32 位微控制器

HC32M120 系列的通用定时器 TIMER2

适用对象

系列	产品型号
HC32M120	HC32M120J6TB HC32M120F6TB

目 录

1 摘要

本篇应用笔记主要介绍 HC32M120 系列 MCU 的通用定时器 TIMER2（以下简称 TIMER2）的特点，以及各种功能的实现方法，这些功能包括基本计数功能、PWM 输出、脉宽测量和周期测量等。

2 TIMER2 简介

2.1 功能简介

HC32M120 系列 MCU 的 TIMER2 具有如下特性：

- 可以选择同步时钟或异步时钟作为时钟源
- 可以通过外部引脚（TRIGA）或内部事件来启动、停止、清零计数值寄存器
- 可产生计数匹配中断和计数溢出中断

这些特性可实现如下功能：

- 基本的定时和计数功能
- PWM 输出
- 输入捕获
- 脉宽测量和周期测量
- 事件信号输出（可用于触发其他外设动作）

TIMER2 的基本框图如图 2-1。

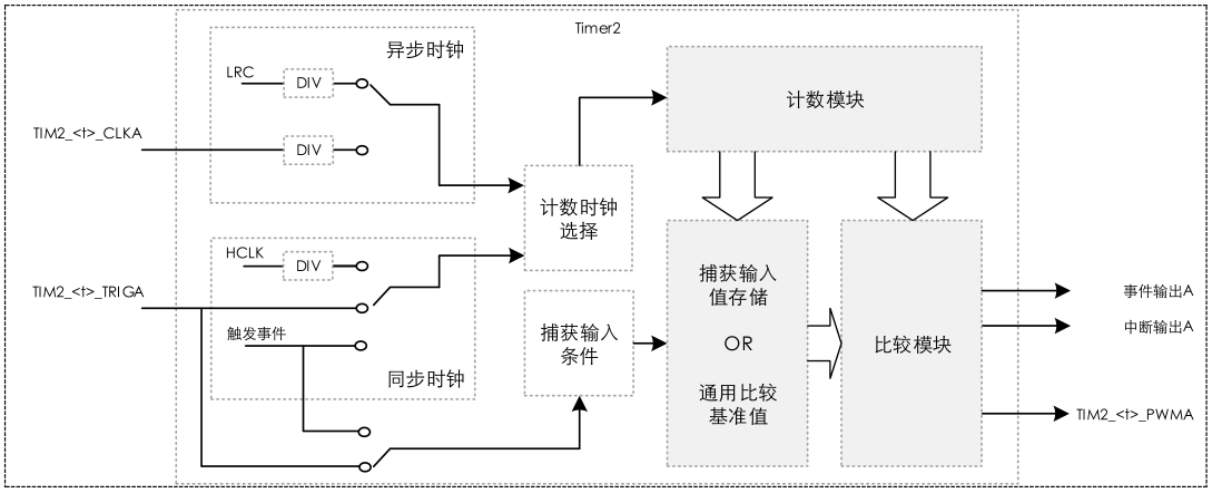

图 2-1 TIMER2 的基本框图如图

2.2 引脚配置

HC32M120 系列 MCU 的 TIMER2 的端口配置如表 2-2 所示。

端口名	方向	功能
TIM2_1_CLKA_A/B/C	输入	异步时钟输入端口
TIM2_1_TRIGA_A/B/C/D/E/F	输入	1. TIMER2 启动、停止、计数值寄存器清零、计数输入端口； 2. 捕获输入端口。
TIM2_1_PWMA_A/B/C/D/E/F	输出	PWM 输出端口

表 2-1 TIMER2 的端口配置

说明：

- 下文将分别简称这些端口为 CLKA、TRIGA 和 PWMA。

3 应用

3.1 时钟源

3.1.1 同步时钟源

同步时钟源的选择有如下三种：

① HCKL 及 HCLK 的 2、4、8、16、32、64、128、256、512、1024 分频作为同步时钟源。

② TRIGA 的有效边沿（上升沿或下降沿）被内部 HCLK 采样后作为同步时钟源。

③ 其他外设产生的事件作为同步时钟源。

3.1.2 异步时钟源

异步时钟源的选择有如下两种：

① LRC（32.768KHz）及 LRC 的 2、4、8、16、32、64、128、256、512、1024 分频作为异步时钟源。

② 从 CLKA 时钟输入端口输入的时钟（不大于 25MHz）及其 2、4、8、16、32、64、128、256、512、1024 分频作为异步时钟源。

TIMER2 的每一个功能都可以选择同步时钟或异步时钟作为时钟源，只是，在选择异步时钟源时，有以下两个注意事项：

① 修改计数值寄存器（CNTAR）、基准值寄存器（CMPAR）、启动位（BCONR.CSTA）、状态位（STFLR.CMFA/OVFA）时，TIMER2 从收到写操作后，需要经过 3 个异步时钟周期才将修改值写入到对应的寄存器中。对这些寄存器进行连续写操作，写操作之间需间隔至少 3 个异步时钟周期。

② 因为第 1 点，所以在使用异步时钟时，推荐先配置好其他寄存器，再配置时钟源及时钟分频，最后启动 TIMER2。

例程 timer2_async_clock 给出了异步时钟源的配置和基本用法。

3.2 数字滤波

TIMER2 的 TRIGA 输入端口具有数字滤波功能，可通过设定端口控制寄存器（PCONR.NOFIENA）的使能位开启滤波功能，滤波用的基准时钟也通过端口控制寄存器（PCONR.NOFICKA[1:0]）来设定。

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而滤掉，如图 3-1 所示。

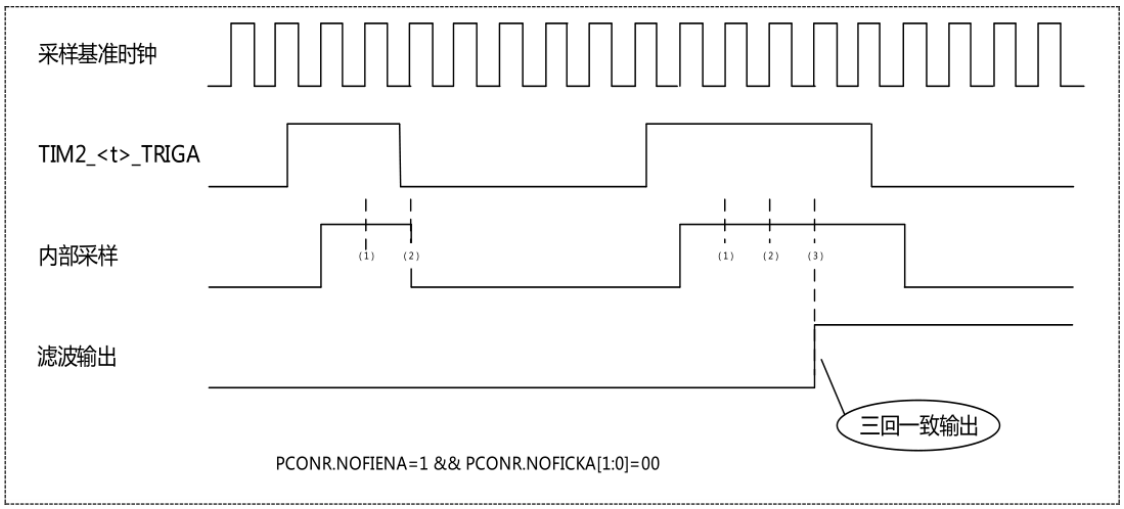

图 3-1 滤波器工作示意图

在用到 TRIGA 端口的地方，都可使用数字滤波功能。

3.3 基本定时功能

3.3.1 功能实现

TIMER2 的定时功能设置比较简单，通过如下几步操作即可实现，在例程 timer2_base_timer 中通过调用函数 TIMER2_TimerConfig 实现：

1. 设置 TIMER2 的功能模式为比较输出功能；
2. 根据定时时间选择合适的时钟源和时钟分频；
3. 根据定时时间、时钟源和时钟分频计算出基准值，并写入基准值寄存器；
4. 配置 TIMER2 的计数匹配中断（非必须）。

基准值可由如下公式计算得到：

CompareVal = Time * (ClockSource / ClockPrescaler)。

其中，CompareVal 是基准值；Time 是定时时间，单位微秒；ClockSource 是 TIMER2 的时钟源，单位 MHz；ClockPrescaler 是时钟源分频数。

3.3.2 例程讲解

例程 timer2_base_timer 用 TIMER2 实现了 1 毫秒定时，并在 TIMER2 计数匹配中断服务函数中翻转一个 IO，可用示波器看到该 IO 输出了一个频率为 500Hz 的方波。配置 TIMER2 的关键代码如程序清单 3-1 所示。

```
static void Timer2Config(void)
{
    stc_timer2_config_t stcCfg;

    /* 1. Enable TIMER2 peripheral clock. */
    CLK_FcgPeriphClockCmd(CLK_FCG_TIM2, Enable);

    /* 2. Set a default configuration value for stcCfg.
        Default configuration values for basic timer is:
          Clock sourc(u16ClkSource): TIMER2_SYNC_CS_HCLK, default clock is HCLK.
          Clock prescaler(u16ClkPrescaler): TIMER2_CLK_PRESCALER_1, prescaler of clock source is 1.
          Start condition(u16HwStartCondition): TIMER2_HW_START_INVALID, do not use the start condition.
          Stop condition(u16HwStopCondition): TIMER2_HW_STOP_INVALID, do not use the stop condition.
          Clear condition(u16HwClearCondition): TIMER2_HW_CLR_INVALID, do not use the clear condition.
          TRIGA filter clock prescaler(u16FilterPrescaler): TIMER2_FILTER_PRESCALER_1, valid while enFilterCmd == Enable.
          TRIGA filter function(enFilterCmd): Disable.
          Counter match interrupt(enMatchIntCmd): Disable.
          Counter overflow interrupt(enOvfIntCmd): Disable.
        All of the above configuration values can be modified based on the application. */
    TIMER2_StructInit(&stcCfg);

    /* 3. Modify the configuration values depends on the application. */
    stcCfg.u16ClkPrescaler = TIMER2_CLK_PRESCALER;
    stcCfg.u16CompareVal   = TIMER2_COMPARE_VALUE;
#if TIMER2_USE_INTERRUPT
    /* Enable counter match interrupt. */
    stcCfg.enMatchIntCmd   = Enable;
#endif

    /* 4. Configures TIMER2 according the configuration value. */
```

```
    TIMER2_TimerConfig(&stcCfg);

#if TIMER2_USE_INTERRUPT
    /* 5. Configures IRQ if needed. */
    Timer2IrqConfig();
#endif
}
```

程序清单 3-1 TIMER2 实现 1 毫秒定时

源码讲解：

1. 使能 TIMER2 外设时钟；
2. 通过调用函数 TIMER2_StructInit 为 TIMER2 配置结构体变量设置默认值，注释部分是对基本定时功能用到的参数的默认值的解释；
3. 根据具体需求修改配置值，如果需要使用中断，在此处使能中断；
4. 调用函数 TIMER2_TimerConfig 将配置值写入到相应寄存器，完成基本定时功能的配置；
5. 如果用到了中断，配置中断向量。

将例程重新编译并下载至目标板，将示波器探头连接至 P30，可看到示波器显示频率为 500Hz 的方波。

3.4 基本计数功能

3.4.1 功能实现

设置外设事件或被内部 HCLK 采样后的 TRIGA 的有效边沿（上升沿或下降沿）作为 TIMER2 的同步时钟源，可实现对外设事件进行计数或 TRIGA 的有效边沿计数，具体配置如下：

1. 设置 TIMER2 的功能模式为比较输出功能；
2. 设置外设事件或 TRIGA 的有效边沿（上升沿或下降沿）作为 TIMER2 的同步时钟源；
3. 设置基准值寄存器；
4. 配置 TIM2_1_TRIGA 引脚或需要进行计数的外设事件；
5. 配置中断（TIMER2 的计数匹配中断）。

说明：

- 以上第 3 步和第 5 步，如果需要在计数指定次数后产生中断才需要；
- 在选择外设事件或被内部 HCLK 采样后的 TRIGA 的有效边沿（上升沿或下降沿）作为 TIMER2 的同步时钟源时，时钟分频无效。

以上操作在例程 timer2_base_counter 中通过调用函数 TIMER2_TimerConfig 实现。

3.4.2 例程讲解

例程 timer2_base_counter 实现了对外部中断 0 事件（EVT_PORT_EIRQ0）或 TRIGA_A（P30）的下降沿进行计数，并设置计数 5 次后产生中断。在实际应用中，可通过读取计数值寄存器来获取计数次数，也可设置在计数指定次数后产生中断。功能实现的关键代码程序清单 3-2 所示。

```
static void Timer2Config(void)
{
    stc_timer2_config_t stcCfg;

    /* 1. Enable TIMER2 peripheral clock. */
    CLK_FcgPeriphClockCmd(CLK_FCG_TIM2, Enable);

    /* 2. Set a default configuration value for stcCfg.
        Default configuration values for basic counter is:
          Clock sourc(u16ClkSource): TIMER2_SYNC_CS_HCLK, default clock is HCLK.
          Clock prescaler(u16ClkPrescaler): TIMER2_CLK_PRESCALER_1, prescaler of clock source is 1.
          Start condition(u16HwStartCondition): TIMER2_HW_START_INVALID, do not use the start condition.
          Stop condition(u16HwStopCondition): TIMER2_HW_STOP_INVALID, do not use the stop condition.
          Clear condition(u16HwClearCondition): TIMER2_HW_CLR_INVALID, do not use the clear condition.
          TRIGA filter clock prescaler(u16FilterPrescaler): TIMER2_FILTER_PRESCALER_1, valid while enFilterCmd == Enable.
          TRIGA filter function(enFilterCmd): Disable.
          Counter match interrupt(enMatchIntCmd): Disable.
          Counter overflow interrupt(enOvfIntCmd): Disable.
        All of the above configuration values can be modified based on the application. */
    TIMER2_StructInit(&stcCfg);

    /* 3. Modify the configuration values depends on the application. */
    stcCfg.u16ClkSource    = TIMER2_SYNC_CLK_SRC;
    stcCfg.u16CompareVal   = TIMER2_COUNT_TIMES - 1u;
#if TIMER2_USE_INTERRUPT
    /* Enable counter match interrupt. */
    stcCfg.enMatchIntCmd   = Enable;
    /* Counter overflow interrupt can also be enabled if needed. */
    stcCfg.enOvfIntCmd     = Enable;
```

```
#endif

    /* 4. Configures TIMER2 according the configuration value. */
    TIMER2_TimerConfig(&stcCfg);

#if (TIMER2_SYNC_CLK == SYNC_CLK_EVENT)
    /* 5. Configures the event for TIMER2. */
    /* Enable AOS function. */
    CLK_FcgPeriphClockCmd(CLK_FCG_AOS, Enable);
    /* Use event TIMER2_SYNC_CLK_EVENT(depends on the application) as synchronous clock.  */
    TIMER2_SetTrigEvent(TIMER2_SYNC_CLK_EVENT);
#else
    /* 5. Set the specified pin as TRIGA input pin. */
    GPIO_SetFunc(TIMER2_TRIGA_PORT, TIMER2_TRIGA_PIN, GPIO_FUNC_3_TIM2);
#endif // #if (TIMER2_SYNC_CLK == SYNC_CLK_EVENT)

#if TIMER2_USE_INTERRUPT
    /* 6. Configures IRQ if needed. */
    Timer2IrqConfig();
#endif
}
```

程序清单 3-2 TIMER2 实现计数功能

源码讲解：

1. 使能 TIMER2 外设时钟；
2. 通过调用函数 TIMER2_StructInit 为 TIMER2 配置结构体变量设置默认值，注释部分是对基本计数功能用到的参数的默认值的解释；
3. 根据具体需求修改配置值，如果需要使用中断，在此处使能中断；
4. 调用函数 TIMER2_TimerConfig 将配置值写入到相应寄存器，完成基本计数功能的配置；
5. 如果选择的是对事件计数，则配置该事件为外部中断 0 事件（EVT_PORT_EIRQ0）；如果选择的是对 TRIGA 的上升沿计数，则配置 TRIGA_A（P30）的端口功能；
6. 根据需要来配置中断向量。

   设置好例程的功能，如设置为对外部中断 0 事件计数（在开发板上，按下 SW1 可产生该事件），调试程序，将断点设置在中断服务函数中，连续按下 SW1 按键 5 次后，程序将会停止在中断服务函数中的断点处。

3.5 PWM 输出

3.5.1 功能介绍

TIMER2 的 PWM 实现原理是，计数值寄存器累加到与基准值寄存器相等时，翻转指定的端口的电平，同时计数值寄存器清零，如图 3-2 所示。

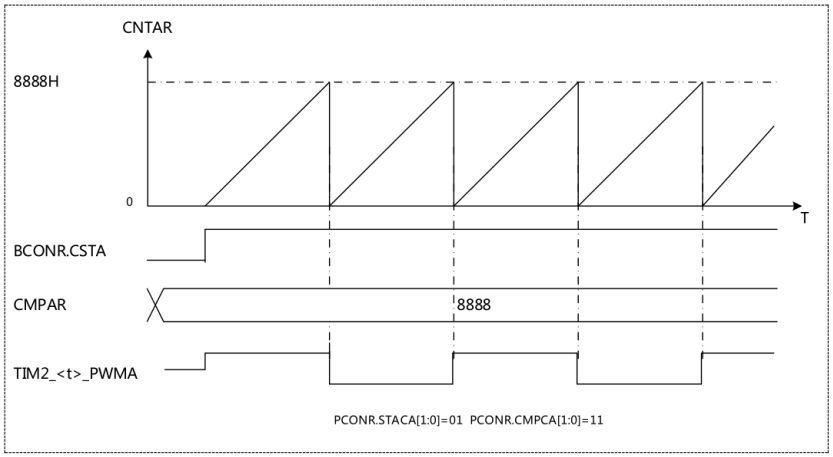

图 3-2 TIMER2 的 PWM 实现原理示意图

TIMER2 用于 PWM 输出的端口共有 6 个，分别是 PWMA_A（P30）、PWMA_B（P31）、PWMA_C（P01）、PWMA_D（P50）、PWMA_E（P00）和 PWMA_F（P41）。

注意：

- PWMA 端口与 TRIGA 端口共用，但不可同时使用。如果使能了 PWM 功能，并设置了一个端口作为 PWM 输出端口，那么其他端口的 TRIGA 其他相关功能都不可再使用。
- TIMER2 的 PWM 占空比不可调，恒为 50%。
- PWM 频率对应的基准值的计算公式如下：
- CompareVal = (ClockSource / ClockPrescaler) / PwmFrequency / 2
- 其中，CompareVal 是基准值；ClockSource 是时钟源频率（Hz）；ClockPrescaler 是时钟源分频数；PwmFrequency 是 PWM 频率（Hz）。

3.5.2 例程讲解

例程 timer2_pwm 实现了用 TIMER2 输出一路频率为 2KHz 的 PWM，关键配置代码如程序清单 3-4 所示。

```
static void Timer2Config(void)
{
    stc_timer2_config_t stcCfg;

    /* 1. Enable TIMER2 peripheral clock. */
    CLK_FcgPeriphClockCmd(CLK_FCG_TIM2, Enable);

    /* 2. Set a default configuration value for stcCfg.
        Default configuration values for basic timer is:
          Clock sourc(u16ClkSource): TIMER2_SYNC_CS_HCLK, default clock is HCLK.
          Clock prescaler(u16ClkPrescaler): TIMER2_CLK_PRESCALER_1, prescaler of clock source is 1.
          Start condition(u16HwStartCondition): TIMER2_HW_START_INVALID, do not use the start condition.
          Stop condition(u16HwStopCondition): TIMER2_HW_STOP_INVALID, do not use the stop condition.
          Start polarity(u16PwmStartPolarity): TIMER2_PWM_START_LOW, PWM start polarity is low.
          Stop polarity(u16PwmStopPolarity): TIMER2_PWM_START_LOW, PWM stop polarity is low.
        All of the above configuration values can be modified based on the application.
        NOTE: Start condition and stop condition CAN NOT be the edge of TRIGA. */
    TIMER2_StructInit(&stcCfg);

    /* 3. Modify the configuration value depends on the application. */
    stcCfg.u16ClkPrescaler = TIMER2_CLK_PRESCALER;
    stcCfg.u16CompareVal   = TIMER2_COMPARE_VALUE;

    /* 4. Configures TIMER2 according the configuration value. */
    TIMER2_PwmConfig(&stcCfg);

    /* 5. Set the GPIO pin as PWM output pin. */
    GPIO_SetFunc(TIMER2_PWM_PORT, TIMER2_PWM_PIN, GPIO_FUNC_3_TIM2);
}
```

程序清单 3-4 TIMER2 实现 PWM 输出

源码讲解：

1. 使能 TIMER2 外设时钟；
2. 通过调用函数 TIMER2_StructInit 为 TIMER2 配置结构体变量设置默认值，注释部分是对 PWM 功能用到的参数的默认值的解释；
3. 根据具体需求修改配置值；
4. 调用函数 TIMER2_TimerConfig 将配置值写入到相应寄存器，完成 PWM 功能的配置；
5. 设置 TIMER2 对应的 PWM 引脚的功能。

将例程重新编译并下载至目标板，将示波器探头分别连接至 P30，可看到示波器显示一路频率为 2KHz 的方波。

3.6 硬件控制

TIMER2 的硬件触发源有两类，一个是 TRIGA 的有效边沿（上升沿或下降沿），另一个是外设产生的事件。这两类触发源均可实现计数器的启动、停止、清零以及硬件捕获输入功能。各种功能组合设定也可实现脉宽测量和周期测量等功能，这将在 3.7 节详细描述。

TIMER2 的事件触发源可选择芯片的片上外设可产生的任何一个事件（具体请参考用户手册中断控制器一章），包括 TIMER2 自身的事件（但只能用于停止或清零）。

TIMER2 的硬件控制的配置和用法，例程 timer2_hardware_control 有详细展示，该例程实现了用外部中断 0 事件 EVT_PORT_EIRQ0 来启动 TIMER2，用 TRIGA_A（P30）的下降沿来停止 TIMER2。将例程重新编译并下载到开发板，按下开发板上的 SW1 按键（以产生外部中断 0 事件），会启动 TIMER2，开发板上的红色 LED 闪烁；在 P30 上制造一个下降沿，红色 LED 将会停止闪烁。

3.7 输入捕获

3.7.1 捕获有效边沿或事件

以 TRIGA 端口输入的有效边沿（上升沿或下降沿）或内部事件作为捕获的条件时，可实现输入捕获功能。当硬件控制寄存器（HCONR）选择的捕获条件有效时，则发生一次捕获动作，当前的计数值就被保存到基准值寄存器（CMPAR）中。图 3-3 为捕获输入的示意图。

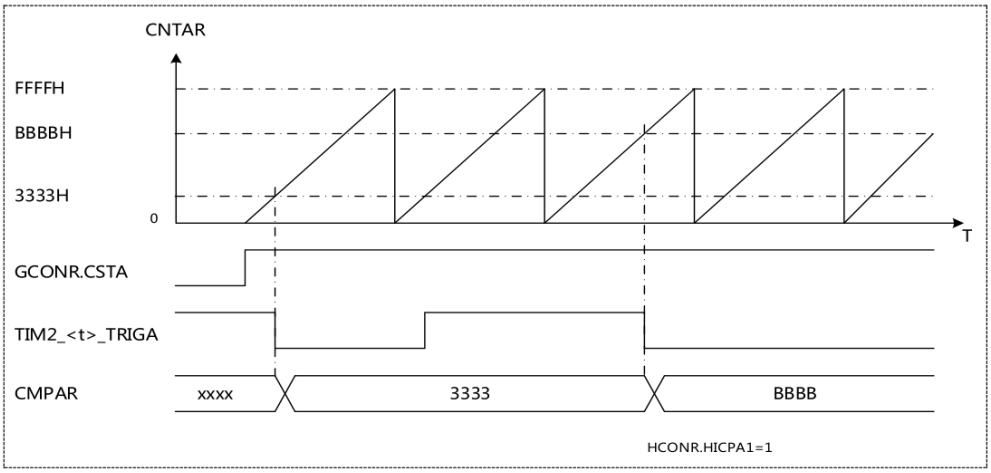

图 3-3 捕获输入示意图

使能计数匹配中断，当捕获动作发生时，则产生一次中断，由此可对 TRIGA 的有效边沿进行计数；使能计数溢出中断，可计算出两次捕获之间的时间间隔。详细实现过程，将在 3.7.4 例程讲解中说明。

3.7.2 脉宽测量

将 TIMER2 的硬件启动条件设为 TRIGA 的上升沿，硬件清零条件、停止条件和捕获输入条件均设为 TRIGA 的下降沿，即可实现连续的脉冲宽度测量，如图 3-4 所示。

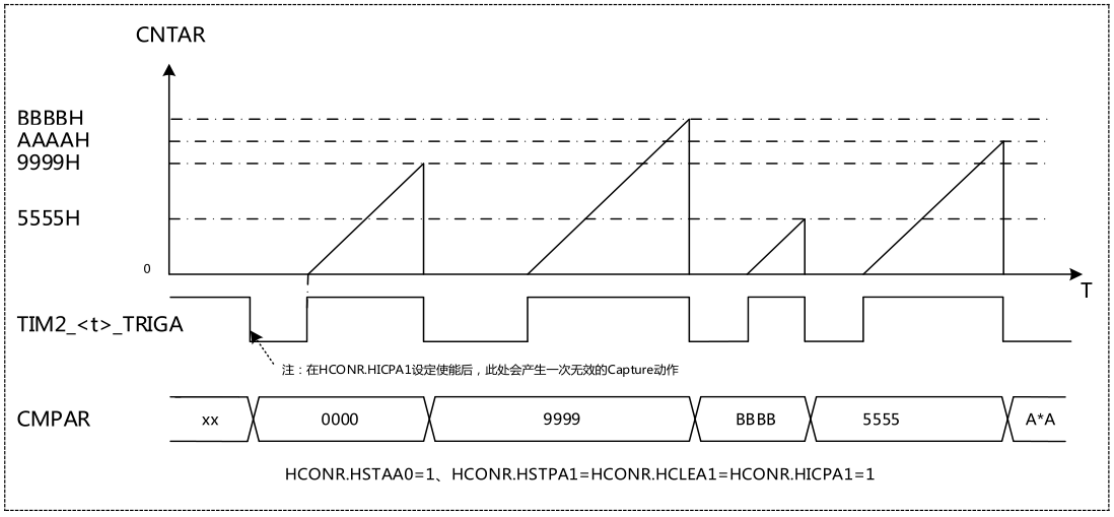

图 3-4 脉宽测量示意图

在实际应用中，需根据输入脉冲的最大宽度来设置 TIMRE2 的时钟源和时钟分频数，以避免计数值溢出而使测量值错误。

3.7.3 周期测量

将 TIMER2 的硬件启动条件、硬件清零条件和捕获输入条件设置为 TRIGA 的相同边沿（上升沿或下降沿），就可用实现连续的周期测量，如图 3-5 所示。

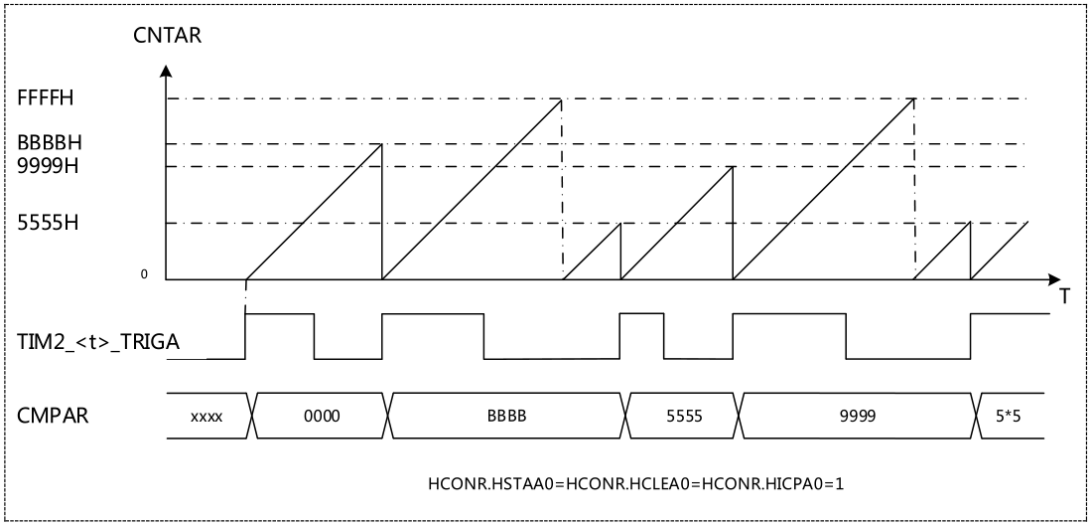

图 3-5 周期测量示意图

在实际应用中，需根据输入波形的最大周期来设置 TIMRE2 的时钟源和时钟分频数，以避免计数值溢出而使测量值错误。

3.7.4 例程讲解

例程 timer2_capture 实现了对 TRIGA_A（P30）下降沿的捕获、对外部中断 2 事件（EVT_PORT_EIRQ2）的捕获、对输入自 TRIGA_A（P30）的方波的脉宽测量和周期测量，主要配置代码如程序清单 3-5 所示。

```
static void Timer2Config(void)
{
    stc_timer2_config_t stcCfg;

    /* 1. Enable TIMER2 peripheral clock. */
    CLK_FcgPeriphClockCmd(CLK_FCG_TIM2, Enable);

    /* 2. Set a default configuration value for stcCfg. */
    TIMER2_StructInit(&stcCfg);

    /* 3. Modify the configuration value depends on the application. */
    stcCfg.u16ClkPrescaler       = TIMER2_CLK_PRESCALER;
    stcCfg.u16HwStartCondition   = TIMER2_START_CONDITION;
    stcCfg.u16HwStopCondition    = TIMER2_STOP_CONDITION;
    stcCfg.u16HwClearCondition   = TIMER2_CLR_CONDITION;
    stcCfg.u16HwCaptureCondition = TIMER2_CAP_CONDITION;
#if TIMER2_TRIGA_FILTER_ENABLE
    /* The clock prescaler of TRIGA filter depends on the application. */
    stcCfg.u16FilterPrescaler    = TIMER2_FILTER_PRESCALER_4;
    stcCfg.enFilterCmd           = Enable;
#endif // #if TIMER2_TRIGA_FILTER_ENABLE
    /* Enable capturing interrupt. */
    stcCfg.enMatchIntCmd = Enable;
#if ((APP_FUNCTION == APP_CAPTURE_EVENT) || (APP_FUNCTION ==
APP_CAPTURE_TRIGA))
    /* Enable counter overflow interrupt if needed.
       If the time interval between two events or two falling edges is greater than 65536 TIMER2 clock
cycles,
       enable counter overflow interrupt.*/
    stcCfg.enOvfIntCmd   = Enable;
#endif

    /* 4. Configures TIMER2 according the configuration value. */
```

```
    TIMER2_CaptureConfig(&stcCfg);

#if (APP_FUNCTION == APP_CAPTURE_EVENT)
    /* 5. Set an event as the object for TIMER2 capturing. */
    /* Enable AOS function. */
    CLK_FcgPeriphClockCmd(CLK_FCG_AOS, Enable);
    /* Set the event. */
    TIMER2_SetTrigEvent(TIMER2_EVENT_FOR_CAPTURE);
#else
    /* 5. Set the specified pin as TRIGA input pin. */
    GPIO_SetFunc(TIMER2_TRIGA_PORT, TIMER2_TRIGA_PIN, GPIO_FUNC_3_TIM2);
#endif // #if (APP_FUNCTION == APP_CAPTURE_EVENT)

    /* 6. Configures IRQ for this example. */
    Timer2IrqConfig();
}
```

程序清单 3-5 TIMER2 实现输入捕获

源码讲解：

1. 使能 TIMER2 外设时钟；
2. 通过调用函数 TIMER2_StructInit 为 TIMER2 配置结构体变量设置默认值，注释部分是对捕获功能用到的参数的默认值的解释；
3. 根据具体需求修改配置值；
4. 调用函数 TIMER2_TimerConfig 将配置值写入到相应寄存器，完成捕获功能的配置；
5. 如果配置捕获外部中断 2 事件，那么需要对外部中断 2 事件进行相关配置，并将 TIMER2 的触发事件设置为外部中断 2 事件 EVT_PORT_EIRQ2；其他情况，则只需配置 TRIGA_A（P30）的引脚功能为功能 GPIO_FUNC_3_TIM2 即可；
6. 本例中用到了 TIMER2 中断，故在此配置中断向量。

在本例中：

1. 设置 HCLK 为同步时钟源（本例程中 HCLK 为 XTAL 20MHz），并设置分频数为 8 分频；
2. 配置硬件控制功能：

1) 配置为捕获外部中断 2 事件时，硬件启动设置为事件启动，硬件停止设置为无效，硬件清零和捕获条件均设置为外部中断 2 事件；

2) 配置为捕获 TRIGA_A（P30）的下降沿时，停止都设置为无效，硬件启动、硬件清零和捕获条件均设置为 TRIGA_A（P30）的下降沿；

3) 配置为脉宽测量时，硬件启动条件设置为 TRIGA_A（P30）的上升沿，硬件清零条件、停止条件和捕获输入条件均设为 TRIGA_A（P30）的下降沿；

4) 配置为周期测量时，硬件启动、硬件清零和捕获条件均设置为 TRIGA_A（P30）的下降沿，硬件停止条件设置为无效。

在实际应用中，可将连续捕获的值（单位是 TIMER2 计数时钟周期）保存在数组中，当达到设定的捕获次数时，再对数组进行处理，如程序清单 3-6 所示。

```
void Timer2GCmp_IrqHandler(void)
{
  if (TIMER2_GetFlag(TIMER2_FLAG_CNT_MATCH) == Set)
  {
    /* Get the capture value. */
    m_au16CaptureVal[m_u8CaptureCount % CAPTURE_COUNT] = TIMER2_GetCompareVal();
    m_u8CaptureCount++;
    TIMER2_ClrFlag(TIMER2_FLAG_CNT_MATCH);
  }
}
```

程序清单 3-6 在计数匹配中断服务函数中保存捕获值

注意：

- 在连续测量脉宽的应用中，计数匹配中断第一次产生时保存的捕获值是无效的。

在捕获 TRIGA 的边沿或其他事件时，如果两次边沿或事件之间的间隔时间较长，已经超过 65536 个 TIMER2 计数时钟周期，则需要借助 TIMER2 的计数溢出中断来计数溢出次数，结合在计数匹配中断服务函数中保存的捕获值，最终算出这个间隔时间。在计数溢出中断服务函数中计数溢出次数，如程序清单 3-7 所示。当使用程序清单 3-6 的配置时，间隔时间可由如下公式算得，其中常数和变量的单位都是 TIMER2 的计数时钟周期：

$$interval = 65536 * ovfCount + capture2$$

其中，interval 是间隔时间；ovfCount 是溢出次数；capture2 是捕获结束时的值。

```
void Timer2GOV_IrqHandler(void)
{
    if (TIMER2_GetFlag(TIMER2_FLAG_CNT_OVF) == Set)
    {
        if (m_u8CaptureCount != 0u)
        {
            m_u32OvfCount++;
        }
        TIMER2_ClrFlag(TIMER2_FLAG_CNT_OVF);
    }
}
```

程序清单 3-6 在计数溢出中断服务函数中计数溢出次数

测试本例程时，先设置好程序的功能，重新编译，设置好断点，在调试状态下全速运行程序，然后制造相应的捕获条件。

4 总结

本应用笔记简要介绍了 HC32M120 系列 TIMER2 模块的各种功能以及实现方法，在实际开发中，用户可根据具体应用场景按需配置和应用 TIMER2 模块。

5 版本信息 & 联系方式

日期	版本	修改记录
2019/12/10	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：www.hdsc.com.cn

通信地址：上海市浦东新区中科路 1867 号 A 座 10 层

邮编：201203