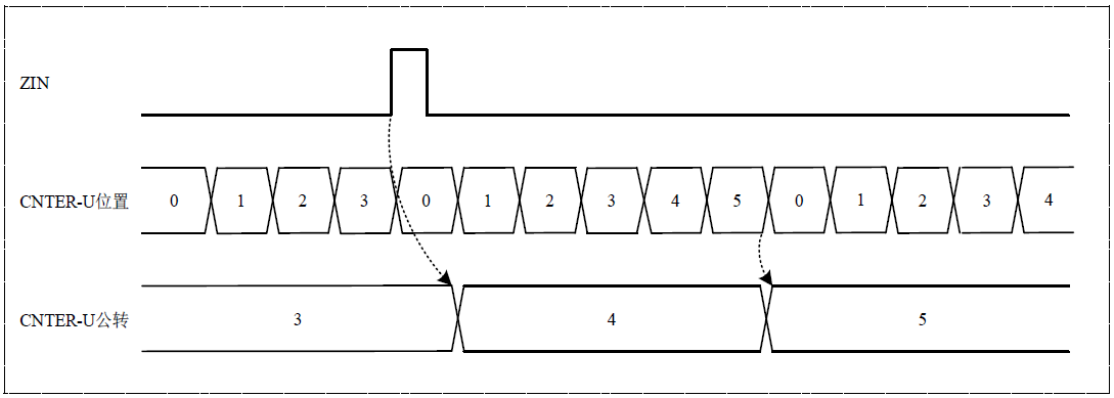


图 3-17 公转计数模式—混合计数

三相正交编码器每转一周，其 Z 相会输出一个脉冲。例程 timera_oc_revolution_count 正是应用这个原理，分别用 Z 相计数和位置溢出计数两种方法，实现了对正交编码器的转速（RPM）的计算。

4 TimerA 例程讲解

4.1 基本流程

要使用 TimerA，一般需要做图 4-1 所示的配置。

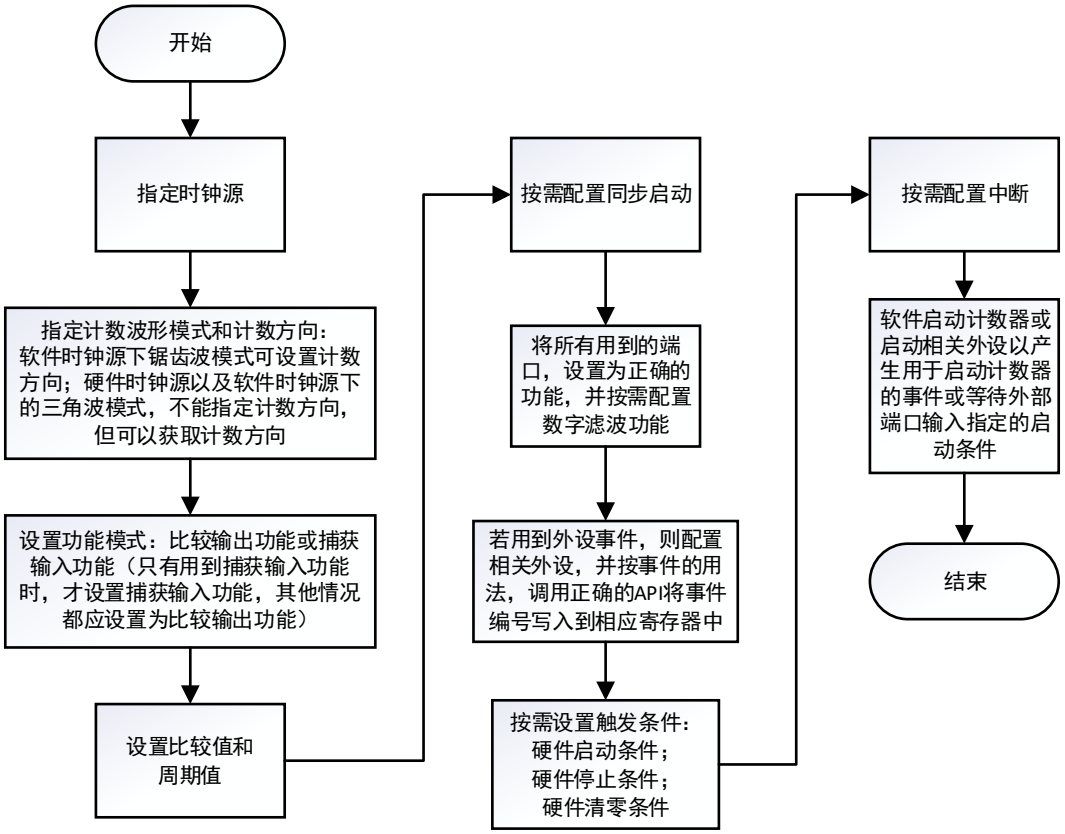


图 4-1 TimerA 应用配置流程

4.2 应用程序源码说明

用户可根据 4.1 节的流程图编写自己的 TimerA 应用程序，也可以通过华大半导体的网站下载 HC32F4A0 系列 MCU 的设备驱动库（Device Driver Library，DDL），参考其中的 TimerA 例程。这里以其中的例程 timera_base_timer，对 TimerA 应用程序的源码做一个简单的说明。

4.2.1 应用程序基本结构

```
int32_t main(void)
{
    /* MCU Peripheral registers write unprotected. */
    Peripheral_WE();
    /* Configures the system clock. */
    SystemClockConfig();
```

```
    /* Configures indicator. */
    IndicateConfig();
    /* Configures TimerA. */
    TmrAConfig();
    /* MCU Peripheral registers write protected. */
    Peripheral_WP();

    /* Starts TimerA. */
    TmrAStart();

    /****************** Configuration end, application start **************/
    for (;;)
    {
      /* Your application code. */
    }
}
```

程序清单 4-1 TimerA 应用程序例程基本结构

说明：

1) Peripheral_WE()：解锁用于保护的寄存器；

2) SystemClockConfig()：配置新的系统时钟（按需配置）；

3) IndicateConfig()：用于指示 TimerA 运行情况（按需配置）；

4) TmrAConfig()：TimerA 的所有配置，包括时钟源、功能模式、硬件触发条件等；

5) Peripheral_WP()：给 Peripheral_WE()解锁的寄存器上锁；

6) TmrAStart()：启动 TimerA。

4.2.2 重要源码讲解

下面详细介绍例程中 TimerA 的配置。

1. TimerA 配置函数 TmrAConfig()。

```
static void TmrAConfig(void)
{
    stc_tmra_init_t stcInit;

    /* 1. Enable TimerA peripheral clock. */
    PWC_Fcg2PeriphClockCmd(APP_TMRA_PERIP_CLK, Enable);

    /* 2. Set a default initialization value for stcInit. */
    (void)TMRA_StructInit(&stcInit);
```

```
    /* 3. Modifies the initialization values depends on the application. */
    stcInit.u32ClkSrc    = APP_TMRA_CLK;
    stcInit.u32PCLKDiv   = APP_TMRA_PCLK_DIV;
    stcInit.u32CntDir    = APP_TMRA_DIR;
    stcInit.u32CntMode   = APP_TMRA_MODE;
    stcInit.u32PeriodVal = APP_TMRA_PERIOD_VAL;
    (void)TMRA_Init(APP_TMRA_UNIT, &stcInit);

    /* 4. Set the comparison reference value. */
    TMRA_SetCmpVal(APP_TMRA_UNIT, APP_TMRA_CH, APP_TMRA_CMP_VAL);

#if (APP_TMRA_USE_INTERRUPT > 0U)
    /* 5. Configures IRQ if needed. */
    TmrAIrqConfig();
#endif /* #if (APP_TMRA_USE_INTERRUPT > 0U) */

#if (APP_TMRA_USE_HW_TRIG > 0U)
    /* 6. Configures hardware trigger condition if needed. */
    TmrATrigCondConfig();
#endif /* #if (APP_TMRA_USE_HW_TRIG > 0U) */
}
```

程序清单 4-2 TimerA 配置

说明：

1) 使能 TimerA 外设时钟；

2) 调用 API 给 TimerA 初始化结构体赋初值；

3) 按需修改 TimerA 初始化结构体的值，如修改计数时钟源、时钟分频、计数方向、计数波形模式、设置周期值等；

4) 设置通道的比较值；

5) 按需配置并使能中断；

6) 按需配置硬件触发条件。

如果只是将 TimerA 用作普通的定时功能，则完成上述前 3 个步骤的配置即可，可使用 TimerA 单元的计数上溢或下溢中断。

2. TimerA 硬件触发条件配置函数 TmrATrigCondConfig()。

```
static void TmrATrigCondConfig(void)
{
    stc_tmra_trig_cond_t stcTrigCond;
```

```
    /*
     * If a peripheral is used to generate the event which is used as a hardware trigger condition of TimerA, \
     *   call the API of the peripheral to configure it.
     * The following operations are only used in this example.
     */

#if (defined APP_TMRA_FILTER_ENABLE &&  APP_TMRA_FILTER_ENABLE > 0U)
    TMRA_FilterConfig(APP_TMRA_UNIT, APP_TMRA_FILTER_PIN, APP_TMRA_FILTER_CLK_DIV
   );
    TMRA_FilterCmd(APP_TMRA_UNIT, APP_TMRA_FILTER_PIN, Enable);
#endif
    GPIO_SetFunc(APP_TMRA_TRIG_PORT, APP_TMRA_TRIG_PIN, APP_TMRA_TRIG_PIN_FUNC,
   PIN_SUBFUNC_DISABLE);

    (void)TMRA_TrigCondStructInit(&stcTrigCond);
    stcTrigCond.u32StartCond = APP_TMRA_START_COND;
    stcTrigCond.u32StopCond  = APP_TMRA_STOP_COND;
    (void)TMRA_SetTrigCond(APP_TMRA_UNIT, &stcTrigCond);
}
```

程序清单 4-3 TimerA 触发条件配置

说明：

1) TMRA_FilterConfig()和 TMRA_FilterCmd()：如果用到了外部端口作为硬件触发条件，则按需配置和使能该端口的数字滤波功能。本例程使用 PC0 的上升沿作为 TimerA 的启动条件，下降沿作为 TimerA 的停止条件；

2) GPIO_SetFunc()：将用到的端口设置为正确的功能；

3) 如果用到片上外设的事件作为硬件触发条件，则需要配置相关外设；

4) TMRA_TrigCondStructInit()：给硬件触发条件配置结构体赋初值；

5) TMRA_SetTrigCond()：配置硬件触发条件。

3. TimerA 启动函数 TmrAStart()。

```
static void TmrAStart(void)
{
    /*
     * If a peripheral is used to generate the event which is used as a hardware trigger condition of TimerA, \
     *   call the API of the peripheral to start the peripheral here or anywhere else you need.
     * The following operations are only used in this example.
     */
```

```
#if ((APP_TMRA_USE_HW_TRIG == 0U) || \
    ((APP_TMRA_USE_HW_TRIG > 0U) && (APP_TMRA_START_COND == TMRA_START_COND_I
  NVALID)))
   TMRA_Start(APP_TMRA_UNIT);
#else
   /* Make an rising edge on pin TIMA_<t>_TRIG to start TimerA. */
#endif
}
```

程序清单 4-4 TimerA 启动函数

说明：

1) 如果硬件启动条件无效，则直接调用 API TMRA_Start()来启动 TimerA；

2) 如果用到外部端口来启动 TimerA，则需要等待指定端口产生指定的启动条件；如果是用到片上外设产生的事件来启动 TimerA，则需要在合适的位置（满足应用需求）启动该外设，以产生用于启动 TimerA 的事件。

5 总结

本篇应用笔记详细介绍了通用定时器 TimerA 的功能原理及基本的应用流程，用户须结合用户手册对其进行正确的配置，合理的应用。

6 版本信息 & 联系方式

日期	版本	修改记录
2021/7/27	Rev1.0	初版发布

如果您在购买与使用过程中有任何意见或建议，请随时与我们联系。

Email：mcu@hdsc.com.cn

网址：http://www.hdsc.com.cn/mcu.htm

通信地址：上海市浦东新区中科路 1867 号 A 座 10 层

邮编：201203