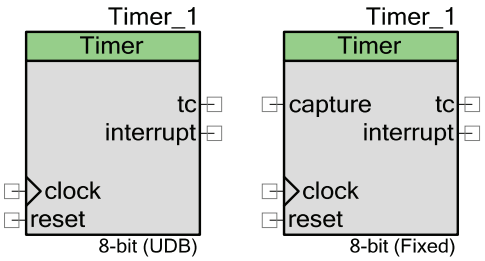
定时器
2.20

特性



- 固定功能 (FF) 和通用数字模块 (UDB) 实现
- 8、16、24 或 32 位定时器
- 可选捕获输入
- 使能、触发和复位输入，用于与其他组件同步
- 连续或单触发运行模式

概述

定时器组件提供了用于测量间隔时间的方法。它可以实现基本定时器功能并实现高级功能（如使用捕获计数器进行捕获以及中断/ DMA 生成）。

此组件可以使用 FF 模块或 UDB 实现。用 UDB 实现所具有的功能通常多于用 FF 实现。如果设计足够简单，请考虑使用 FF，从而省下 UDB 资源以用于其他用途。

下表显示了 FF 与 UDB 之间的主要功能差异。用 FF 与 UDB 实现之间还存在许多特定功能差异，并且在不同器件中用 FF 实现之间也存在差异。有关各种实现的详细时序波形，请参见配置一节。

特性	FF	UDB
位数	8 或 16	8、16、24 或 32
运行模式	连续或单触发	连续、单触发或单触发中断时停止
计数模式	仅向下	仅向下
启用输入	是（硬件或软件使能）	是（硬件或软件使能）
捕获输入	是	是
捕获模式	仅上升沿	上升沿、下降沿、任意沿或软件控制
捕获 FIFO	否（一个捕获寄存器）	是（最多四个捕获）
触发器输入	否	是
触发模式	无	上升沿、下降沿、任意沿或软件控制

特性	FF	UDB
复位输入	是	是
终端计数输出	是	是
中断输出	是（仅 PSoC 3）	是
中断条件	TC、捕获	TC、捕获和 FIFO 已满
捕获输出	否	是
周期寄存器	是	是
周期重新加载	是（始终在复位或 TC 产生时重新加载）	是（始终在复位或 TC 产生时重新加载）
时钟输入	限制为时钟系统中的数字时钟	任何信号

何时使用定时器

定时器的默认使用是生成周期性事件或中断信号。但是，也可以用于其他用途：

- 通过将时钟驱动到定时器时钟输入脚，并使用终端计数(TC)输出作为分频的时钟输出，来创建时钟分频器。
- 通过将时钟驱动到时钟输入并将测试信号驱动到使能或捕获输入，来测量硬件事件之间的时间长度。

注意：在侧重于对事件计数的情况下，使用计数器组件会更好。在需要具有更多控制功能的多个比较输出（如中心对齐、输出截止和死区输出）的情况下，使用 PWM 组件会更好。

定时器通常用于记录事件之间的时钟周期循环数。比如测量由转速计传感器生成的两个上升沿之间的时钟数。更加复杂的用法是测量 PWM 输入的周期和占空比。对于 PWM 测量，定时器组件配置为在上升沿时启动，捕获下一个下降沿，然后捕获并停止在下一个上升沿。最终捕获时的中断通知 CPU 所有捕获的值都已在 FIFO 中就绪。

输入/输出连接

本章节介绍定时器的各种输入和输出连接。某些 I/O 在其描述所列的条件下其符号可能不显示。

注意：除非另有指定，否则所有信号都是高电平有效。

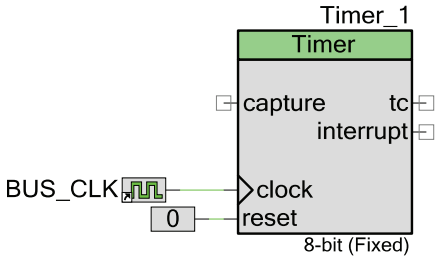
输入	可被隐藏	说明
clock	不可以 (N)	时钟输入定义定时器组件的工作频率。即，在定时器组件使能时，定时器周期计数器值在此输入的上升沿上递减。

输入	可被隐藏	说明
reset	不可以 (N)	此输入是同步复位。它至少需要时钟的一个上升沿以实现计数器值和捕获计数器的复位。它将周期计数器复位为周期值。它还复位捕获计数器。
使能	可以(Y)	此输入是定时器硬件使能。此连接使周期计数器可以在时钟的每个上升沿上递减。如果此输入为低电平，则输出仍有效，但是定时器组件工作状态不变。当 **Enable Mode**（启用模式）参数设置为 **Hardware Only**（仅硬件）或 **Software and Hardware**（软件和硬件）时，此输入可见。
捕获	可以(Y)	捕获输入将捕获当前计数值到捕获寄存器或 FIFO 中。如果 **Capture Mode**（捕获模式）参数设置为 **None**（无）之外的任何模式，则该输入可见。根据 **Capture Mode**（捕获模式）设置，捕获可以在此输入的上升沿、下降沿或任意沿上产生。捕获输入在时钟输入上采样。如果禁用定时器，则不捕获任何值。捕获输入可以保持悬空，而不进行外部连接。如果没有任何对象连接到捕获线，则组件会向其分配逻辑常量 0。
触发	可以(Y)	触发输入基于可配置硬件事件启用定时器及停止计数。如果 **Trigger Mode**（触发模式）参数设置为 **None**（无）之外的任何模式，则该输入可见。该输入使定时器延迟计数，直至检测到合适的沿。没捕获到触发沿，它也不生成中断。

输出	可被隐藏	说明
tc	不可以 (N)	终端计数是指示计数值等于零时的同步输出。该输出与定时器的时钟输入同步。此输出的时序准确性取决于器件以及是使用 UDB 还是 FF 实现。
中断	不可以 (N)	中断输出由硬件中配置的中断源驱动。所有中断源一起进行“或”运算以形成最终输出信号。中断源可以是：终端计数、捕获或 FIFO 已满。 在触发中断之后，中断输出保持置位，直至读取了状态寄存器。 在 PSoC 5 上用 FF 实现定时器，不支持中断连接。如果需要此功能，则可以将中断组件连接到 tc 信号，也可以使用 UDB 实现。
capture_out (捕获输出)	可以(Y)	capture_out 输出用于指示触发硬件捕获产生。此信号只在用 UDB 实现定时器时有效。此输出与定时器的时钟输入同步。

原理图宏信息

组件目录中的默认定时器是使用带默认设置的定时器组件的原理图宏。它连接到总线时钟和一个逻辑低电平组件上。



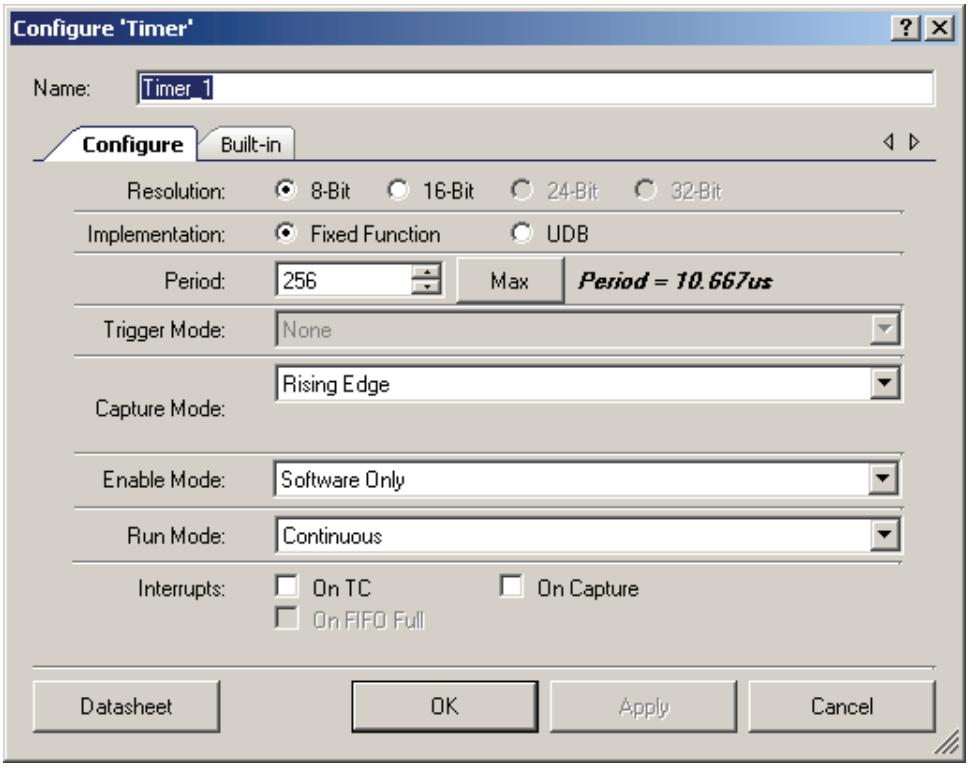
元件参数

将定时器拖入设计中，双击它以打开 **Configure**（配置）对话框。

硬件与软件配置选项

硬件配置选项用于更改放置在硬件中的综合能力。如果您对任何这些选项进行了更改，则必须重新构建硬件。软件配置选项不影响综合能力或放置。如果在构建之前设置这些参数，则需要设置其初始值。可以使用提供的 API 随时修改这些值。下面章节中描述的大多数参数是硬件选项。软件选项也将一同说明。

“配置”选项卡



分辨率

Resolution（分辨率）参数定义定时器的位宽度分辨率。此值可以设置为 8、16、24 或 32，分别对应于最大计数值 255、65535、16777215 和 4294967295。用 FF 实现分辨率限制为 8 或 16 位。

实现

通过 **Implementation**（实现）参数可以在固定功能模块 (FF) 实现和 UDB 之间选择定时器的实现方式。如果选择 FF，则禁用 UDB 功能。

周期（软件选项）

Period（周期）参数定义计数器的周期。定时器组件的最大计数值（或翻转点）等于 **Period**（周期）减一。**Period**（周期）减一是加载到周期寄存器中的初始值。软件可以使用 Timer_WritePeriod() API 随时更改此寄存器。若要使用此 API 获得等效结果，必须将自定义窗口提供的 **Period**（周期）值减一用作函数中的参数。

此值的限制范围,通过 **Resolution**（分辨率）参数进行定义。对于 8、16、24 和 32 位 **Resolution**（分辨率），**Period**（周期）分别为：2^8、2^16、2^24 和 2^32 或 256、65536、16777216 和 4294967296。

触发模式（软件选项）

Trigger Mode（触发模式）参数定义触发输入的实现方式。此参数仅当 **Implementation**（实现）设置为 **UDB** 时有效。

Trigger Mode（触发模式）可以设置为以下任何值：

- **None**（无）（默认）– 无任何触发方式实现，隐藏触发输入引脚
- **Rising Edge**（上升沿）– 在触发输入的第一个上升沿触发（使能）计数
- **Falling Edge**（下降沿）– 在触发输入的第一个下降沿触发（使能）计数
- **Either Edge**（任意沿）– 在触发输入的第一个沿（上升或下降）触发（使能）计数
- **Software Controlled（软件控制）–** 可以调用 Timer_SetTriggerMode() API，在运行时将触发模式设置为以上四个触发模式之一。在使用此 API 设置其他值之前，默认触发器为 **None**（无）。

捕获模式（软件选项）

Capture Mode（捕获模式）部分包含三个参数：**Capture Mode（捕获模式值）**、**Enable Capture Counter**（使能捕获计数器）和 **Capture Count**（捕获计数）。

捕获模式

Capture Mode（捕获模式）参数配置进行捕获的时间。捕获输入在时钟输入的上升沿采样。此模式可以设置为以下任何值（对于固定功能实现，只能使用 **None**（无）和 **Rising Edge**（上升沿））：

- **None**（无）– 无任何捕获产生，隐藏捕获输入引脚
- **Rising Edge**（上升沿）– 在与时钟输入同步的捕获输入的上升沿捕获计数器值。
- **Falling Edge**（下降沿）– 在与时钟输入同步的捕获输入的下降沿捕获计数器值。

- **Either Edge**（任意沿）– 在与时钟输入同步的捕获输入的任意沿捕获计数器值。
- **Software Controlled**（软件控制）– 可以调用 Timer_SetCaptureMode() API，在运行时将捕获模式设置为以上四个捕获模式之一。在使用此 API 设置其他值之前，默认触发器为 **None**（无）。

使能捕获计数器（软件选项）

通过 **Enable Capture Counter**（使能捕获计数器）参数可以定义在实际捕获计数器捕获之前发生的捕获事件数。例如，可能需要每三个事件捕获一次，在这种情况下，应将捕获计数器的值设置为 3。此参数只能用于 UDB 实现。

捕获计数（软件选项）

Capture Count（捕获计数）参数设置在实际捕获计数器之前发生的捕获事件的初始数量。它可以设置为介于 2 到 127 之间的值。可以通过调用 API 函数 Timer_SetCaptureCount()，在运行时修改捕获计数值。此参数只能用于 UDB 实现时。

使能模式

Enable Mode（使能模式）参数配置定时器的使能实现方式。使能输入在时钟输入的上升沿采样。此模式可以设置为以下任何值：

- **Software Only**（仅软件）– 定时器仅基于控制寄存器的使能位进行使能。
- **Hardware Only**（仅硬件）– 定时器仅基于使能输入。（仅用于 UDB 实现组件）
- **Software and Hardware**（软件和硬件）– 如果硬件和软件使能条件都为真，则使能定时器。

运行模式

通过 **Run Mode**（运行模式）参数可以将定时器组件配置为连续运行或以单触发模式运行：

- **Continuous**（连续）– 定时器在使能后连续运行。
- **One Shot**（单触发）– 定时器开始计数并在达到零时停止计数。在复位之后，它开始另一个循环。停止时，对于 UDB 定时器，它将 **Period**（周期）重新加载到计数寄存器中；对于固定功能定时器，计数寄存器保留终端计数值。
- **One Shot (Halt on Interrupt)**（单触发（中断时停止）– 定时器开始计数并在达到零或发生中断时停止计数。在复位之后，它开始另一个循环。停止时，对于 UDB 定时器，它将 **Period**（周期）重新加载到计数寄存器中；对于固定功能定时器，计数寄存器保留终端计数值。

注意：若要确保 One Shot（单触发）模式不会过早开始，应使用 **Trigger Mode**（触发模式）控制开始时间，或使用某种形式的软件启用模式（**Software Only**（仅软件）或 **Software and Hardware**（软件和硬件））。

中断（软件选项）

通过 **Interrupt**（中断）参数可以配置初始中断源。在以下一个或多个所选事件发生时，会生成中断。软件可以随时重新配置此模式；此参数定义初始配置。

- **On TC**（TC 时）– 此参数始终有效；默认情况下会清除它。
- **On Capture（捕获时）(1-4) –** 使您可以在发生给定数量的捕获时中断；默认情况下会清除它。
- **On FIFO Full（FIFO 已满时）–** 使您可以在捕获 FIFO 已满时中断；默认情况下会清除它。

时钟选择

有关 PSoC 3 或 PSoC 5 时钟系统的详细信息，请参见时钟组件数据手册和相应器件数据手册。

固定功能组件

配置为使用器件中的 FF 模块时，定时器组件具有以下限制：

- 时钟输入必须为来自时钟系统的数字时钟。
- 如果时钟频率要与总线时钟相同，则时钟必须为实际总线时钟。

  打开相应时钟组件的 **Configure**（配置）对话框以将 **Clock Type**（时钟类型）参数配置为 **Existing**（现有），并将 **Source**（源）参数配置为 **BUS_CLK**。不能从任何其他源（如主控时钟、IMO 等）分频得到此频率的时钟。

对于基于 UDB 的组件

可以将来自任何源的任何数字信号连接到时钟输入。该信号的频率受本数据手册直流电和交流电电气特性（UDB 实现）一节中定义的频率范围的限制。

放置

PSoC Creator 基于 **Implementation**（实现）参数将定时器组件放置在器件中。如果设置为 **Fixed Function**（固定功能），则将此组件放置在任何可用的 FF 计数器/定时器模块中。如果设置为 **UDB**，则将此组件放置在采用尽可能最佳配置的 UDB 阵列中。

资源

分辨率	数字模块					API 占用内存空间（字节）		引脚（每个外部 I/O）
	数据路径	宏单元	状态寄存器	控制寄存器	计数器 7	闪存	RAM	
8 位 UDB 定时器 [1]	1	6	1	1	0	257	5	-
8 位 FF 定时器 [2]	0	0	0	0	0	234	2	-
16 位 UDB 定时器 [1]	2	6	1	1	0	295	6	-
16 位 FF 定时器 [2]	0	0	0	0	0	248	2	
24 位 UDB 定时器 [1]	3	6	1	1	0	287	8	-
32 位 UDB 定时器 [1]	4	6	1	1	0	287	8	-
8 位 UDB 定时器单触发 [3]	1	8	1	1	0	257	5	-
16 位 UDB 定时器单触发 [3]	2	8	1	1	0	295	6	-

应用程序编程接口

应用程序编程接口 (API) 子程序允许您使用程序配置组件。下表列出了每个函数的接口，并进行了说明。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 将实例名称“Timer_1”分配给给定设计中的第一个定时器组件实例。您可以将其重命名为遵循标识符语法规则的任何唯一值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。出于可读性考虑，下表中使用的实例名称为“Timer”。

函数	说明
Timer_Start()	设置 initVar 变量，调用 Timer_Init() 函数，然后调用 Enable 函数。

[1] 该 UDB 定时器分辨率配置仅对应软件使能模式、上升沿触发模式、连续运行模式和 TC 时中断（无捕获模式）。

[2] 该 FF 定时器分辨率配置仅对应软件使能模式、上升沿捕获模式、连续运行模式和 TC 时中断配。

[3] 该 UDB 定时器分辨率配置仅对应软件使能模式、上升沿触发模式、单触发模式和 TC 时中断（无捕获模式）。

函数	说明
Timer_Stop()	禁用定时器。
Timer_SetInterruptMode()	使能或禁用中断输出源。
Timer_ReadStatusRegister()	返回状态寄存器的当前状态。
Timer_ReadControlRegister()	返回控制寄存器的当前状态。
Timer_WriteControlRegister()	设置控制寄存器的位字段。
Timer_WriteCounter()	将新值直接写入计数寄存器。（仅用于 UDB 实现）
Timer_ReadCounter()	强制捕获，然后返回捕获值。
Timer_WritePeriod()	对周期寄存器写操作。
Timer_ReadPeriod()	读取周期寄存器。
Timer_ReadCapture()	返回捕获寄存器的内容或 FIFO 的输出。
Timer_SetCaptureMode()	设置进行捕获的硬件或软件条件。
Timer_SetCaptureCount()	设置在将计数器寄存器捕获到 FIFO 中之前要发生的捕获事件次数。
Timer_ReadCaptureCount()	报告捕获事件数的当前设置。
Timer_SoftwareCapture()	强制将计数器值捕获到捕获 FIFO
Timer_SetTriggerMode()	设置发生触发的硬件或软件条件。
Timer_EnableTrigger()	启用定时器的触发模式。
Timer_DisableTrigger()	禁用定时器的触发模式。
Timer_SetInterruptCount()	设置在触发中断之前要计数的捕获数。
Timer_ClearFIFO()	清除捕获 FIFO。
Timer_Sleep()	停止定时器并保存其当前配置。
Timer_Wakeup()	恢复定时器配置并重新启用定时器。
Timer_Init()	按 Configure（配置）对话框设置初始化或恢复定时器。
Timer_Enable()	启用定时器。
Timer_SaveConfig()	保存定时器的当前配置。
Timer_RestoreConfig()	恢复定时器的配置。

全局变量

变量	说明
Timer_initVar	指示定时器是否已初始化。变量将初始化为 0，并在第一次调用 Timer_Start() 时设置为 1。这允许第一次调用 Timer_Start() 子程序后组件无需重新初始化便可重新启动。 如果需要重新初始化组件，则可以在调用 Timer_Start() 或 Timer_Enable() 函数之前调用 Timer_Init() 函数。

void Timer_Start(void)

说明： 这是开始执行组件操作的首选方法。Timer_Start() 设置 initVar 变量，调用 Timer_Init() 函数，然后调用 Timer_Enable() 函数。

参数： 无

返回值： 无

副作用： 如果已设置 initVar 变量，则该函数仅调用 Timer_Enable() 函数。

void Timer_Stop(void)

说明： 对于用固定功能实现的定时器，这会禁用定时器并关闭其电源。对于用 UDB 实现定时器，仅在软件使能模式下禁用定时器。

参数： 无

返回值： 无

副作用： 因为通过此函数关闭了固定功能定时器电源，所以会将 TC 输出驱动为低电平。

void Timer_SetInterruptMode(uint8 interruptMode)

说明： 启用或禁用中断输出源。

参数： uint8：中断源。对于位定义，请参考本数据手册的模式寄存器一节。

返回值： 无

副作用： FF 与 UDB 之间的位定义位置不同。提供屏蔽 #defines (宏)以屏蔽差异。

uint8 Timer_ReadStatusRegister(void)

说明： 返回状态寄存器的当前状态。

参数： 无

返回值： uint8：当前状态寄存器值
对于位定义，请参考本数据手册的状态寄存器一节。

副作用： 在读取状态寄存器时，会清除(清零)这些位中的某些位。读取时的清除位在本数据手册的状态寄存器一节中定义。

uint8 Timer_ReadControlRegister(void)

说明： 返回控制寄存器的当前状态。当无需控制寄存器时（如以 UDB 实现的定时器，使能模式为仅硬件，捕获模式不是软件控制，并且触发模式不是软件控制），此 API 在此特殊情况下不可用。

参数： 无

返回值： uint8：控制寄存器位字段
对于位定义，请参考本数据手册的控制寄存器一节。

副作用： 无

void Timer_WriteControlRegister(uint8 control)

说明： 设置控制寄存器的位字段。当无需控制寄存器时（如以 UDB 实现的定时器，启用模式为仅硬件，捕获模式不是软件控制，并且触发模式不是软件控制），此 API 在此特殊情况下不可用。

参数： uint8：控制寄存器位字段
对于位定义，请参考本数据手册的控制寄存器一节。

返回值： 无

副作用： 无

void Timer_WriteCounter(uint8/16/32 counter)

说明： 将新值直接写入计数寄存器。此函数只能用于 UDB 实现的定时器。

参数： uint8/16/32：新计数值。对于 24 位定时器，该参数为 uint32。

返回值： 无

副作用： 覆盖计数值。这可能会导致终端计数输出或周期宽度上出现非预期的行为。该函数不是总能写入的，函数可能会中断。调用此函数前必须先禁用定时器。

uint8/16/32 Timer_ReadCounter(void)

说明： 强制捕获，然后返回捕获值。

参数： 无

返回值： uint8/16/32：当前计数值。对于 24 位定时器，返回类型为 uint32。

副作用： 返回捕获寄存器的内容或 FIFO 的输出（仅用于 UDB 实现的定时器）。

void Timer_WritePeriod(uint8/16/32 period)

说明： 写入周期寄存器。

参数： uint8/16/32：新周期值。对于 24 位定时器，该参数为 uint32。

返回值： 无

副作用： 在从周期寄存器重新加载计数器前，定时器的周期不会更改。

uint8/16/32 Timer_ReadPeriod(void)

说明： 读取周期寄存器。

参数： 无

返回值： uint8/16/32：当前周期值。对于 24 位定时器，返回类型为 uint32。

副作用： 无

uint8/16/32 Timer_ReadCapture(void)

说明： 返回捕获寄存器的内容或 FIFO 的输出 (UDB)。

参数： 无

返回值： uint8/16/32：当前捕获值。对于 24 位定时器，返回类型为 uint32。

副作用： 在 UDB 实现的定时器中，会从 FIFO 中删除该值。

void Timer_SetCaptureMode(uint8 captureMode)

说明： 设置捕获模式。仅对于 UDB 实现的定时器并且当 **Capture Mode**（捕获模式）参数设置为 **Software Controlled**（软件控制）时，此函数才可用。

参数： uint8：枚举捕获模式。另请参考控制寄存器一节：

```
Timer__B_TIMER__CM_NONE
Timer__B_TIMER__CM_RISINGEDGE
Timer__B_TIMER__CM_FALLINGEDGE
Timer__B_TIMER__CM_EITHEREDGE
Timer__B_TIMER__CM_SOFTWARE
```

返回值： 无

副作用： 无

void Timer_SetCaptureCount(uint8 captureCount)

说明： 设置执行捕获之前要计数的捕获事件数。仅对于 UDB 实现的定时器并且当在 Configure（配置）对话框中选择了 **Enable Capture Counter**（使能捕获计数器）参数时，此函数才可用。

参数： uint8 captureCount：在将计数值捕获到捕获 FIFO 之前要计数的捕获事件数。介于 2 到 127 之间的值有效。

返回值： 无

副作用： 无

uint8 Timer_ReadCaptureCount(void)

说明： 读取在 Timer_SetCaptureCount() 函数中设置的 captureCount 参数的当前值。仅对于 UDB 实现的定时器并且仅当在 Configure（配置）对话框中选择了 **Enable Capture Counter**（启用捕获计数器）参数时，此函数才可用。

参数： 无

返回值： uint8：当前捕获计数

副作用： 无

void Timer_SoftwareCapture(void)

说明： 强制将当前计数器值通过软件捕获到 FIFO 中。此函数只能用于 UDB 实现的定时器。

参数： 无

返回值： 无：

副作用： 无

void Timer_SetTriggerMode(uint8 triggerMode)

说明： 设置触发模式。仅对于 UDB 实现的定时器并且仅当 Trigger Mode（触发模式）参数设置为 Software Controlled（软件控制）时，此函数才可用。

参数： uint8：枚举捕获模式。另请参考控制寄存器一节。

```
Timer__B_TIMER__TM_NONE
Timer__B_TIMER__TM_RISINGEDGE
Timer__B_TIMER__TM_FALLINGEDGE
Timer__B_TIMER__TM_EITHEREDGE
Timer__B_TIMER__TM_SOFTWARE
```

返回值： 无

副作用： 无

void Timer_EnableTrigger(void)

说明： 启用触发器。仅当 **Trigger Mode**（触发模式）设置为 **Software Controlled**（软件控制）时，此函数才可用。

参数： 无

返回值： 无

副作用： 无

void Timer_DisableTrigger(void)

说明： 禁用触发模式。仅当 **Trigger Mode**（触发模式）设置为 **Software Controlled**（软件控制）时，此函数才可用。

参数： 无

返回值： 无

副作用： 无

void Timer_SetInterruptCount(uint8 interruptCount)

说明： 设置在针对 InterruptOnCapture 源生成中断之前要计数的捕获数。仅当启用了 InterruptOnCaptureCount 时，此函数才可用。

参数： uint8 interruptCount：在生成捕获中断之前要计数的捕获事件数。介于 0 到 3 之间的值有效。

返回值： 无

副作用： 无

void Timer_ClearFIFO(void)

说明： 清除捕获 FIFO。此函数只能用于 UDB 实现的定时器。另请参考本数据手册功能描述一节中的 UDB FIFO。

参数： 无

返回值： 无

副作用： 无

void Timer_Sleep(void)

说明： 这是准备让组件进入睡眠模式的首选子程序。Timer_Sleep() 保存当前组件状态，然后调用 Timer_Stop() 函数，并调用 Timer_SaveConfig() 以保存硬件配置。

在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 Timer_Sleep() 函数。有关电源管理函数的更多信息，请参考 PSoC Creator *System Reference Guide*（《系统参考指南》）。

参数： 无

返回值： 无

副作用： 对于 FF 实现的定时器，在低功耗模式下保留所有寄存器。对于 UDB 实现的定时器，会保存并恢复控制寄存器和计数值寄存器。此外，当调用 Timer_Sleep() 时，会存储使能状态，以免在未调用 Timer_Stop() 的情况下调用 Timer_Sleep()。

void Timer_Wakeup(void)

说明： 该函数是将组件恢复到调用 Timer_Sleep() 之前状态的首选子程序。Timer_Wakeup() 函数调用 Timer_RestoreConfig() 函数以恢复配置。如果组件在调用 Timer_Sleep() 函数前为使能状态，则 Timer_Wakeup() 函数将同样重新使能组件。

参数： 无

返回值： 无

副作用： 调用 Timer_Wakeup() 函数前未调用 Timer_Sleep() 或 Timer_SaveConfig() 函数可能会产生非预期的行为。

void Timer_Init(void)

说明： 根据自定义“配置”对话框的设置来初始化或恢复组件配置。一般情况下无需调用 Timer_Init()，因为 Timer_Start() 子程序会调用该函数并且它是开启组件操作的首选方法。

参数： 无

返回值： 无

副作用： 所有寄存器将设置为自定义“配置”对话框中的值。

void Timer_Enable(void)

说明： 激活硬件并开始执行组件操作。一般情况下无需调用 Timer_Enable()，因为 Timer_Start() 子程序会调用该函数，这是开启组件操作的首选方法。此函数针对软件控制的任一使能模式使能定时器。

参数： 无

返回值： 无

副作用： 如果 **Enable Mode**（使能模式）参数设置为 **Hardware Only**（仅硬件），则此函数不对定时器操作产生任何影响。

void Timer_SaveConfig(void)

说明： 此函数会保存组件配置和非保留寄存器。它还保存 Configure（配置）对话框中定义的或通过相应 API 修改过的当前组件参数值。此函数由 Timer_Sleep() 函数调用。

参数： 无

返回值： 无

副作用： 无

void Timer_RestoreConfig(void)

说明： 此函数会恢复组件配置和非保留寄存器。它还将组件参数值恢复为在调用 Timer_Sleep() 函数之前的值。

参数： 无

返回值： 无

副作用： 调用该函数前未调用 Timer_Sleep() 或 Timer_SaveConfig() 函数可能会产生非预期的行为。

固件源代码示例

PSoC Creator 在 Find Example Project（查找实例工程）对话框中提供了许多包括原理图和示例代码的示例项目。要获取组件特定的示例，请右键打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开开始页或**文件**菜单中的对话框。根据需要，使用对话框中的**过滤器选项**可缩小可选项目的列表。

有关更多信息，请参考 PSoC Creator 帮助中的“查找示例项目”主题。

功能描述

如前所述，可以针对多种用途配置定时器组件。本节更加详细地介绍这些配置。

常规操作

在时钟输入的每个上升沿，定时器组件始终递减计数。它在计数值达到零值之后的下一个时钟沿，从周期寄存器重新加载计数寄存器值。

定时器在通过硬件或软件（取决于配置设置）使能之前保持禁用状态。在调用 Timer_Start() 之前不能使用组件，因为此函数针对定义的配置设置定时器的工作寄存器。

定时器输出

可以监控和重新加载计数寄存器。tc 输出可用于监控计数寄存器的当前值；它在计数值为零时为高电平。

定时器输入

可以在硬件或固件中实现捕获操作。计数寄存器中的当前值会复制到捕获寄存器或 FIFO。固件随后可以读取捕获的值。

复位和使能功能使定时器组件可以与其他组件同步。定时器组件仅在被使能且未置于复位状态下才进行计数。还可以在发生触发器输入事件时启动计数。可以通过硬件或固件复位或启用它。所有触发都是硬件。

注意：FF 实现的定时器所有输入（捕获、复位和启用）都在 FF 定时器中进行双同步。同步器以 BUS_CLK 时钟频率运行。这会在这些信号进入时与它们生效时产生延迟。该延迟取决于 BUS_CLK 与运行定时器的时钟之间的比率。以 FF 实现的定时器显示的所有波形都在信号同步之后显示信号。

定时器中断

中断输出可用于将事件发生通知给 CPU 或其他组件。可以将中断设置在发生一个或多个事件组合时激活。应仔细设计中断处理程序，以便可以确定中断源以及它是对沿还是电平敏感，并清除中断源。

定时器寄存器

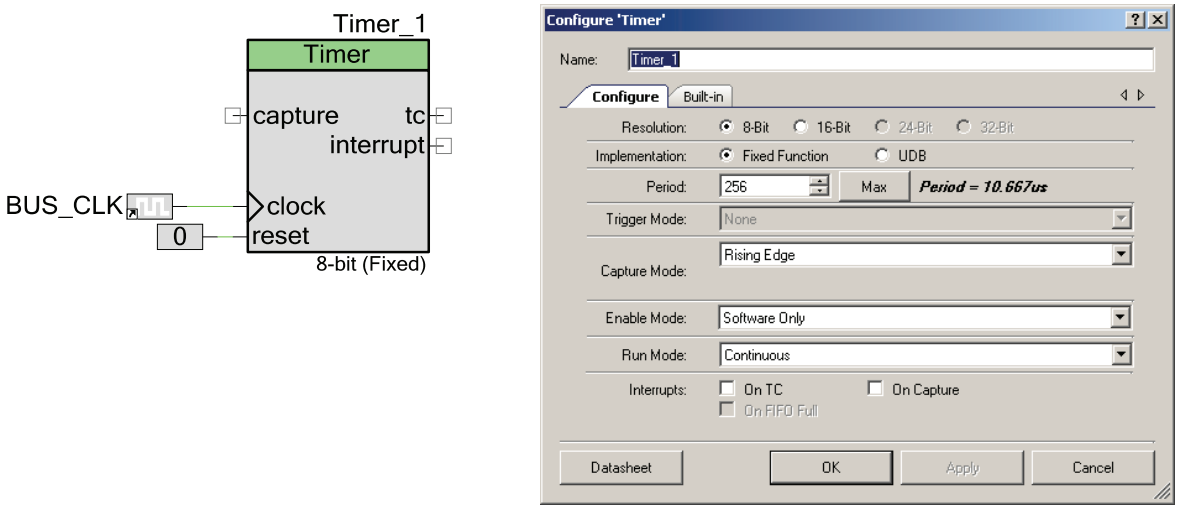
有三个寄存器：模式、状态和控制。请参考寄存器一节。

配置

默认配置

在将定时器组件拖动到 PSoC Creator 原理图上时，默认配置为一个 8 位 FF 定时器，该定时器在时钟输入的上升沿递减计数寄存器。图 1 显示默认原理图宏和 Configure（配置）对话框。

图 1. 默认定时器配置



此定时器的精度功能因不同实现方式和不同芯片而异。以下各图显示以 UDB 实现以及以 FF 实现在不同芯片上此定时器的功能。

针对 UDB 实现定时器时的默认配置功能在图 2 中显示。

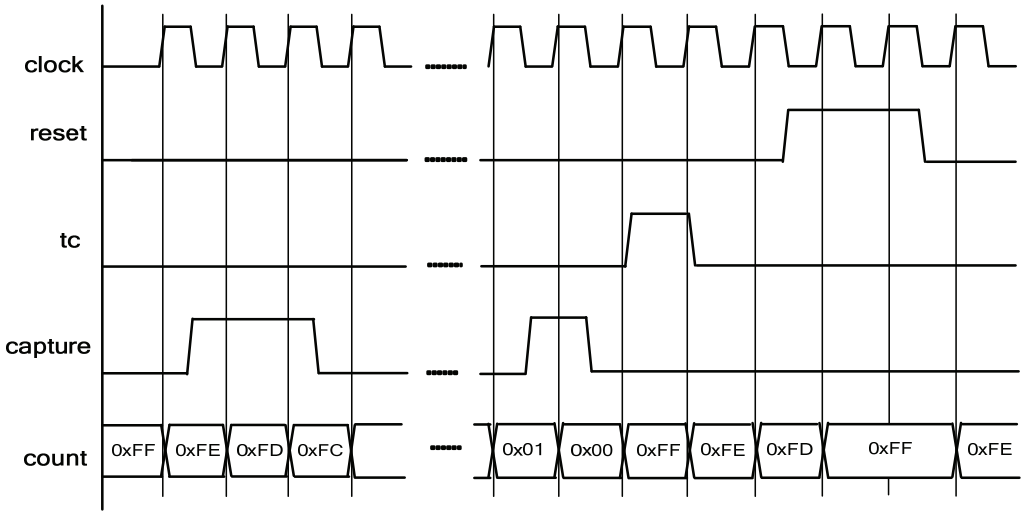
计数值在定时器配置过程中预加载，并在每次计数值达到零时重新加载。在默认配置中，**Period**（周期）设置为 256。它将 0xFF 加载到计数寄存器中，因为从 0xFF 计数到 0 会生成 256 的周期。

复位信号强制从周期寄存器重新加载计数值。在移除复位信号之前，计数值会保持此状态。

终端计数指示定时器已递减计数到零。它在计数值达到零时的下个时钟周期内有效。终端计数信号不基于复位事件发生而生成。

默认情况下，捕获功能配置为在捕获输入的每个上升沿捕获。无论捕获脉冲宽度如何，都捕获单个值。在此示例中，会捕获值 0xFE 和 0x01，并且可以由 CPU 读取这些值。

图 2. 默认 UDB 实现得定时器示例波形



针对 PSoC 3 上的固定功能实现的定时器的默认功能配置在图 3 中显示。

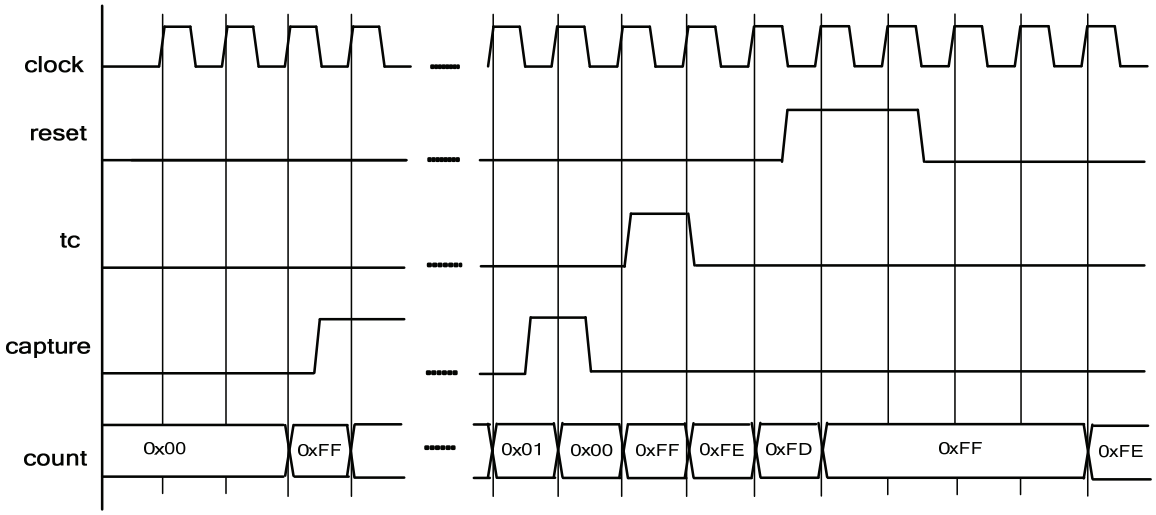
对于固定功能实现的定时器，在配置时不会预加载计数值；而且计数值从零值开始。对于 PSoC 3，这会导致 FF 实现相对于 UDB 实现有三个时钟周期的初始化延迟时间。它们分别是定时器开始计数之前的两时钟周期延迟和一个时钟周期用于从周期寄存器加载计数值。在定时器运行之后，周期与 UDB 实现相同。

复位信号强制计数值从周期寄存器加载并保持该计数，直至移除了复位。移除了复位之后，在计数器开始递减计数之前又有两时钟周期的延迟。

终端计数指示定时器已递减计数到零。它在计数值达到零时的下一个时钟周期内激活。终端计数信号不基于复位事件发生或由于初始化计数值为零而生成。

默认情况下，捕获功能配置为在捕获输入的每个上升沿捕获。无论捕获脉冲宽度如何，都捕获单个值。在此示例中，会捕获值 0xFF 和 0x01，并且可以由 CPU 读取这些值。此功能与 UDB 实现的定时器相同。

图 3. 默认 PSoC 3 FF 实现的定时器示例波形



针对 PSoC 5 上的固定功能实现的定时器时默认功能配置在图 4 中显示。

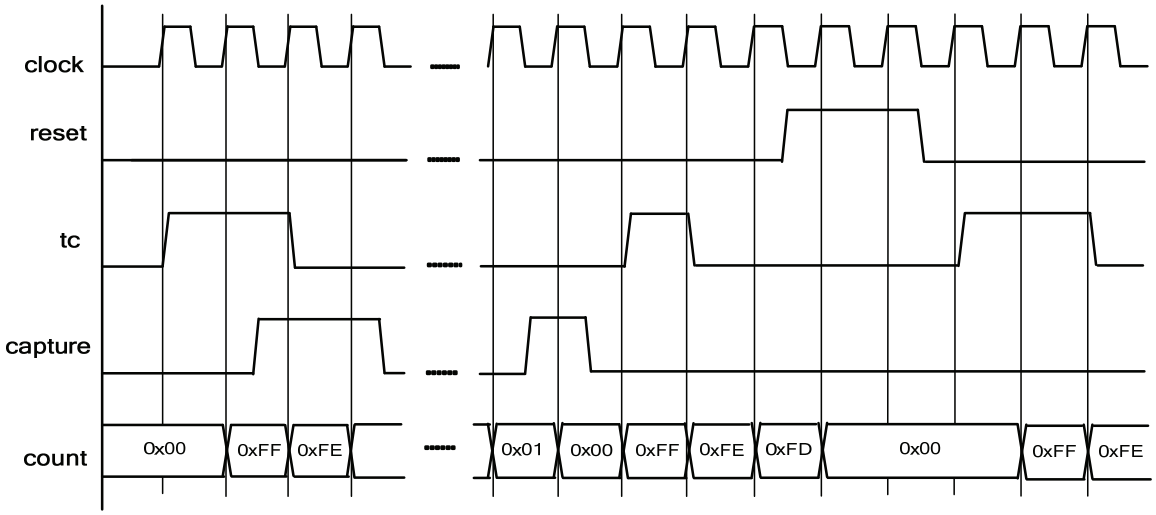
对于固定功能实现的定时器，在配置时不会预加载计数值；而且计数值从零值开始。对于 PSoC 5，这会导致 FF 相对于 UDB 实现定时器有两时钟周期的初始化延迟时间。这是定时器开始计数之前的一循环延迟和一个用于从周期寄存器加载计数器的循环。在定时器运行之后，周期与 UDB 实现的定时器相同。

复位信号强制计数值清除并保持为零，直至移除了复位。复位之后的功能与初始状态的功能类似，只不过第一个周期比 UDB 实现的定时器要长两个时钟周期。

终端计数指示定时器的值为零。结合计数值的初始值和复位时的值，这会在初始化时以及复位之后产生两个时钟周期宽 TC 脉冲。TC 在复位有效时保持低电平，但是随后在移除复位之后的两个循环内保持高电平。

默认情况下，捕获功能配置为在捕获输入的每个上升沿捕获。无论捕获脉冲宽度如何，都捕获单个值。在此示例中，会捕获值 0xFF 和 0x01，并且可以由 CPU 读取这些值。此功能与 UDB 实现的定时器相同。

图 4. 默认 PSoC 5 FF 实现的定时器示例波形

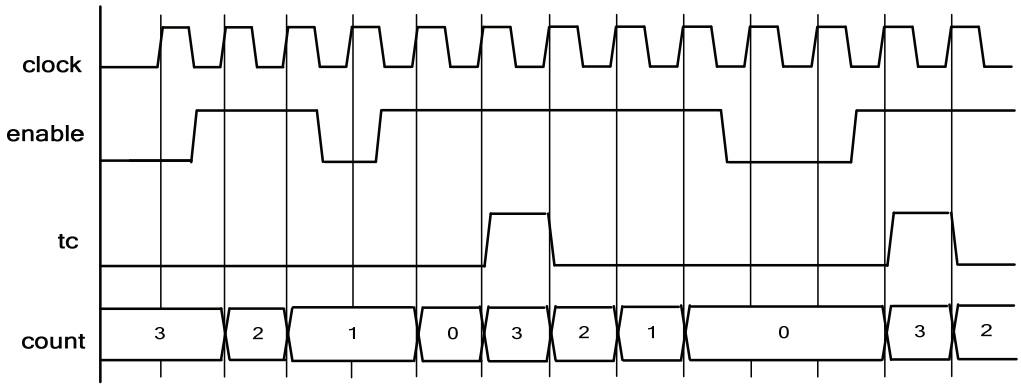


软件和硬件使能配置

硬件使能的功能因特定实现方式而异。针对使用 UDB 实现的定时器，配置软件和硬件使能时的定时器功能在图 5 中显示。

当使能了定时器时，计数值在每个时钟周期递减。在从周期寄存器重新加载计数值的一个时钟周期里，会生成单个时钟周期宽的终端计数脉冲。TC 信号始终为单个时钟周期宽的脉冲。请注意，它在重新加载循环周期期间发生。如果因为计数值达到零计数时计数处于禁用状态而延迟重新加载，则也会延迟 TC 脉冲，直至重新使能计数并重新加载计数值。如果由于复位信号而强制计数器重新加载，则不会生成 TC 脉冲。

图 5. 软件和硬件使能 UDB 实现的定时器示例波形

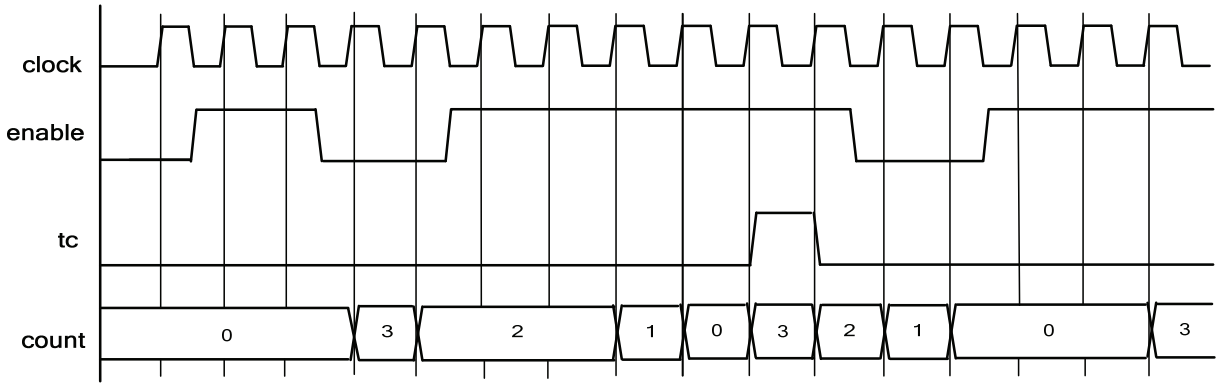


针对使用 PSoC 3 FF 实现的定时器配置软件和硬件使能时的功能在图 6 中显示。

计数的硬件使能与有效启用之间有两个时钟周期的延迟。因此如果早两个时钟周期内的使能信号为高电平，则计数值会递减。此延迟应用于启用和禁用计数。在从周期寄存器重新加载计数值的一个时钟周期里，会生成单个时钟周期宽的终端计数脉冲。TC 信号始终为单个时钟周期脉冲。

注意：如果定时器在计数器达到零之前的两个循环内具有低电平的启用信号，则不会在此定时器周期内生成 TC 输出脉冲。当重新使能定时器时，它会重新加载，而不生成 TC 信号。这显示在示例波形中。

图 6. 软件和硬件使能 PSoC 3 FF 实现的定时器示例波形

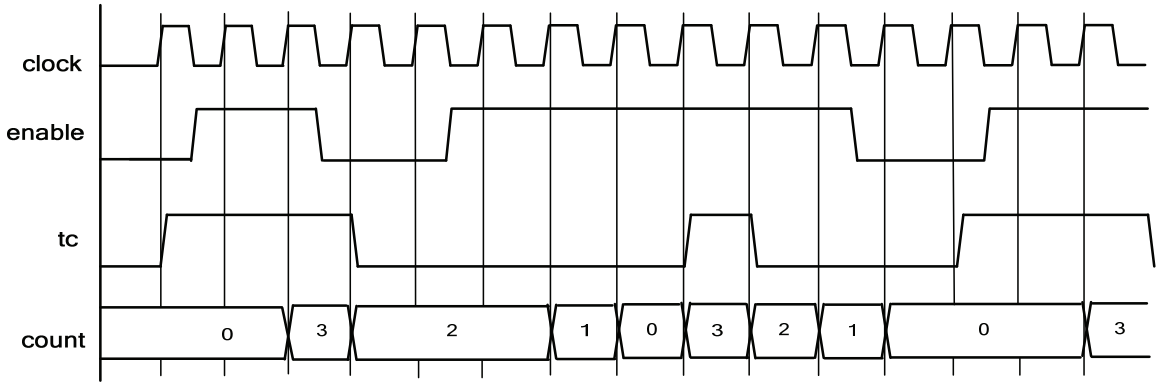


针对使用 PSoC 5 FF 实现的定时器，配置软件和硬件使能时的功能在图 7 中显示。

计数的硬件启用和与有效启用之间有一个时钟周期的延迟。因此使能信号为高电平一个时钟周期，则计数值会递减。此延迟应用于启用和禁用计数。只要计数器值等于零并且一时钟周期的延迟后，便会产生终端计数信号。这会在初始配置时发生。如果使能信号使定时器在计数值等于零时停止，则 TC 信号保持高电平。

注意：如果使能信号不能保持单个时钟周期，则硬件使能信号不会按预期方式工作。单时钟周期禁用脉冲将定时器锁定为该计数值，直至定时器再次禁用，然后重新使能。因此，硬件禁用信号必须始终保持为两个或更多时钟周期。单时钟周期使能使其按预期方式工作。

图 7. 软件和硬件使能 PSoC 5 FF 实现的定时器示例波形



单触发配置

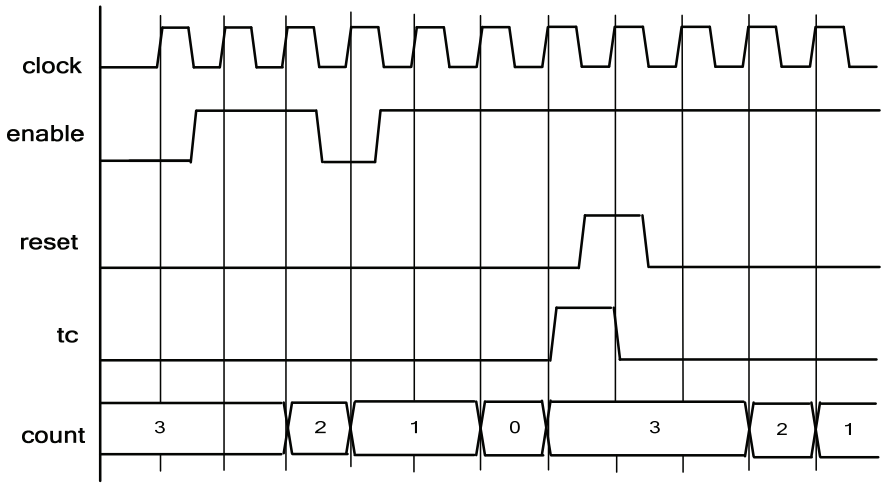
单触发运行模式的功能因特定实现方式而异。针对使用 UDB 实现的定时器配置单触发模式时的功能在图 8 中显示。

计数的硬件启用和与有效启用之间有一时钟周期的延迟。因此如果使能信号为高电平保持一个时钟周期，则计数值会递减。此延迟应用于使能和禁用计数器。这与连续运行模式下的行为（该行为在没有延迟的情况下计数）不同。

TC 信号始终为单个时钟周期宽的脉冲。请注意，它在重新加载计数值时产生。如果因为计数值达到零计数时处于禁用状态而延迟重新加载，则也会延迟 TC 脉冲，直至重新启用计数并重新加载计数值。如果由于复位信号而强制计数值重新加载，则不会生成 TC 脉冲。

在单触发周期完成之后，定时器可以设置为使用硬件复位运行下一个周期。硬件复位从周期寄存器重新加载计数值。在移除复位之后且硬件使能信号保持有效的一个时钟周期后，会使能定时器以递减计数。

图 8. 单触发运行 UDB 实现的定时器示例波形



针对 PSoC 3 上使用 FF 实现的定时器配置单触发运行时的功能在图 9 中显示。

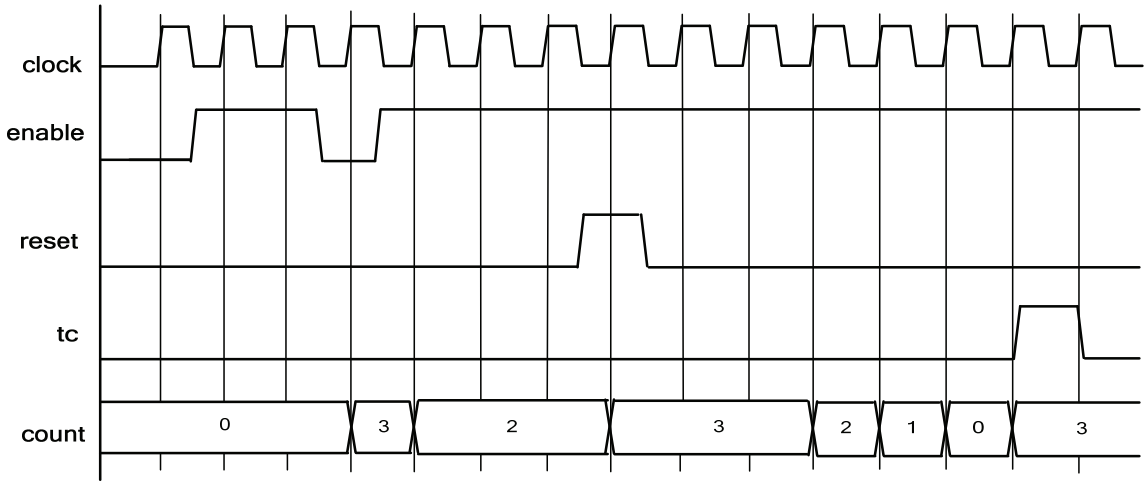
计数器的硬件启用和与有效启用之间有两个时钟周期的延迟。因此如果使能信号为高电平保持两个时钟周期，则计数值会递减。此延迟应用于使能和禁用计数。在从周期寄存器重新加载计数值的一个时钟周期内，会产生单个时钟周期宽的终端计数脉冲。TC 信号始终为单个时钟周期脉冲。这等效于连续运行模式下的操作。

单触发模式的一个额外特点（仅用于此实现方式）是在定时器开始计数之后，使能信号首次变为低电平时会将计数器停止在该值上。若要再次开始计数，必须复位定时器。

在单触发周期完成或者由于禁用使能信号而停止之后，定时器可以设置为使用硬件复位运行下一个周期。硬件复位从周期寄存器重新加载计数值。从释放复位到使能定时器以再次递减计数有两个时钟周期的延迟。

注意：对于此实现方式，只能使用 Timer_Stop() API，然后使用 Timer_Start() API 来重新启动定时器。这使计数器可以继续计数，但是它不会重新加载计数器值，因此只应将此方法用于计数器已完成一个周期并且已重新加载计数值的情况。

图 9. 单触发运行 FF PSoC 3 实现的定时器示例波形



针对 PSoC 5 上使用 FF 实现的定时器配置单触发运行时的功能在图 10 中显示。

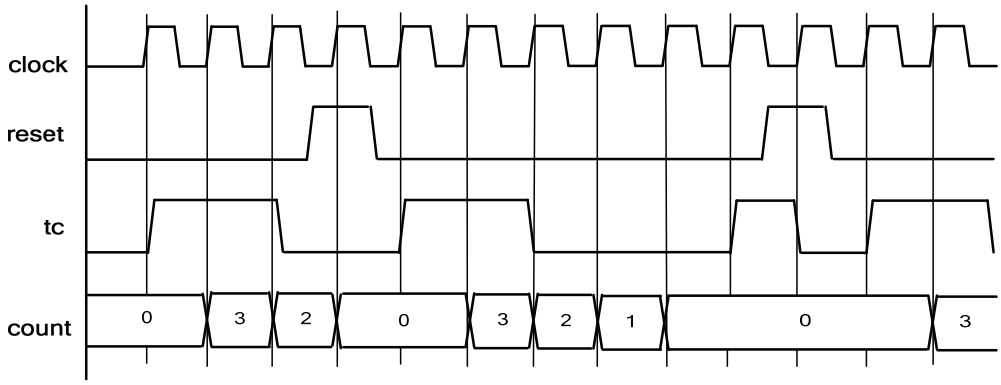
只要计数值等于零并且在一个时钟周期的延迟后，便会产生终端计数信号。这会在初始配置时发生。TC 信号在单触发周期完成之后保持高电平，因为计数器值保持为零值。在计数值为零时生成 TC 信号的一个例外情况是，当复位信号有效时，TC 始终保持为零。

在单触发周期完成之后，定时器可以设置为使用硬件复位运行下一个周期。硬件复位会使用零重新加载计数值并配置定时器以再次运行。从释放复位到使能定时器以再次递减计数有一个时钟周期的延迟。

注意：PSoC 5 FF 配置不支持单触发模式下的硬件使能。

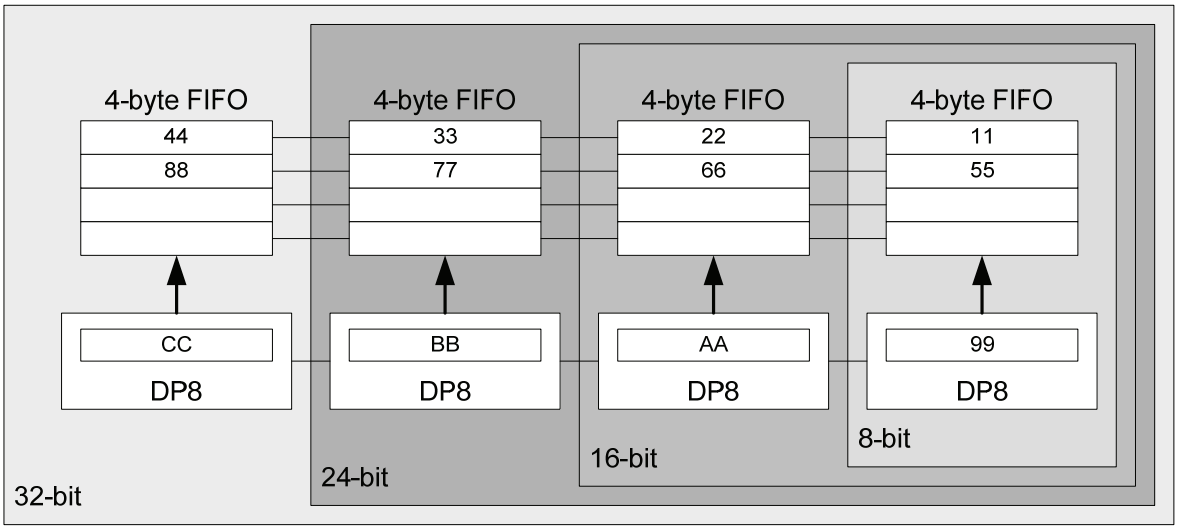
注意：因为 TC 信号在复位信号有效时保持低电平，所以每次单触发运行完成时会生成两个 TC 脉冲 第一个是在计数值计数到零时。第二个是在移除复位之后，不过在计数值开始计数之前。这显示在示例波形中。

图 10. PSoC 5 FF 实现的定时器在单触发下运行示例波形



UDB FIFO

UDB 数据路径 FIFO 用于捕获计数值。每个 FIFO 的深度为四个字节。采用多字节配置，会同时捕获计数值的每个字节到在指定 UDB 的 FIFO 中。因此，为了以避免丢失数据，在 CPU 必须读取捕获寄存器之前最多可以执行四次捕获。



Capture Value #1 = 0x44332211
Capture Value #2 = 0x88776655
Accumulator = 0xCCBBAA99

寄存器

状态寄存器

状态寄存器是只读寄存器，包含为定时器定义的状态位。使用 Timer_ReadStatusRegister() 函数可读取状态寄存器值。对状态寄存器进行的所有操作必须将以下定义用于位字段，因为这些位字段在 FF 与 UDB 实现之间可能不同。

状态寄存器中的某些位是粘滞的，这表示在它们设置为 1 之后，会保留该状态，直至在读取寄存器时它们才会被清除。状态数据在定时器的输入时钟沿上寄存，这通过所有粘滞位提供了定时器的时序分辨率。所有非粘滞位都是直接可读的，直接从状态寄存器的输入读取。

Timer_Status（UDB 实现）

位	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	FIFO 非空	FIFO 已满	捕获	TC
粘滞位	不可用	不可用	不可用	不可用	FALSE	FALSE	TRUE	TRUE

Timer_Status（固定功能实现）

位	7	6	5	4	3	2	1	0
名称	TC	捕获	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD
粘滞	TRUE	TRUE	不可用	不可用	不可用	不可用	不可用	不可用

位名称	头文件中的 #define (宏定义)	说明
TC	Timer_STATUS_TC	当计数值等于零时，此位变为 1。
捕获	Timer_STATUS_CAPTURE	只要触发了有效捕获事件，此位便会变为 1。这不包括软件捕获。
FIFO 已满	Timer_STATUS_FIFOFULL	当 UDB FIFO 达到已定义的四个捕获的已满状态时，此位变为 1。
FIFO 非空	Timer_STATUS_FIFONEMP	当 UDB FIFO 包含至少一个捕获值时，此位变为 1。

模式寄存器

模式寄存器是读/写寄存器，包含为计数定义的中断屏蔽位。使用 Timer_SetInterruptMode() 函数可设置模式位。对模式寄存器进行的所有操作必须使用以下定义的位字段，因为这些位字段在 FF 与 UDB 实现的定时器之间可能不同。

定时器组件所有中断源的中断输出是"或"的关系。可以通过模式寄存器中的对应位来启用或屏蔽每个源。

Timer_Mode（UDB 实现）

位	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	RSVD	FIFO 已满	捕获	TC

Timer_Mode（固定功能实现）

位	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	TC	捕获	RSVD	RSVD

位名称	头文件中的 #define(宏定义)	使能中断输出
TC	Timer_STATUS_TC_INT_MASK	计数寄存器等于 0
捕获	Timer_STATUS_CAPTURE_INT_MASK	捕获
FIFO 已满	Timer_STATUS_FIFOFULL_INT_MASK	UDB FIFO 已满

控制寄存器

控制寄存器允许您控制计数的常规操作。此寄存器使用 Counter_WriteControlRegister() 函数调用写入，使用 Counter_ReadControlRegister() 函数读取。对控制寄存器进行的所有操作必须使用以下定义的位字段，因为这些位字段在 FF 与 UDB 实现的定时器之间可能不同。

注意：当写入控制寄存器时，不得更改任何保留位。所有操作都必须是读操作修改写操作，同时屏蔽保留位。

Timer_Control（UDB 实现）

位	7	6	5	4	3	2	1	0
名称	使能	捕获模式 [1:0]		触发使能	触发模式 [1:0]		中断计数 [1:0]	

Timer_Control1（固定功能实现）

位	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	使能

位名称	头文件中的 #define(宏定义)	说明/枚举类型
中断计数	Timer_CTRL_INTCNT_MASK	中断计数位定义在触发中断之前要计数的捕获事件数。
触发模式	Timer_CTRL_TRIG_MODE_MASK	触发模式控制位定义了计划的触发输入的功能。此位字段在初始化时使用 **Trigger Mode**（触发模式）参数中定义的触发模式进行配置。 ■ Timer__B_TIMER__TM_NONE ■ Timer__B_TIMER__TM_RISINGEDGE ■ Timer__B_TIMER__TM_FALLINGEDGE ■ Timer__B_TIMER__TM_EITHEREDGE ■ Timer__B_TIMER__TM_SOFTWARE
触发使能	Timer_CTRL_TRIG_EN	触发使能位允许软件控制何时准备开始等待触发事件。
捕获模式	Timer_CTRL_CAP_MODE_MASK	捕获模式控制位是用于定义计划的捕获输入操作的两位字段。此位字段在初始化时使用 **Capture Mode**（捕获模式）参数中定义的捕获模式进行配置。 ■ Timer__B_TIMER__CM_NONE ■ Timer__B_TIMER__CM_RISINGEDGE ■ Timer__B_TIMER__CM_FALLINGEDGE ■ Timer__B_TIMER__CM_EITHEREDGE ■ Timer__B_TIMER__CM_SOFTWARE
使能	Timer_CTRL_ENABLE	启用软件控制计数。仅当 **Enable Mode**（启用模式）参数设置为 **Software Only**（仅软件）或 **Software and Hardware**（软件和硬件）时，此位才有效。

计数值（8、16、24 或 32 位，根据分辨率）

计数寄存器包含当前计数值。此寄存器在所有时钟输入的上升沿时递减。可以随时调用 Timer_ReadCounter() 函数读取此寄存器。

捕获（8、16、24 或 32 位，根据分辨率）

捕获寄存器包含捕获的计数值。任何捕获事件都会将计数寄存器的值复制到此寄存器。在 UDB 实现的定时器中，此寄存器实际上是 FIFO。有关详细信息，请参见 UDB FIFO 一节。

周期（基于 8、16、24 或 32 位分辨率）

周期寄存器包含调用 Timer_WritePeriod() 函数设置的周期值和 **Period**（周期）参数在初始化期间定义的周期值。周期寄存器的值会在发生重新加载事件时复制到计数寄存器中。

组件调试窗口

定时器组件支持 PSoC Creator 组件调试窗口。调试窗口中显示了以下寄存器。某些寄存器在 UDB 实现的定时器中可用（通过 * 指示），而某些寄存器仅在固定功能实现的定时器中可用（通过 ** 指示）。所有其他寄存器可用于任一配置。

寄存器： Timer_CONTROL

名称： 控制寄存器

说明： 有关位字段定义，请参考本数据手册前面的 Timer_Control 寄存器说明。

寄存器： Timer_CONTROL2 **

名称： 固定功能控制寄存器 #2

说明： 固定功能定时器模块具有第二个配置寄存器。有关位字段定义，请参考《技术参考手册》。

寄存器： Timer_STATUS_MASK *

名称： 状态寄存器中断屏蔽配置

说明： 允许您使能任何状态位作为组件中断输出引脚上的中断源。有关对应位字段的定义细节，请参考本数据手册前面的 Timer_Status 寄存器说明。

寄存器： Timer_STATUS_AUX_CTRL *

名称： 状态寄存器的辅助控制寄存器

说明： 允许您通过位字段 INT_EN 使能内部状态寄存器的中断输出。有关位字段定义，请参考《技术参考手册》。

寄存器： Timer_PERIOD

名称： 定时器周期寄存器

说明： 定义在每个定时器循环开始时重新加载到周期计数的周期值。

寄存器： Timer_COUNTER

名称： 定时器计数寄存器

说明： 指示当前定时器周期循环的当前计数值（在每个时钟周期中，从 **Period**（周期）值递减到零）。

寄存器： Timer_GLOBAL_ENABLE **

名称： 固定功能定时器全局使能寄存器

说明： 使能固定功能定时器以进行操作。有关位字段定义，请参考《技术参考手册》。

直流电和交流电电气特性（FF 实现）

下面的值表示了预计性能，它们基于初始特性数据。

定时器直流规范

参数	说明	条件	最小值	典型值	最大值	单位
	模块电流消耗	16 位定时器，在所列的输入时钟频率下	–	–	–	µA
	3 MHz		–	15	–	µA
	12 MHz		–	60	–	µA
	48 MHz		–	260	–	µA
	67 MHz		–	350	–	µA

定时器交流规范

参数	说明	条件	最小值	典型值	最大值	单位
	工作频率		DC	–	67	MHz
	捕获脉冲宽度（内部）		15	–	–	ns
	捕获脉冲宽度（外部）		30	–	–	ns
	定时器分辨率		15	–	–	ns
	使能脉冲宽度		15	–	–	ns
	使能脉冲宽度（外部）		30	–	–	ns
	复位脉冲宽度		15	–	–	ns
	复位脉冲宽度（外部）		30	–	–	ns

PSoC 5 的直流电和交流电电气特性（FF 实现）

下面的值表示了预计性能，它们基于初始特性数据。

定时器直流规范

参数	说明	条件	最小值	典型值	最大值	单位
	模块电流消耗	16 位定时器，在所列的输入时钟频率下	–	–	–	µA
	3 MHz		–	65	–	µA
	12 MHz		–	170	–	µA
	48 MHz		–	650	–	µA
	67 MHz		–	900	–	µA

定时器交流规范

参数	说明	条件	最小值	典型值	最大值	单位
	工作频率		DC	–	67.01	MHz
	捕获脉冲宽度（内部）		13	–	–	ns
	捕获脉冲宽度（外部）		30	–	–	ns
	定时器分辨率		13	–	–	ns
	使能脉冲宽度		13	–	–	ns
	使能脉冲宽度（外部）		30	–	–	ns
	复位脉冲宽度		13	–	–	ns
	复位脉冲宽度（外部）		30	–	–	ns

直流电和交流电电气特性（UDB 实现）

下面的值表示了预计性能，它们基于初始特性数据。

时序特性“额定路由的最大值”

参数	说明	配置	最小值	典型值	最大值	单位
f_{CLOCK}	组件时钟频率	8 位 UDB 定时器	–	–	40	MHz
		16 位 UDB 定时器	–	–	38	MHz
		24 位 UDB 定时器	–	–	33	MHz
		32 位 UDB 定时器	–	–	27	MHz

参数	说明	配置	最小值	典型值	最大值	单位
t_{clockH}	输入时钟高电平时间 [4]	不可用	–	0.5	–	t_{CY_clock}
t_{clockL}	输入时钟低电平时间 [4]	不可用	–	0.5	–	t_{CY_clock}
输入						
t_{PD_ps}	要同步的引脚的输入路径延迟 [5]	1	–	–	STA [6]	ns
t_{PD_ps}	要同步的引脚的输入路径延迟	2	–	–	8.5	ns
t_{PD_si}	同步输出到输入的路径延迟（路由）[5]	1,2,3,4	–	–	STA [6]	ns
t_{I_clk}	clockX 与时钟的校准	1,2,3,4	0	–	1	t_{CY_clock}
t_{PD_IE}	组件时钟的输入路径延迟（边沿感应输入）	1,2	$t_{PD_ps} + t_{SYNC} + t_{PD_si}$	–	$t_{PD_ps} + t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{PD_IE}	组件时钟的输入路径延迟（边沿感应输入）	3,4	$t_{SYNC} + t_{PD_si}$	–	$t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{IH}	输入高电平时间	1,2,3,4	t_{CY_clock}	–	–	ns
t_{IL}	输入低电平时间	1,2,3,4	t_{CY_clock}	–	–	ns

时序特性“所有路由的最大值”

参数	说明	配置	最小值	典型值	最大值 [7]	单位
f_{CLOCK}	组件时钟频率	8 位 UDB 定时器	–	–	20	MHz
		16 位 UDB 定时器	–	–	15	MHz
		24 位 UDB 定时器	–	–	20	MHz
		32 位 UDB 定时器	–	–	15	MHz
t_{clockH}	输入时钟高电平时间 [8]	不可用	–	0.5	–	$1/f_{clock}$

[4] $t_{CY_clock} = 1/f_{CLOCK}$。这是一个时钟周期的循环时间。

[5] t_{PD_ps} 和 t_{PD_si} 是路由路径延迟。由于路由是动态的，这些值可以更改，且将直接影响最大组件时钟和同步时钟频率。静态时序分析结果中一定能够找到这些值。

[6]配置 2 中的 t_{PD_ps} 是为器件的每个引脚定义的固定值。此处列出的数字是器件上可用的所有引脚的额定值。

[7] “所有路由的最大值”时序数值通过将“额定路由”时序数值按系数 2 降额来进行计算。如果组件实例以这些速度或更低速度运行，则对于此组件不应遇到时序问题。

[8] $t_{CY_clock} = 1/f_{CLOCK}$。 这是一个时钟周期的循环时间。

参数	说明	配置	最小值	典型值	最大值 [7]	单位
t_{clockL}	输入时钟低电平时间	不可用	–	0.5	–	$1/f_{clock}$
输入						
t_{PD_ps}	要同步的引脚的输入路径延迟 [9]	1	–	–	STA	ns
t_{PD_ps}	要同步的引脚的输入路径延迟 [10]	2	–	–	8.5	ns
t_{PD_si}	同步输出到输入的路径延迟（路由）[99]	1,2,3,4	–	–	STA [9]	ns
t_{I_clk}	clockX 与时钟的校准	1,2,3,4	0	–	1	t_{CY_clock}
t_{PD_IE}	组件时钟的输入路径延迟（边沿感应输入）	1,2	$t_{PD_ps} + t_{SYNC} + t_{PD_si}$	–	$t_{PD_ps} + t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{PD_IE}	组件时钟的输入路径延迟（边沿感应输入）	3,4	$t_{SYNC} + t_{PD_si}$	–	$t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{IH}	输入高电平时间	1,2,3,4	t_{CY_clock}	–	–	ns
t_{IL}	输入低电平时间	1,2,3,4	t_{CY_clock}	–	–	ns

[9] t_{PD_ps} 和 t_{PD_si} 是路由路径延迟。由于路由是动态的，这些值可以更改，且将直接影响最大组件时钟和同步时钟频率。静态时序分析结果中一定能够找到这些值。

[10]配置 2 中的 t_{PD_ps} 是为器件的每个引脚定义的固定值。此处列出的数值是器件上可用的所有引脚的额定值。

如何将 STA 结果用于特性数据

额定路由最大值是通过使用静态时序分析 (STA) 进行多次测试而收集的。您可以用下列方法，使用 STA 结果计算设计的最大值：

f_{CLOCK} 最大组件时钟频率显示在名为外部时钟的时钟汇总时序结果中。下图演示了_timing.html 中的时钟限制示例：

-Clock Summary

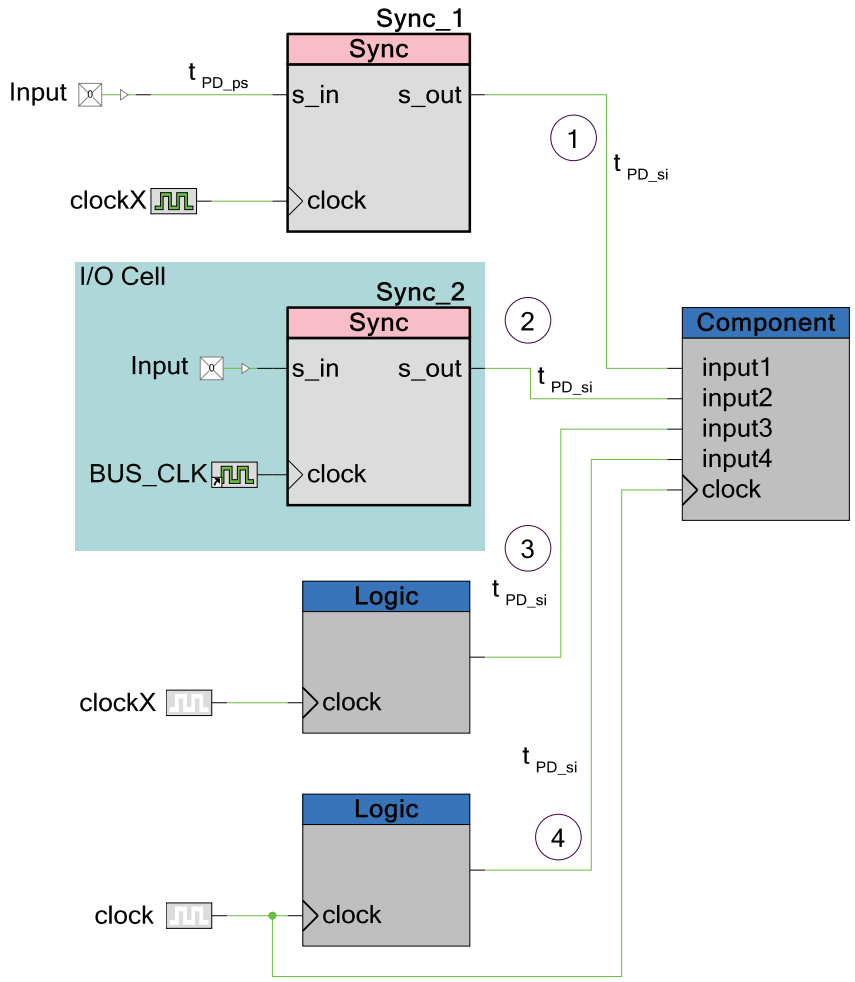
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

输入路径延迟和脉冲宽度

当要表现输入功能的特性时，所有输入（无论您如何配置它们）看上去都类似于四种可能配置之一，如图 11 所示。

必须同步所有输入。同步机制取决于组件输入源。为了完全解析您的系统如何工作，您必须了解已为每个输入设置了哪种输入配置以及系统的时钟配置。本节介绍如何使用静态时序分析 (STA) 结果确定系统的特性。

图 11. 组件时序规范的输入配置



配置	组件时钟	同步器时钟（频率）	图形
1	master_clock	master_clock	图 16

配置	组件时钟	同步器时钟（频率）	图形
1	clock	master_clock	图 14
1	clock	clockX = 时钟 [11]	图 12
1	clock	clockX > clock	图 13
1	clock	clockX < clock	图 15
2	master_clock	master_clock	图 16
2	clock	master_clock	图 14
3	master_clock	master_clock	图 21
3	clock	master_clock	图 19
3	clock	clockX = clock [11]	图 17
3	clock	clockX > clock	图 18
3	clock	clockX < clock	图 20
4	master_clock	master_clock	图 21
4	clock	clock	图 17

[11] 时钟频率相等，但是不保证上升沿的对齐。

1. 输入由器件引脚驱动，并在内部与“同步”组件同步。此组件同步采用与组件所使用时钟不同的内部时钟（所有内部时钟派生自 master_clock）。

   当要表现按此方法配置的输入的特性时，clockX 可以快于、等于或慢于组件时钟。它还可以等于 master_clock。这会生成如图 12、图 13、图 15 和图 16 所示的特性参数。

2. 输入由器件引脚驱动，并使用 master_clock 与引脚同步。

   当要表现按此方法配置的输入的特性时，master_clock 快于或等于组件时钟（不能慢于组件时钟）。这会生成如图 13 和图 16 所示的特性参数。

图 12. 输入配置 1 和 2：同步时钟频率 = 组件时钟频率（不保证时钟和 clockX 的边沿对齐）

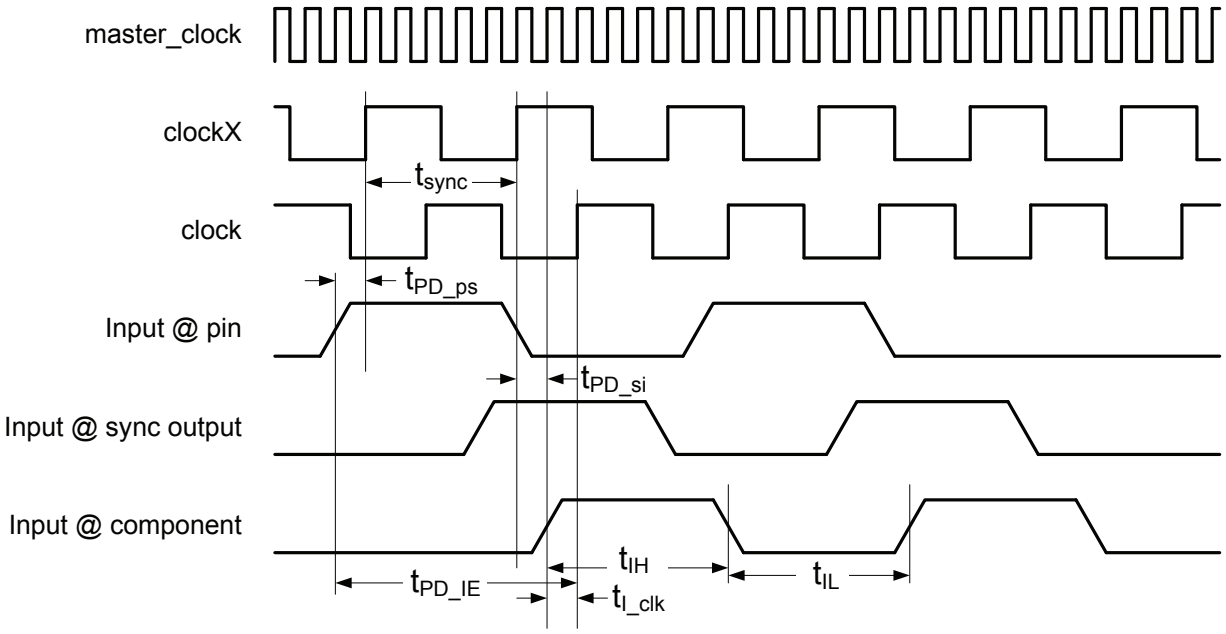


图 13. 输入配置 1 和 2：同步 时钟频率 > 组件时钟频率

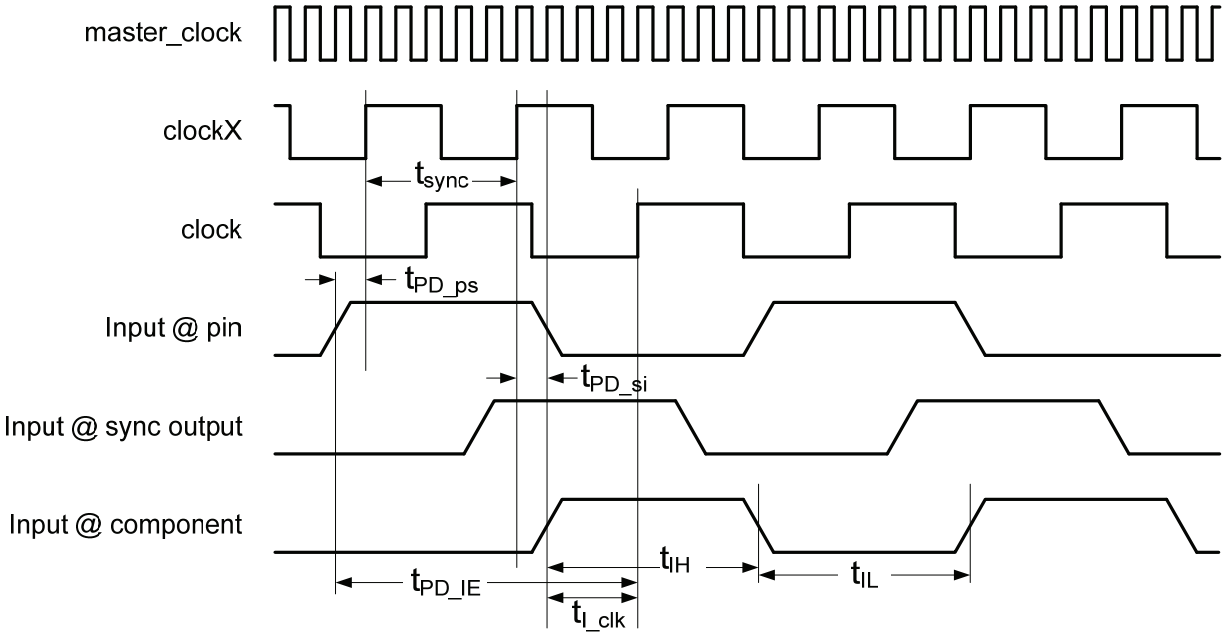


图 14. 输入配置 1 和 2：[同步 时钟频率 == master_clock] > 组件时钟频率

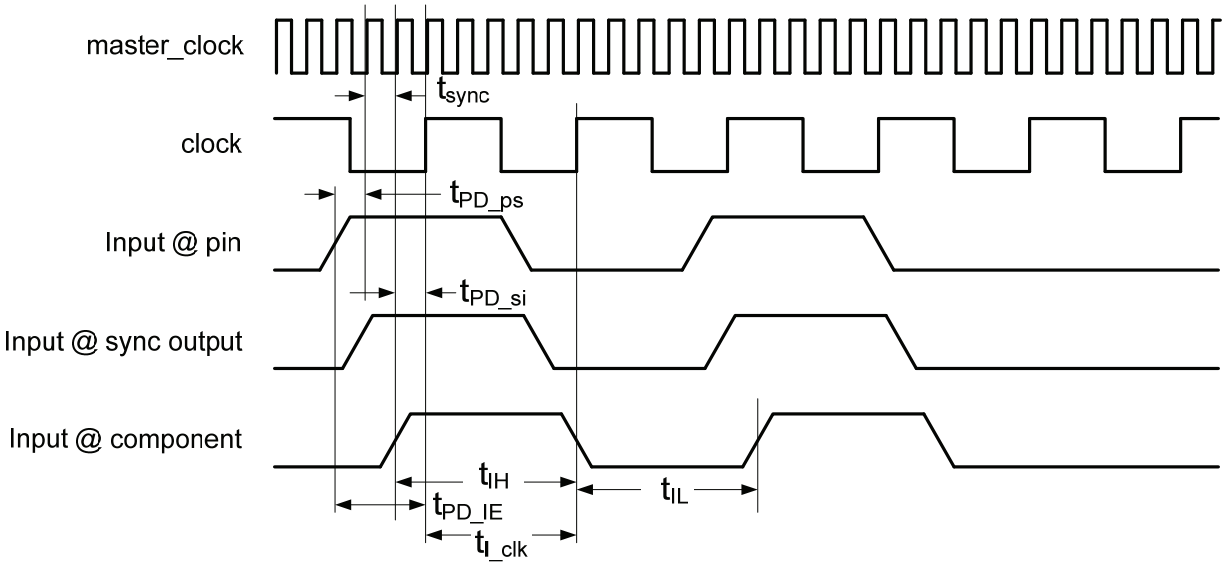


图 15. 输入配置 1：同步 时钟频率 < 组件时钟频率

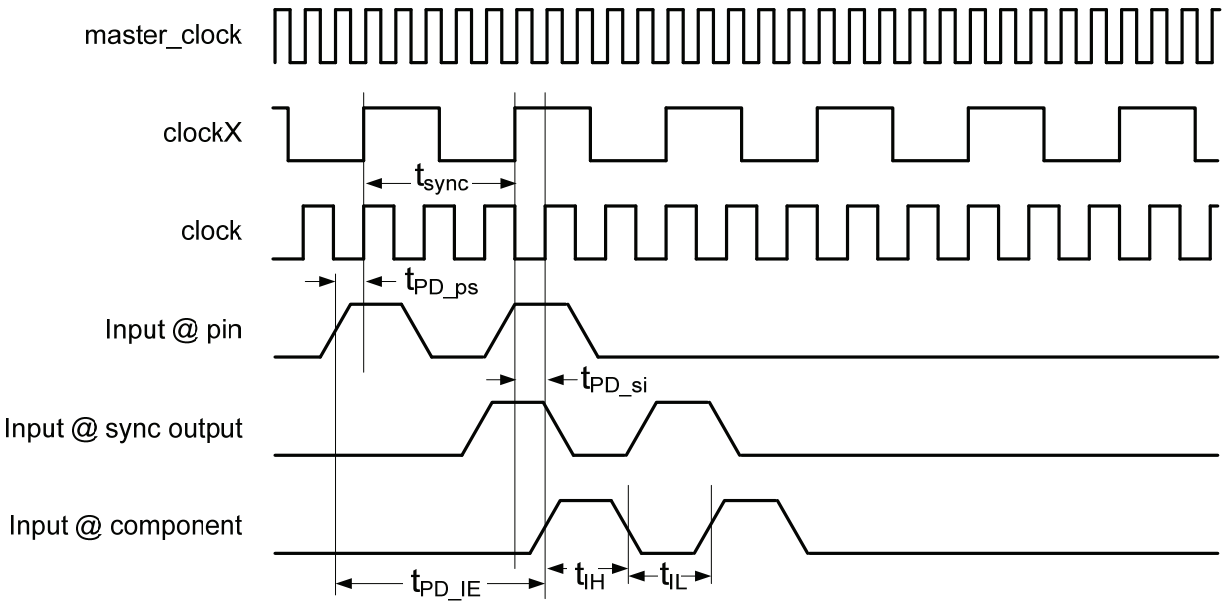
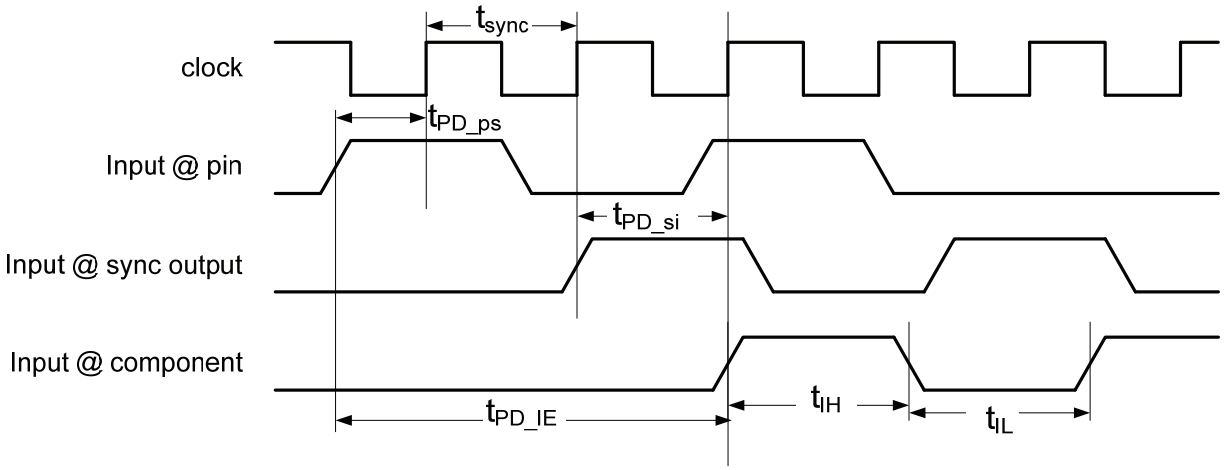


图 16. 输入配置 1 和 2：同步 时钟 = 组件时钟 = master_clock



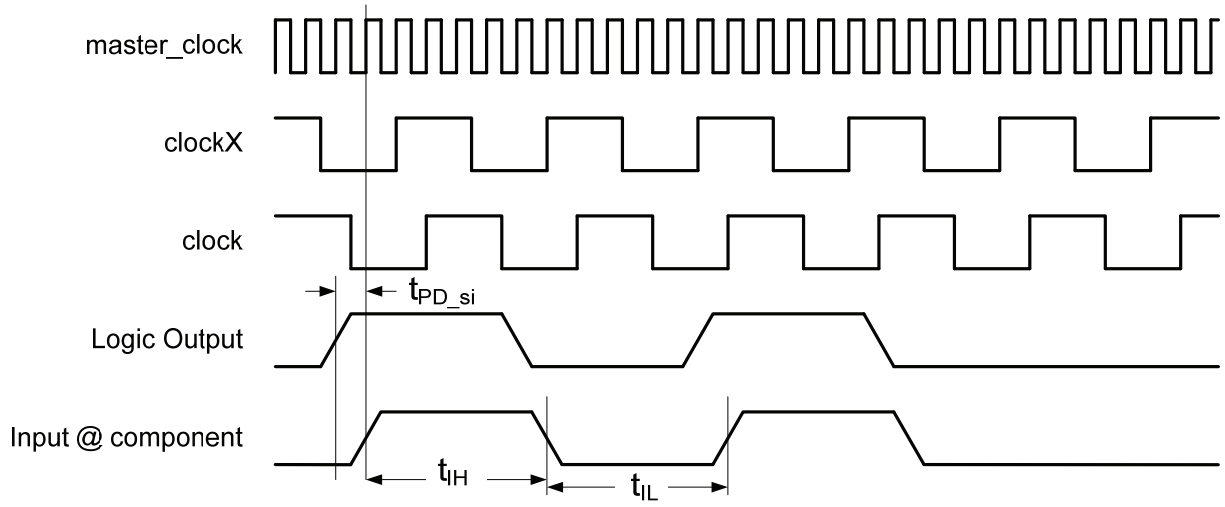
3. 输入由 PSoC 内部逻辑驱动，它基于与组件所使用的时钟不同的时钟同步（所有内部时钟都派生自 master_clock）。

   当要表现按此方法配置的输入的特性时，同步器时钟快于、慢于或等于组件时钟。这会生成如图 17、图 18 和图 20 所示的特性参数。

4. 输入由 PSoC 内部逻辑驱动，它基于与组件所使用的时钟同步。

   当要表现按此方法配置的输入的特性时，同步器时钟等于组件时钟。这会生成如图 21 所示的特性参数。

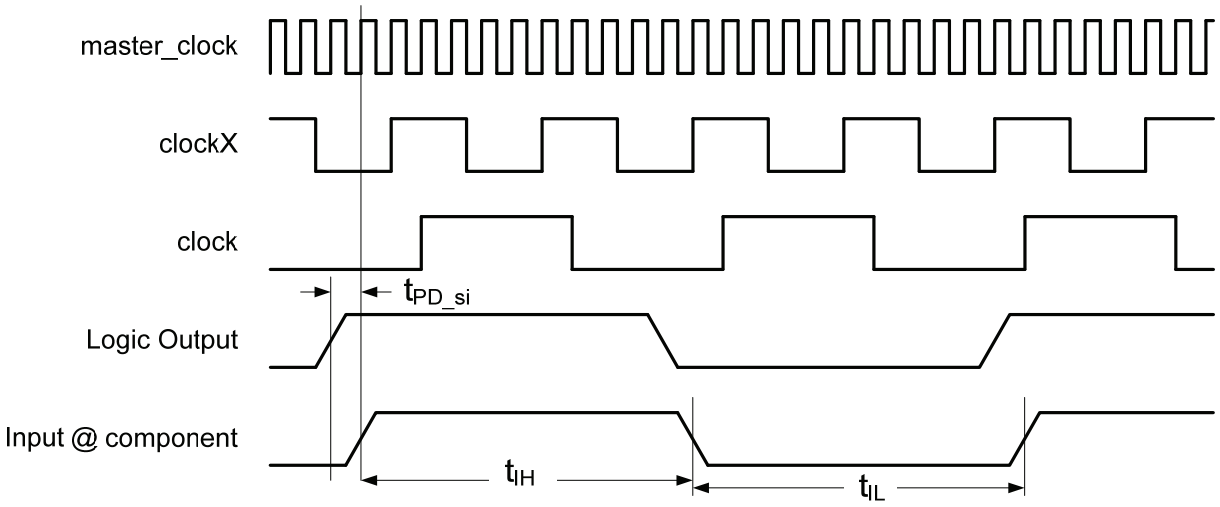
图 17. 仅输入配置 3：同步 时钟频率 = 组件时钟频率（不保证时钟和 clockX 的边沿对齐）



此图表示静态时序分析与时钟相关信息。数字时钟域中的所有时钟与 master_clock 同步。不过，具有相同频率的两个时钟的上升沿很可能不对齐。因此，静态时序分析工具不了解时钟同步到哪

个边沿，并且必须假设一个 master_clock 时钟周期的最小值。这意味着 t_{PD_si} 现在对系统 master_clock 的影响有限。 如果此路径延迟太长，则 master_clock 设置时间会出现冲突。您必须更改系统的同步时钟，或者以较慢的频率运行 master_clock。

图 18. 输入配置 3：同步 时钟频率 > 组件时钟频率



与图 17 中的方法几乎相同，所有时钟都派生自 master_clock。STA 在此配置中指明了 t_{PD_si} 对 master_clock 的限制要在一个 master_clock 时钟周期里。如果此路径延迟太长，则 master_clock 设置时间出现冲突。您必须更改系统的同步时钟，或者以较慢的频率运行 master_clock。

图 19. 输入配置 3：同步器时钟频率 = master_clock > 组件时钟频率

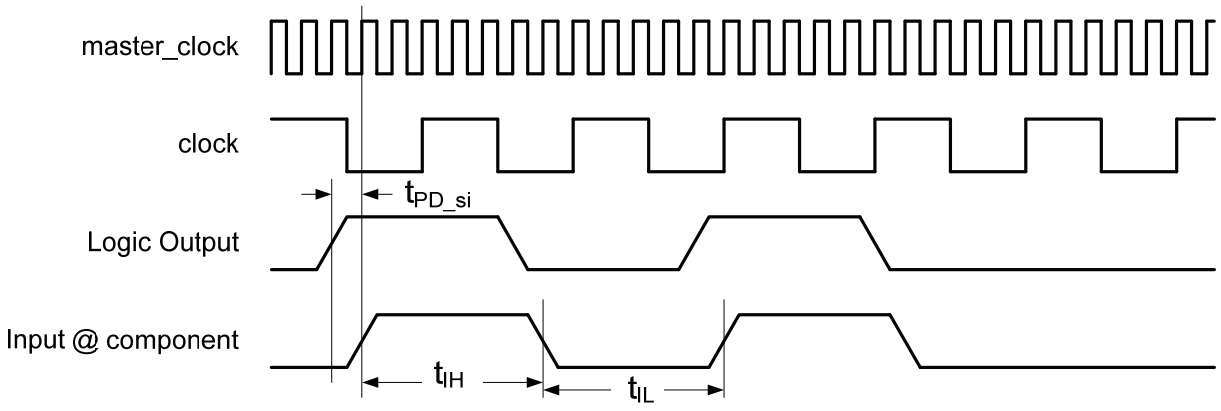
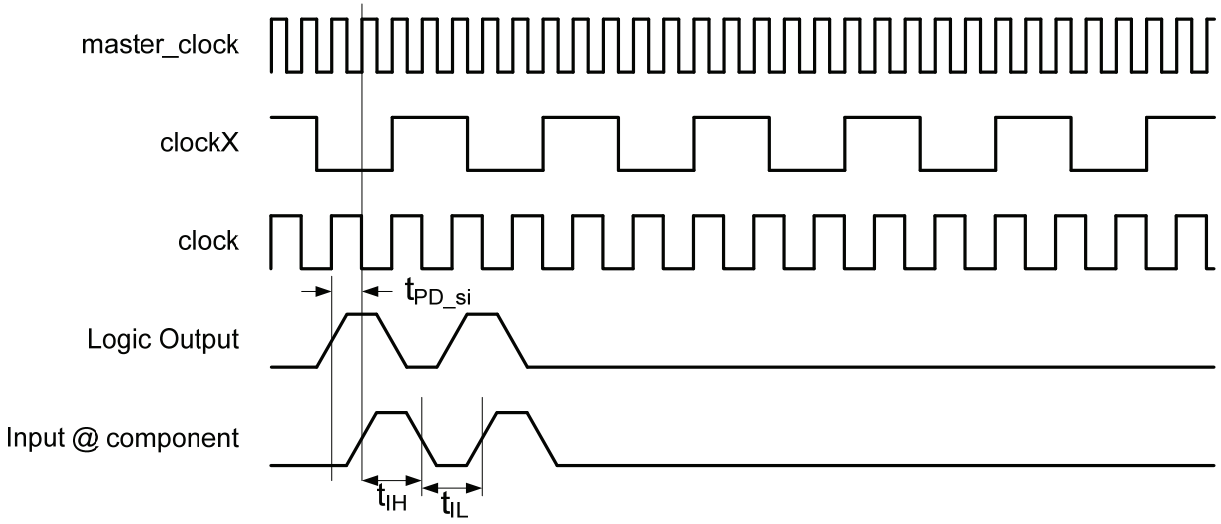
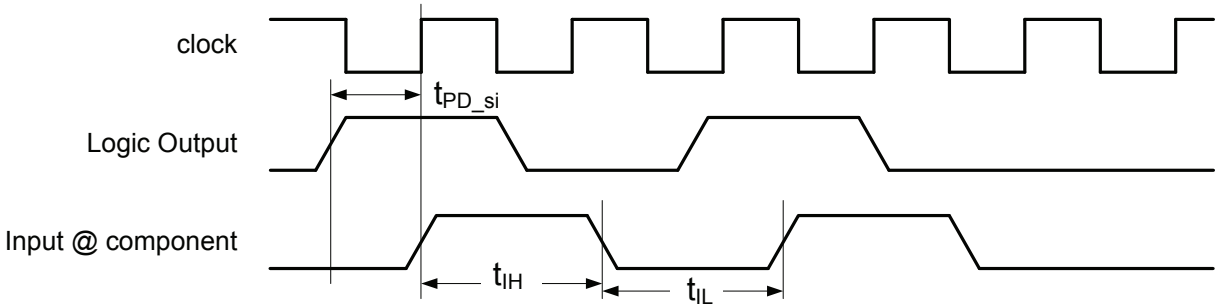


图 20. 输入配置 3：同步器时钟频率 < 组件时钟频率



与图 17 中的方法几乎相同，所有时钟都派生自 master_clock。STA 在此配置中指明了 t_{PD_si} 对 master_clock 的限制要在一个 master_clock 时钟周期里。如果此路径延迟太长，则 master_clock 设置时间出现冲突。您必须更改系统的同步时钟，或者以较慢的频率运行 master_clock。

图 21. 仅输入配置 4：同步器时钟 = 组件时钟



在本节的所有上述图形中，最关键参数是用于已知的实现方式下 f_{CLOCK} 和 t_{PD_IE}。t_{PD_IE} 由 t_{PD_ps} 和 t_{sync}（仅针对配置 1 和 2）、t_{PD_si} 和 t_{I_Clk} 定义。重要的是记住 t_{PD_si} 定义最大组件时钟频率。t_{I_Clk} 不源自 STA 结果，但是用于表示何时寄存 t_{PD_IE}。这是同步器与组件时钟之间的路由之后的余留。

t_{PD_ps} 和 t_{PD_si} 包括在 STA 结果中。

要查找 t_{PD_ps}，请查看 *_timing.html* 文件中定义的输入设置时间。此输入的终端可以大于 1，因此您需要计算这些路径的最大值。

-Setup times

-Setup times to clock BUS_CLK

Start	Register	Clock	Delay (ns)
input1(0):iocell.pad_in	input1(0):iocell.ind	BUS_CLK	16.500

t_{PD_si} 是在“寄存器至寄存器”时间中定义的。您需要知道使用 *_timing.html* 文件的网络的名称。此路径的终端可以大于 1，因此您需要计算这些路径的最大值。

-Register-to-register times

-Destination clock clock

Destination clock clock (Actual freq: 24.000 MHz)

+Source clock clock

-Source clock clock_1

Source clock clock_1 (Actual freq: 24.000 MHz)
Affected clock: BUS_CLK (Actual freq: 24.000 MHz)

Start	End	Period (ns)	Max Freq	Frequency	Violation
\Sync_1:genblk1[0]:INST\:synccell.syncq	\PWM_1:PWMUDB:runmode_enable\:macrocell.mc_d	7.843	127.508 MHz	24.000 MHz	

输出路径延迟

当要表现输出路径延迟的特性时，必须考虑输出的去向，以了解在 STA 结果中何处可以找到数据。对于此组件，所有输出同步到组件时钟。输出可以是下列两类之一。输出到器件中的另一个组件，或输出到器件外的引脚。在第一种情况下，必须查看已显示的“逻辑至输入”说明的“寄存器至寄存器”时间（源时钟是组件时钟）。对于第二种情况，可以在*_timing.html* STA 结果中查看“时钟至输出”时间。

组件更改

本节介绍组件与以前版本相比的主要更改。

版本	更改说明	更改原因/影响
2.20	针对 UDB 实现的定时器仿真模型更改	用于解决当使用硬件启用时，TC 输出在某些条件下可能遗漏的情况。
	记录中断信号不可用于 PSoC 5 FF 实现的定时器	移除了此功能，因为芯片无法支持它
	更新了自定义程序以使 Cancel（取消）按钮始终可用	在某些错误情况下，Cancel（取消）按钮不可用

版本	更改说明	更改原因/影响
	扩展数据手册更新	添加了基于不同实现方式（UDB、PSoC 3 FF、PSoC 5 FF）定时器的差别，以前未充分介绍这些差异。具体而言，请参见功能描述的“配置”一节中提供的波形。
2.10	仿真模型更新和自定义程序相关更新	用于解决与触发器逻辑有关的细节问题和 GUI 相关问题
	当 Capture Mode（捕获模式）设置为 None（无）时，会禁用“Interrupt on Capture（捕获时中断）”	“Interrupt on Capture（捕获时中断）”复选框选项即使在 Capture Mode（捕获模式）设置为“None（无）”时也可用，但本不应该为可用状态
2.0	同步的输入	固定功能实现定时器的所有输入是在输入模块中同步的。
	Timer_GetInterruptSource() 函数转换为宏	Timer_GetInterruptSource() 函数是与 Timer_ReadStatusRegister() 函数完全一致的实现。为了节省代码空间，该函数转换为 Timer_ReadStatusRegister() 函数的宏替换。
	输出寄存到组件时钟上	为了避免组件输出中出现故障脉冲，需要将所有输出同步。如果可能，此同步在数据路径内部完成，以避免过多使用资源。
	实现了写入辅助控制寄存器的关键区域。	当写入辅助控制寄存器时使用 CyEnterCriticalSection 和 CyExitCriticalSections 函数，以便它不会被任何其他进程线程修改。
	纠正了在使用 SetCaptureMode() API 设置捕获模式时的错误屏蔽。	用于屏蔽设置捕获模式的屏蔽值有错误时。
	向数据手册中添加了特性数据	
	对数据表进行了少量编辑和更新	

© 赛普拉斯半导体公司，2012。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品的内嵌电路之外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC® 是赛普拉斯半导体公司的注册商标，PSoC Creator™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码（软件和/或固件）均归赛普拉斯半导体公司（赛普拉斯）所有，并受全球专利法规（美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和/或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定的用途之外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对此材料提供任何类型的明示或暗示保证，包括（但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不做出通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用的赛普拉斯软件许可协议限制。