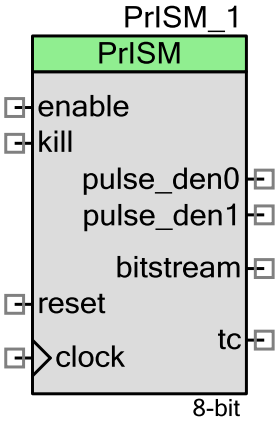
精确照明信号调制 (PrISM)

2.0

特性



- 2 到 32 位分辨率的可编程无闪烁调光
- 两个脉冲密度输出
- 可编程的输出信号密度
- 串行输出位流
- 连续运行模式
- 用户可配置序列启动值
- 为所有序列长度提供的标准或自定义多项式
- 非同步输入将禁用密度输出并强制它们处于低电平
- 使能输入提供与其他组件进行同步操作
- 复位输入允许重新启动序列开始值以实现与其他组件的同步操作
- 适用于 8 位、16 位、24 位和 32 位序列长度的终端计数输出。

概述

精确照明信号调制 (PrISM) 组件使用线性反馈移位寄存器 (LFSR) 生成伪随机序列。此序列输出伪随机位流以及最多两个用户可调伪随机脉冲密度。这些脉冲密度的范围在 0 到 100% 之间。

LFSR 采用 Galois 形式（有时称为模形式），使用提供的最大长度代码。PrISM 组件启动后，只要使能输入处于高电平，此组件将持续运行。PrISM 伪随机数发生器可使用任意有效值（0 除外）进行启动。

何时使用 PrISM

PrISM 组件提供了调制技术，可大大降低低频闪烁和电磁辐射干扰 (EMI)，这些是高亮度 LED 设计的常见问题。PrISM 也可用于其他需要这种优势的应用，例如电机控制和供电电源。

输入/输出连接

本节介绍 PrISM 的各种输入和输出连接。I/O 列表中的星号 (*) 表示，在 I/O 说明中列出的情况下，该 I/O 可能不可见。

时钟 – 输入

时钟输入定义用于计算伪随机序列的信号。

复位 – 输入

复位输入用于将伪随机序列复位为高电平下的开始值。此输入仅对于已启动的组件有效，提供与其他组件的同步操作。

非同步停止 – 输入

高电平有效非同步停止输入用于禁用 PrISM 脉冲密度输出并将它们设置为 0 直至非同步停止释放为低电平。

使能 – 输入

PrISM 组件启动后，只要使能输入处于高电平，复位输入为低电平，此组件将继续运行。此输入提供与其他组件的同步操作。

pulse_den0/pulse_den1 – 输出

有两个脉冲密度输出可用；这两个输出都是派生自同一个伪随机序列。每个输出都是通过将需要的脉冲密度值与当前的伪随机序列数字进行比较而生成的。如果脉冲密度类型配置为 **Less Than or Equal**（小于或等于），则输出将处于高电平，而伪随机序列数字将小于或等于脉冲密度值。另一个选项是将脉冲密度类型设置为 **Greater Than or Equal**（大于或等于），则输出将处于高电平，而伪随机序列数字将大于或等于脉冲密度值。

位流 – 输出

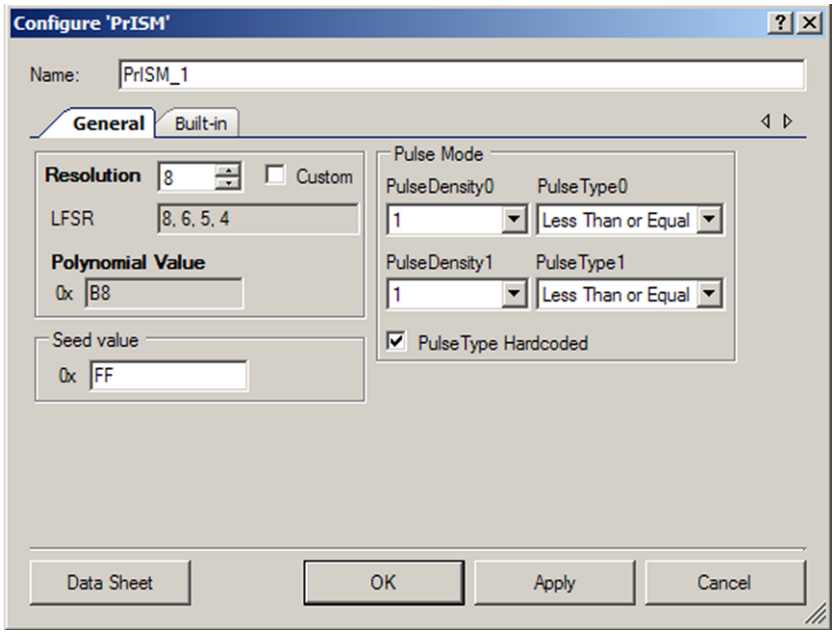
位流输出用于连续输出 LFSR 的 LSb。

tc – 输出 *

终端计数输出适用于 8 位、16 位、24 位 和 32 位长度 PrISM 组件。在每个时钟周期内，每次伪随机序列数字等于 0xFF（8 位）、0xFFFF（16 位）、0xFFFFFF（24 位）或 0xFFFFFFFF（32 位）时，终端输出技术输出都会处于高电平，每个时钟伪随机序列数字发生器的每个周期都会出现一次这种情况。

组件参数

将一个 PrISM 组件拖放到设计上，并双击以打开 **Configure**（配置）对话框。



PrISM 组件包含下列参数：

Resolution（分辨率）

此参数用于定义 PrISM 最大代码长度（周期）。最大代码长度为（$2^{\text{分辨率}} - 1$）。可能值包括 2 - 32 位。最大长度代码用于设置伪随机序列数字发生器的长度，因此即要生成的序列的长度。序列越长，脉冲密度分辨率将越高，同时辐射的 EMI 就越低。下表中列出的最大长度代码以 Galois 形式提供，在 PSoC 3 UDB ALU 中使用这些代码之前不需要进行转换。

Resolution（分辨率）	LFSR		Resolution（分辨率）	LFSR		Resolution（分辨率）	LFSR
2	2, 1		13	13, 12, 10, 9		24	24, 23, 21, 20
3	3, 2		14	14, 13, 11, 9		25	25, 24, 23, 22
4	4, 3		15	15, 14, 13, 11		26	26, 25, 24, 20
5	5, 4, 3, 2		16	16, 14, 13, 11		27	27, 26, 25, 22
6	6, 5, 3, 2		17	17, 16, 15, 14		28	28, 27, 24, 22
7	7, 6, 5, 4		18	18, 17, 16, 13		29	29, 28, 27, 25
8	8, 6, 5, 4		19	19, 18, 17, 14		30	30, 29, 26, 24
9	9, 8, 6, 5		20	20, 19, 16, 14		31	31, 30, 29, 28
10	10, 9, 7, 6		21	21, 20, 19, 16		32	32, 30, 26, 25

Resolution（分辨率）	LFSR		Resolution（分辨率）	LFSR		Resolution（分辨率）	LFSR
11	11, 10, 9, 7		22	22, 19, 18, 17			
12	12, 11, 8, 6		23	23, 22, 20, 18			

要手动设置 LFSR 系数：

定义 **Resolution**（分辨率）。

选择 **Custom**（自定义）复选框。

在 LFSR 文本框中输入用逗号分隔的系数并按 [**Enter**（输入）]。将自动重新计算 **Polynomial Value**（多项式值）。

Polynomial Value（多项式值）按十六进制格式显示。

注意 LFSR 系数值不能大于 **Resolution**（分辨率）值。

Polynomial Value（多项式值）

此参数按十六进制格式显示。根据选择的 **Resolution**（分辨率）选择正确的多项式。可选择性地指定自定义多项式。

种子值

默认情况下，此参数设置为最大的可能值（$2^{\text{分辨率}} - 1$）。此值可更改为任意值（0 除外）。**Seed value**（种子值）按十六进制格式显示。

注意更改 **Resolution**（分辨率）会将 **Seed value**（种子值）设置为默认值。

脉冲模式

这些参数值是从组合框中选定的。可用值范围为 1 到 $2^{\text{分辨率}} - 1$，阶为 $2^{\text{分辨率}}$。脉冲比较类型可设置为 **Less Than or Equal**（小于或等于）或 **Greater Than or Equal**（大于或等于）。

PulseType Hardcoded（硬编码 PulseType）

PulseType Hardcoded（硬编码 PulseType）参数在启用后可存储资源（控制寄存器），但是会令无法使用 PrISM_SetPulse0Mode() 或 PrISM_SetPulse1Mode() APIs 更改脉冲类型。

如果启用了此函数，也无法使用 PrISM_Stop() 函数。在这种情况下，要停止 PrISM，使用“使能”输入。

本地参数（供 API 使用）

这些参数用于 API 中，不在 **Configure**（配置）对话框中显示。

- **PolyValue(uint32) –** 包含使用十六进制格式的多项式值。默认值为 0xB8h (LFSR= [8,6,5,4])。
- **Density0(uint32) –** 包含使用十六进制格式的 density0 值。
- **Density1(uint32) –** 包含使用十六进制格式的 density1 值。
- **CompareType0(CompareType) –** 包含 Density0 的 **Pulse Type**（脉冲类型），其可以为 **Less Than or Equal**（小于或等于）或 **Greater Than or Equal**（大于或等于）。
- **CompareType1(CompareType) –** 包含 Density1 的 **Pulse Type**（脉冲类型），其可以为 **Less Than or Equal**（小于或等于）或 **Greater Than or Equal**（大于或等于）。

时钟选择

此组件中没有内部时钟。您必须附加时钟源。最大频率输入为 67 MHz。

放置

PrISM 组件放置于整个 UDB 阵列中，并且所有放置信息通过 *cyfitter.h* 文件提供给 API。

资源

资源	资源类型				API Memory（API 存储器）（字节）		Pins（引脚）（每个外部 I/O）
	数据路径单元	PLD	状态单元	Control/ Count7 单元	Flash（闪存）	RAM	
8 位	1	3	0	1	423	6	8
8 位 *	1	3	0	0	423	6	8
16 位	2	3	0	1	543	13	8
24 位	3	3	0	1	569	23	8
32 位	4	3	0	1	569	23	8

* 已启用参数 **PulseType Hardcoded**（硬编码 PulseType）。

应用程序编程接口

应用程序编程接口 (API) 子程序允许您使用软件配置组件。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“PrISM_1”分配给设计中的第一个组件实例。您可以将该实例重命名为符合标识符语法规则的任意唯一值。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。为增加可读性，下表中使用了实例名称“PrISM”。

函数	说明
PrISM_Start()	此启动函数用于设置定制器提供的多项式、种子和脉冲密度寄存器。
PrISM_Stop()	停止 PrISM 计算。
PrISM_SetPulse0Mode()	设置 Density0 的脉冲密度类型。
PrISM_SetPulse1Mode()	设置 Density1 的脉冲密度类型。
PrISM_ReadSeed()	读取 PrISM 种子寄存器。
PrISM_WriteSeed()	使用开始值写入 PrISM 种子寄存器。
PrISM_ReadPolynomial()	读取 PrISM 多项式寄存器。
PrISM_WritePolynomial()	使用开始值写入 PrISM 多项式寄存器。
PrISM_ReadPulse0()	PrISM 脉冲 Density0 值寄存器。
PrISM_WritePulse0()	使用新的脉冲密度值写入 PrISM 脉冲 Density0 值。
PrISM_ReadPulse1()	读取 PrISM 脉冲 Density1 值寄存器。
PrISM_WritePulse1()	使用新的脉冲密度值写入 PrISM 脉冲 Density1 值。
PrISM_Sleep()	停止并保存用户配置。
PrISM_Wakeup()	恢复并使能用户配置
PrISM_Init()	初始化随自定义程序提供的默认配置。
PrISM_Enable()	使能 PrISM 模块操作。
PrISM_SaveConfig()	保存当前用户配置。
PrISM_RestoreConfig()	恢复当前用户配置。

全局变量

变量	说明
PrISM_initVar	说明 PrISM 是否已初始化。该变量初始化为 0，并在第一次调用 PrISM_Start() 时设置为 1。这样，第一次调用 PrISM_Start() 子程序后，组件不用重新初始化即可重启。 如果需要重新初始化此组件，则在 PrISM_Start() 或 PrISM_Enable() 函数之前可调用 PrISM_Init() 函数。

void PrISM_Start(void)

说明： 这是开始执行组件操作的首选方法。PrISM_Start() 用于设置 initVar 变量，调用 PrISM_Init() 函数并调用 PrISM_Enable() 函数。此启动函数用于设置定制器提供的多项式、种子和脉冲密度寄存器。PrISM 计算在输入时钟的上升沿上开始执行。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void PrISM_Stop(void)

说明： 停止 PrISM 计算。输出保持固定。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： 只有在禁用 **PulseType Hardcoded**（硬编码 PulseType）参数时才有效。

void PrISM_SetPulse0Mode(uint8 pulse0Type)

说明： 设置 Density0 的脉冲密度类型。小于或等于 (<=) 或大于或等于 (>=)。

参数： uint8 pulse0Type：选择脉冲密度类型

参数值	说明
PrISM_LESSTHAN_OR_EQUAL	当伪随机数字小于或等于 PulseDensity0 寄存器值时，pulse_den0 处于高电平。
PrISM_GREATERTHAN_OR_EQUAL	当伪随机数字大于或等于 PulseDensity0 寄存器值时，pulse_den0 处于高电平。

Return Value（返回值）： None（无）

Side Effects（副作用）： 只有在禁用 **PulseType Hardcoded**（硬编码 PulseType）参数时才有效。

void PrISM_SetPulse1Mode(uint8 pulse1Type)

说明： 设置 Density1 的脉冲密度类型。小于或等于 (<=) 或大于或等于 (>=)。

参数： uint8 pulse1Type：选择脉冲密度类型

参数值	说明
PrISM_LESSTHAN_OR_EQUAL	当伪随机数字小于或等于 PulseDensity1 寄存器值时，pulse_den1 处于高电平。
PrISM_GREATERTHAN_OR_EQUAL	当伪随机数字大于或等于 PulseDensity1 寄存器值时，pulse_den1 处于高电平。

Return Value（返回值）： None（无）

Side Effects（副作用）： 只有在禁用 **PulseType Hardcoded**（硬编码 PulseType）参数时才有效。

uint8/16/32 PrISM_ReadSeed(void)

说明： 读取 PrISM 种子寄存器。

参数： None（无）

Return Value（返回值）： uint8/16/32：设置寄存器值

Side Effects（副作用）： None（无）

void PrISM_WriteSeed(uint8/16/32 seed)

说明： 使用开始值写入 PrISM 种子寄存器。

参数： uint8/16/32) 种子：设置寄存器值

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

uint8/16/32 PrISM_ReadPolynomial(void)

说明： 读取 PrISM 多项式。

参数： None（无）

Return Value（返回值）： uint8/16/32：多项式值

Side Effects（副作用）： None（无）

void PrISM_WritePolynomial(uint8/16/32 polynomial)

说明： 写入 PrISM 多项式。

参数： uint8/16/32 多项式：多项式寄存器值

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

uint8/16/32 PrISM_ReadPulse0(void)

说明： 读取 PrISM 脉冲 Density0 值寄存器。

参数： None（无）

Return Value（返回值）： uint8/16/32：PulseDensity0 寄存器值

Side Effects（副作用）： None（无）

void PrISM_WritePulse0(uint8/16/32 pulseDensity0)

说明： 使用新的脉冲密度值写入 PrISM 脉冲 Density0 值。

参数： (unit8/16/32) pulseDensity0：脉冲密度值。

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

uint8/16/32 PrISM_ReadPulse1(void)

说明： 读取 PrISM 脉冲 Density1 值寄存器。

参数： None（无）

Return Value（返回值）： uint8/16/32：PulseDensity1 寄存器值

Side Effects（副作用）： None（无）

void PrISM_WritePulse1(uint8/16/32 pulseDensity1)

说明： 使用新的脉冲密度值写入 PrISM 脉冲 Density1 值。

参数： uint8/16/32 pulseDensity1：脉冲密度值

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void PrISM_Sleep(void)

说明： 这是准备组件进入睡眠的首选 API。PrISM_Sleep() API 保存当前器件的状态。然后，它将调用 PrISM_Stop() 函数，并调用 PrISM_SaveConfig() 以保存硬件配置。

在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 PrISM_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： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void PrISM_Wakeup(void)

说明： 此函数是将器件恢复到调用 PrISM_Sleep() 时状态的首选 API。PrISM_Wakeup() 函数调用 PrISM_RestoreConfig() 函数以恢复配置。如果组件在系统调用 PrISM_Sleep() 函数前已启用，则 PrISM_Wakeup() 函数也将重新启用组件。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： 调用 PrISM_Wakeup() 函数前未调用 PrISM_Sleep() 或 PrISM_SaveConfig() 函数可能会产生意外行为。

void PrISM_Init(void)

说明： 根据自定义程序“配置”对话框设置来初始化或恢复组件。无需调用 PrISM_Init()，因为 PrISM_Start() API 会调用该函数，这是开始组件操作的首选方法。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： 所有寄存器将设置为自定义程序“配置”对话框中的值。

void PrISM_Enable(void)

说明： 激活硬件并开始执行组件操作。您无需调用 PrISM_Enable()，因为 PrISM_Start() 会调用此函数，该函数是开始执行组件操作的首选方法。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void PrISM_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 修改的当前组件参数值。该函数由 PrISM_Sleep() 函数调用。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void PrISM_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 PrISM_Sleep() 函数之前的值。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： 调用该函数前未调用 PrISM_Sleep() 或 PrISM_SaveConfig() 函数可能会产生意外行为。

固件源代码示例

PSoC Creator 在“查找示例项目”对话框中提供了大量包括原理图和代码示例的示例项目。要获取组件特定的示例，请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 Start Page（开始页）或 **File**（文件）菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（滤波器选项）可缩小可选项目的列表。

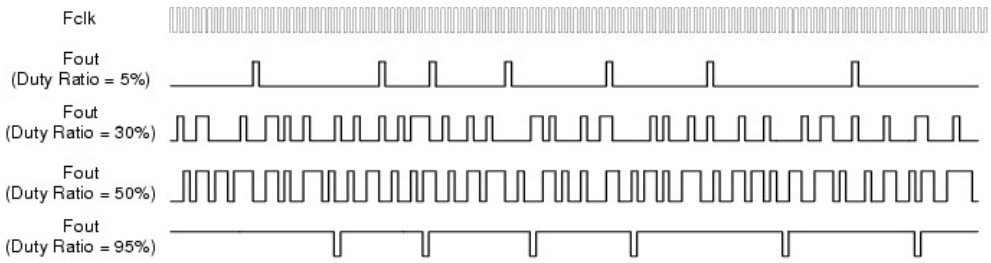
有关更多信息，请参见 PSoC Creator 帮助中的“Find Example Project（查找示例项目）”主题。

功能描述

PrISM 组件启动后，只要“使能”输入处于高电平，此组件将持续运行。PrISM 伪随机数字发生器可以任意有效值（0 除外）进行启动。从而允许多个 PrISM 组件相互之间不同步运行，从而进一步减少 EMI。“复位”输入用于将伪随机数字复位为开始值。高电平有效非同步停止输入用于禁用 PrISM 脉冲密度输出并将它们设置为 0 直至非同步停止输入释放为低电平。“位流”输出用于连续输出 LFSR 的 LSb。

有两个脉冲密度输出可用；这两个输出都是派生自同一个伪随机序列。每个输出都是通过将需要的脉冲密度值与当前的伪随机序列数字进行比较而生成的。

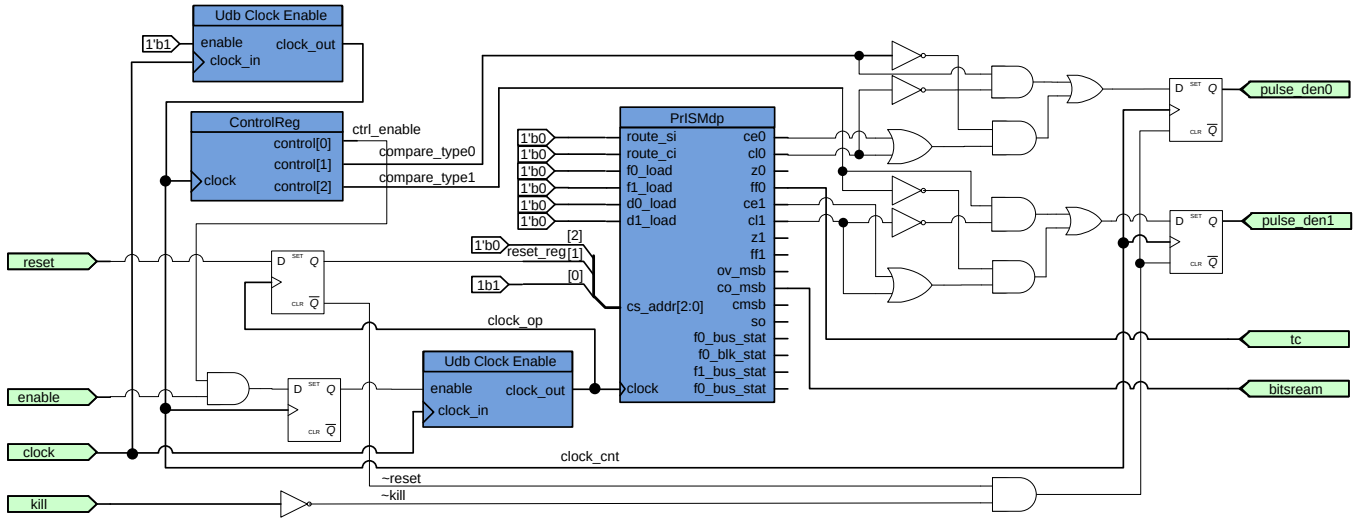
下表显示了基于多个脉冲密度比率的 PrISM 输出。



框图和配置

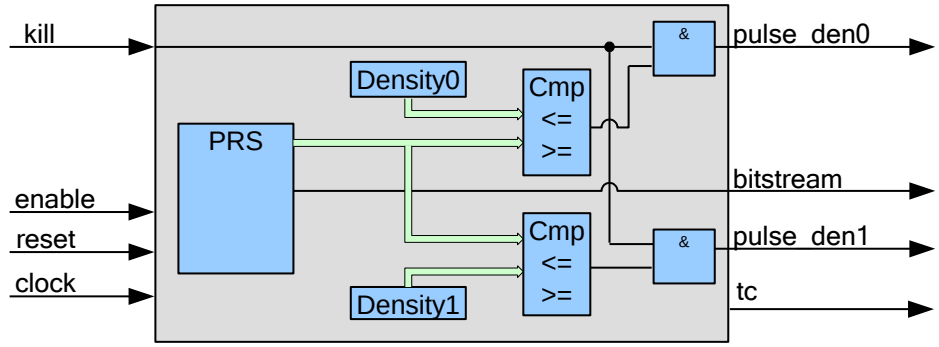
PrISM 仅作为 UDB 配置提供。上面所描述的 API 和此处所描述的寄存器用于定义 PrISM 的整体实现。

下面的框图中描述了实现。



分级架构

2 位到 32 位的硬件 PrISM 组件将伪随机计数器的输出与单个密度值进行比较。当计数值小于（或大于）或等于密度值寄存器中的值时，比较器输出设置。



寄存器

PrISM_CONTROL

位	7	6	5	4	3	2	1	0
值	保留					比较 type1	比较 type0	ctrl 启用

- ctrl 启用：此位启用前面章节中说明的所有信号的生成功能。此值可 PrISM_Start() 和 PrISM_Stop() 函数进行更改。
- 比较 type0：此位执行 pulse_den0 输出的比较类型。此位的值由组件“Configure”（配置）对话框中做出的脉冲比较类型选项确定。同时，此值可由 PrISM_SetPulse0Mode() 函数进行更改。
- 比较 type1：此位执行 pulse_den1 输出的比较类型。此位的值由组件“Configure”（配置）对话框中做出的脉冲比较类型选项确定。同时，此值可由 PrISM_SetPulse1Mode() 函数进行更改。

如果选择了 **PulseType Hardcoded**（硬编码 PulseType）选项，则不使用控制寄存器。

PrISM_SEED

位	7	6	5	4	3	2	1	0
值	种子							

- 种子：包含计算计数时的初始种子值和 PRS 余值。此寄存器的值由组件“Configure”（配置）对话框中的 **Seed value**（种子值）确定。同时，此值可由 PrISM_WriteSeed() 函数进行更改，且可由 PrISM_ReadSeed() 函数进行读取。

PrISM_SEED_COPY

位	7	6	5	4	3	2	1	0
值	Seed_Copy							

- Seed_Copy：包含开始的种子值，当“复位”输入有效时，此值将自动加载 PrISM_SEED 寄存器。此寄存器的值由组件“Configure”（配置）对话框中的**种子值**确定，而且如果调用了 PrISM_WriteSeed() 函数将自动进行更新。

PrISM_POLYNOM

位	7	6	5	4	3	2	1	0
值	多项式							

- 多项式：根据选择的分辨率选择正确的多项式。此值可由 PrISM_WritePolynomial() 函数进行更改，且可由 PrISM_ReadPolynomial() 函数进行读取。

PrISM_DENSITY0

位	7	6	5	4	3	2	1	0
值	脉冲 density0							

- 脉冲 density0 确定 PrISM pulse_den0 输出的值。此寄存器的值由“Configure”（配置）对话框中的 **PulseDensity0** 参数确定。此值可由 PrISM_WritePulse0() 函数进行更改。

PrISM_DENSITY1

位	7	6	5	4	3	2	1	0
值	脉冲 density1							

- 脉冲 density1 确定 PrISM pulse_den1 输出的值。此寄存器的值由“Configure”（配置）对话框中的 **PulseDensity1** 参数确定。此值可由 PrISM_WritePulse1() 函数进行更改。

参考

另请参见 PRS 组件数据手册。

直流和交流电气特性

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			66	MHz
		16 位			46	MHz
		24 位			42	MHz
		32 位			38	MHz
t_{clockH}	输入时钟高电平时间 [1]	不可用		0.5		$1/f_{CLOCK}$
t_{clockL}	输入时钟低电平时间[1]	不可用		0.5		$1/f_{CLOCK}$
输入						
t_{PD_ps}	输入路径延迟，要同步的引脚 [2]	1			STA [3]	ns
t_{PD_ps}	输入路径延迟，要同步的引脚 [4]	2			8.5	ns
t_{PD_si}	输入路径延迟的同步输出（路由）	1,2,3,4			STA [3]	ns
t_{I_clk}	clockX 与时钟的对齐	1,2,3,4	0		1	t_{CY_clock}
t_{PD_IE}	组件时钟的输入路径延迟（边沿敏感输入）	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（边沿敏感输入）	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

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

[2] 可以在后面所述的“静态时序结果”中找到 t_{PD_ps}。此处列出的数字是基于许多输入的 STA 分析的额定值。

[3] 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。

[4] 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。

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[1]	单位
f_{CLOCK}	组件时钟频率	8 位			40	MHz
		16 位			23	MHz
		24 位			21	MHz
		32 位			19	MHz
t_{clockH}	输入时钟高电平时间 [2]	不可用		0.5		$1/f_{CLOCK}$
t_{clockL}	输入时钟低电平时间 [2]	不可用		0.5		$1/f_{CLOCK}$
输入						
t_{PD_ps}	输入路径延迟，要同步的引脚 [3]	1			STA [4]	ns
t_{PD_ps}	输入路径延迟，要同步的引脚 [5]	2			8.5	ns
t_{PD_si}	输入路径延迟的同步输出（路由）	1,2,3,4			STA [4]	ns
t_{l_clk}	clockX 与时钟的对齐	1,2,3,4	0		1	t_{CY_clock}
t_{PD_IE}	组件时钟的输入路径延迟（边沿敏感输入）	1,2	t_{PD_ps} + t_{SYNC} + t_{PD_si}		t_{PD_ps} + t_{SYNC} + t_{PD_si} + t_{l_clk}	ns
t_{PD_IE}	组件时钟的输入路径延迟（边沿敏感输入）	3,4	t_{SYNC} + t_{PD_si}		t_{SYNC} + t_{PD_si} + t_{l_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

[1] “所有路由”的最大值的计算方法是：<额定值>/2，然后取整到最近的整数。如果此组件在此频率或此频率之下运行，此值可以让您不必担心合适的时间问题。

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

[3] 可以在后面所述的“静态时序结果”中找到 t_{PD_ps}。此处列出的数字是基于许多输入的 STA 分析的额定值。

[4] 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。

[5] 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。

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命名外部时钟的时钟汇总中的时序结果中。下图演示了 *_timing.html* 中的时钟限制示例：

-Clock Summary

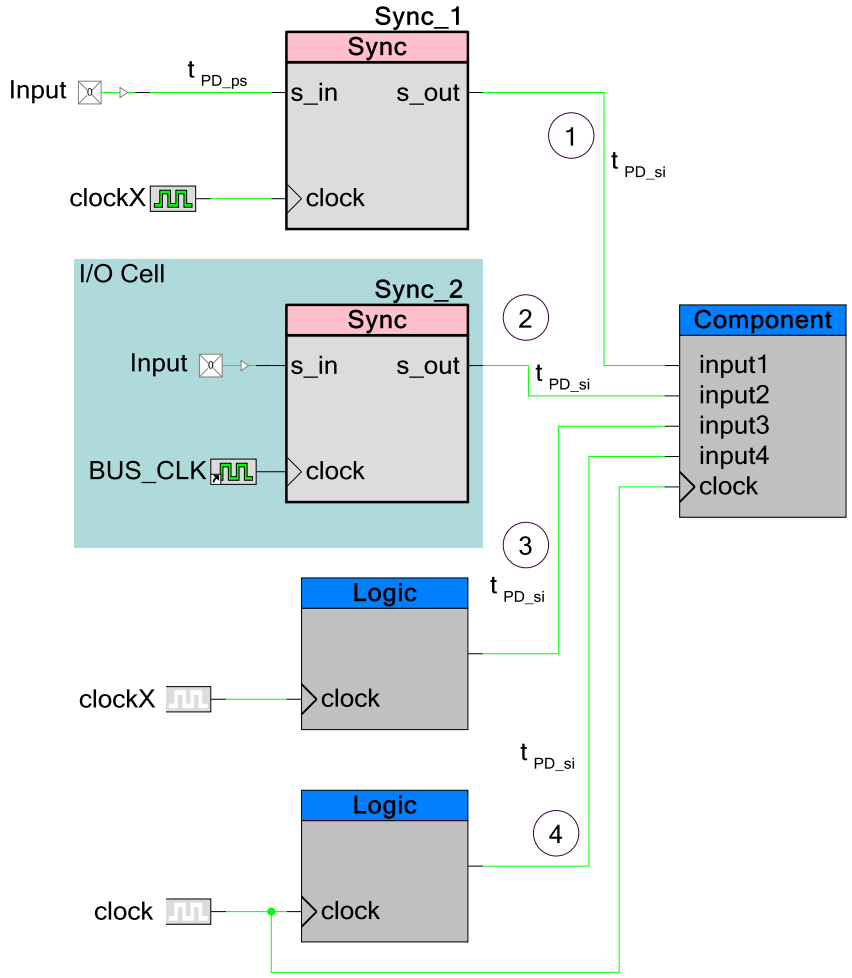
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

输入路径延迟和脉冲宽度

当表现输入功能的特征时，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 1 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哪个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 1. 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 6
1	时钟	master_clock	图 4
1	时钟	clockX = 时钟 [1]	图 2
1	时钟	clockX > 时钟	图 3
1	时钟	clockX < 时钟	图 5
2	master_clock	master_clock	图 6
2	时钟	master_clock	图 4
3	master_clock	master_clock	图 11

[1] 时钟频率相等，但是不保证上升沿的对齐。

配置	组件时钟	同步器时钟（频率）	图形
3	时钟	master_clock	图 9
3	时钟	clockX = 时钟 [1]	图 7
3	时钟	clockX > 时钟	图 8
3	时钟	clockX < 时钟	图 10
4	master_clock	master_clock	图 11
4	时钟	时钟	图 7

1. 输入由器件引脚驱动，并在内部与“同步”组件同步。此组件的时钟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。它也可以等于 master_clock。这会生成如图 2、图 3、图 5 和图 6 所示的特性参数。

2. 输入由器件引脚驱动，并使用 master_clock 在引脚同步。

   当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（从未慢于组件时钟）。这会生成如图 3 和图 6 所示的特性参数。

图 2. 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）

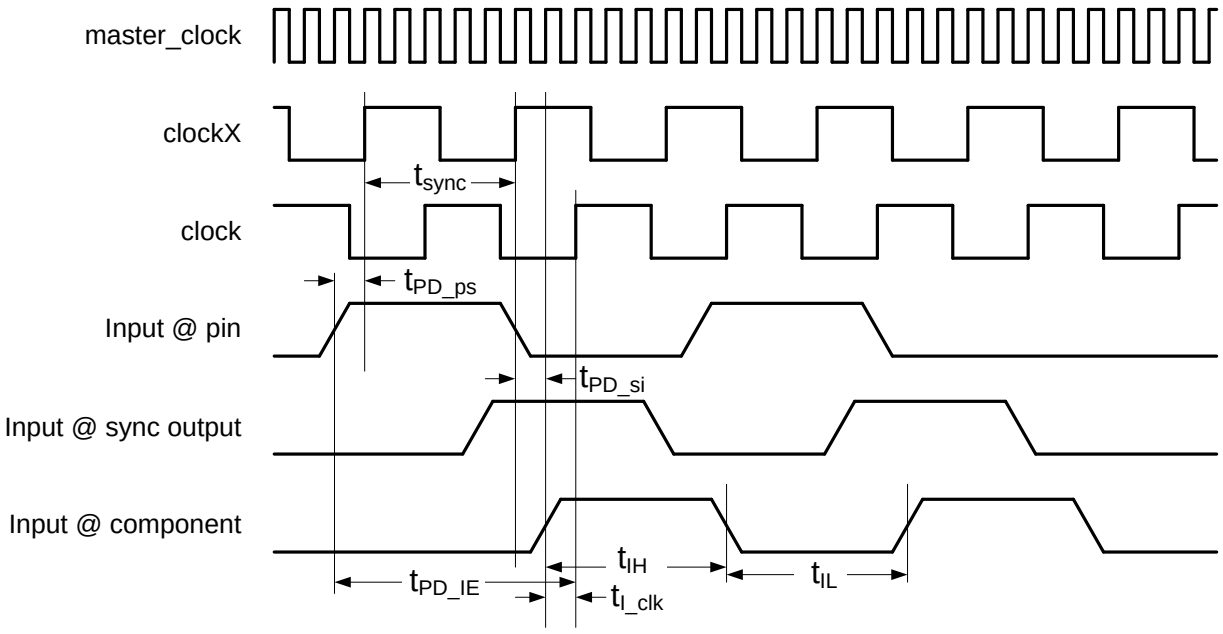


图 3. 输入配置 1 和 2：同步 时钟频率 > 组件时钟频率

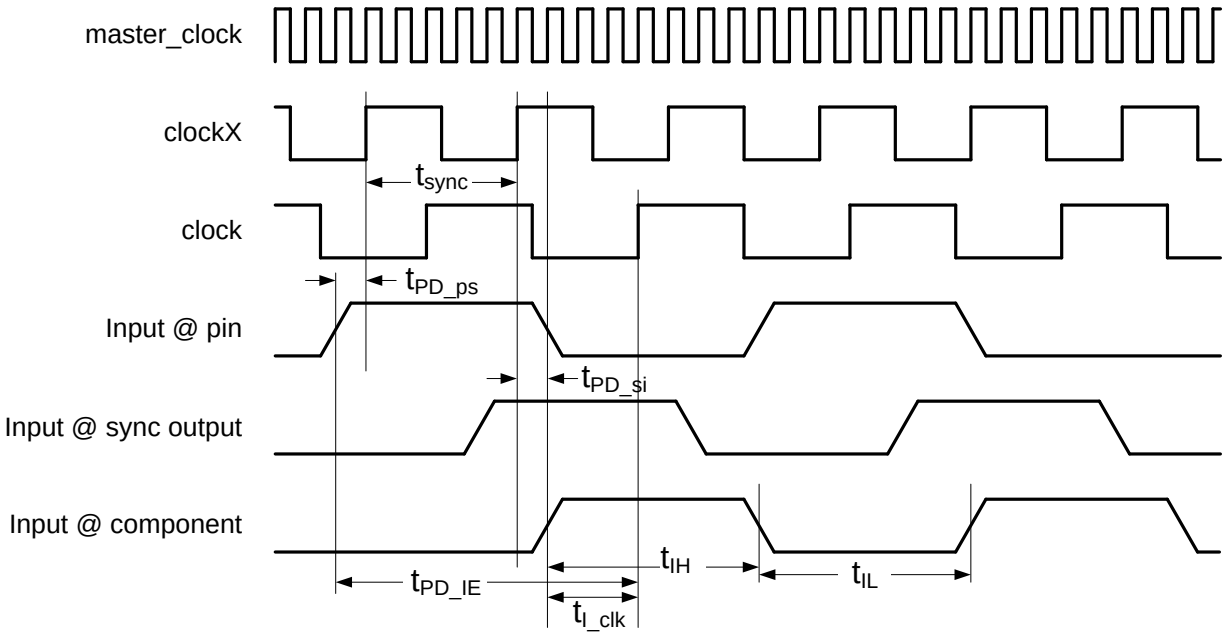


图 4. 输入配置 1 和 2：[同步 时钟频率 == master_clock] > 组件时钟频率

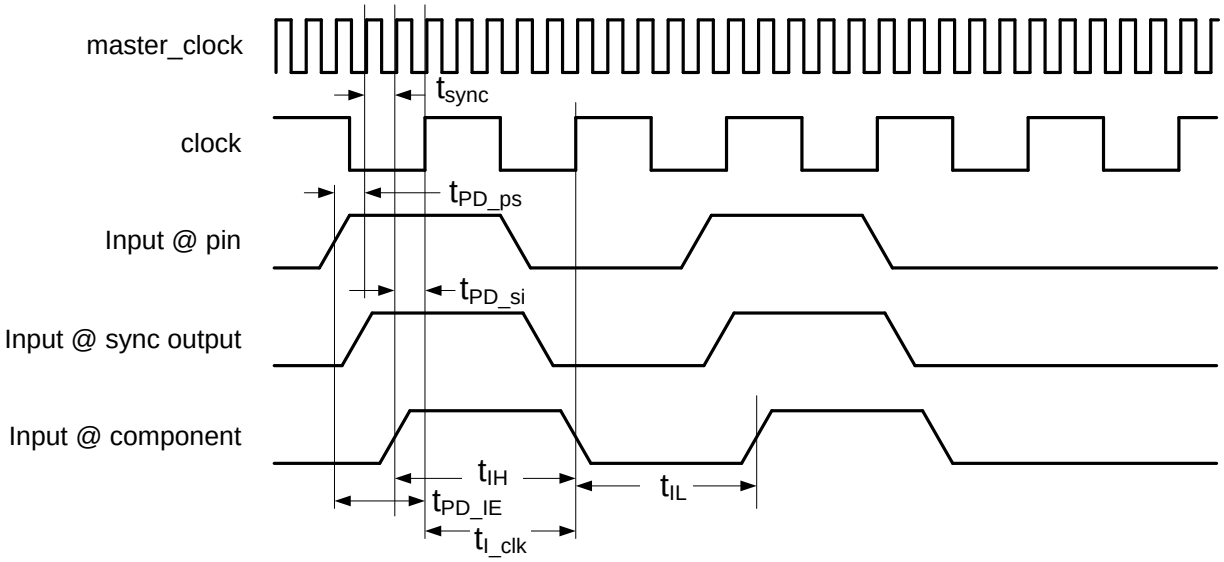


图 5. 输入配置 1：同步 时钟频率 < 组件时钟频率

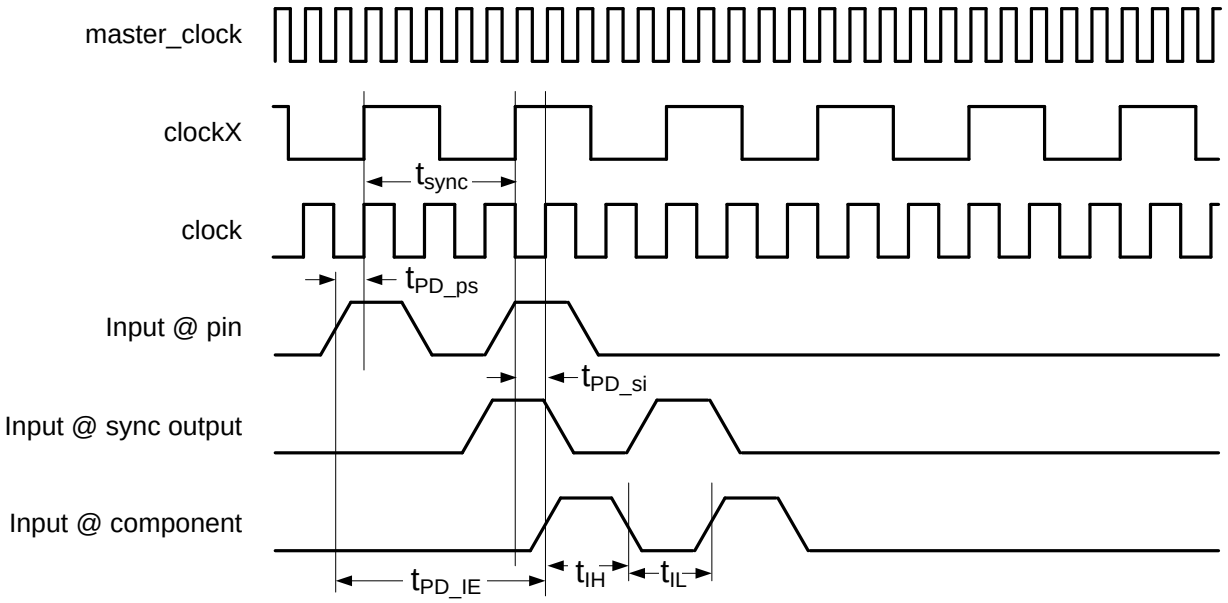
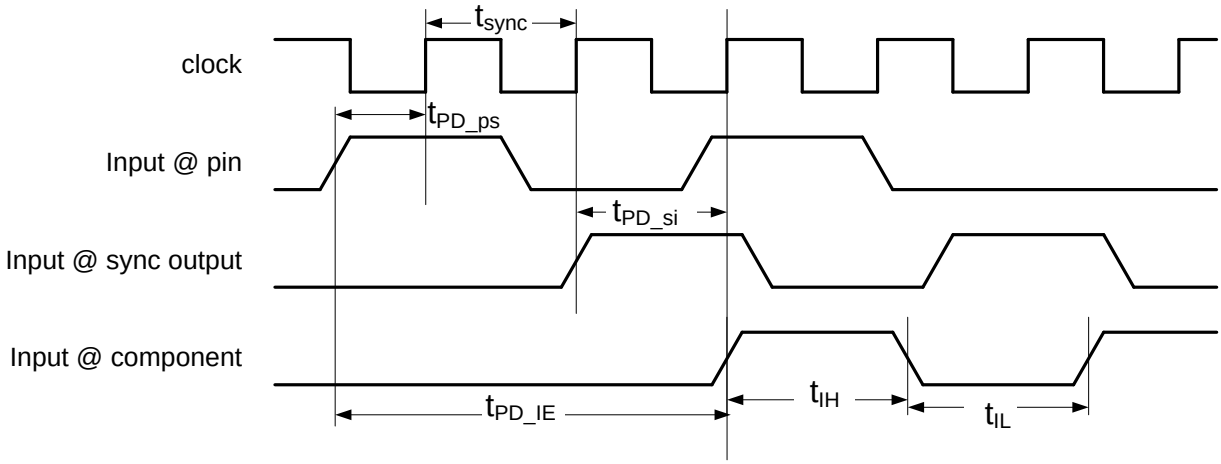


图 6. 输入配置 1 和 2：同步 时钟 = 组件时钟 = master_clock



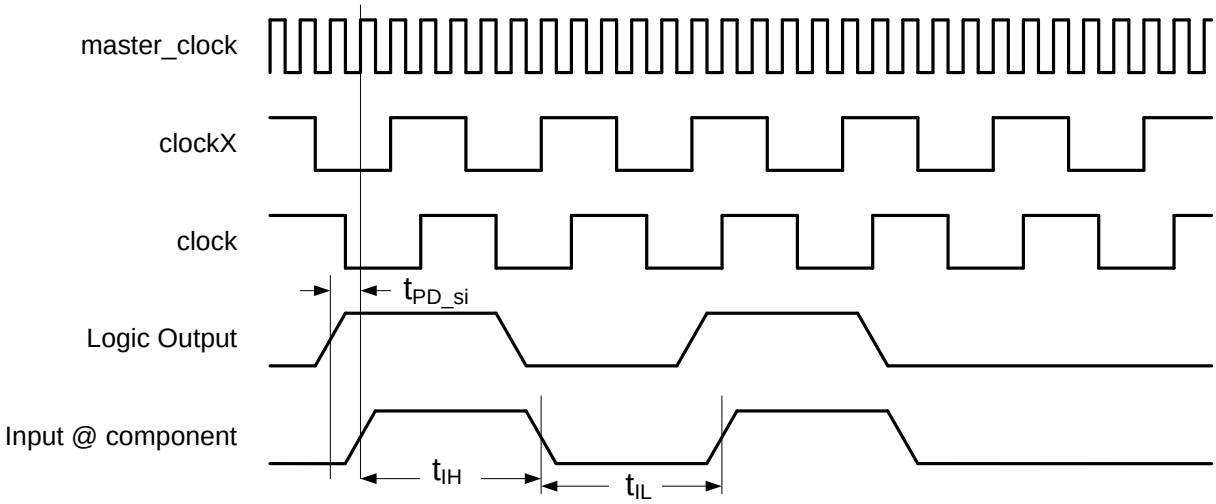
3. 输入由 PSoC 内部逻辑驱动，它基于与组件所使用的时钟不同的时钟同步（所有内部时钟都派生自 master_clock）。

   当表现按此方法配置的输入的特性时，同步器时钟快于、慢于或等于组件钟。这会生成如图 7、图 8 和 图 10 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。

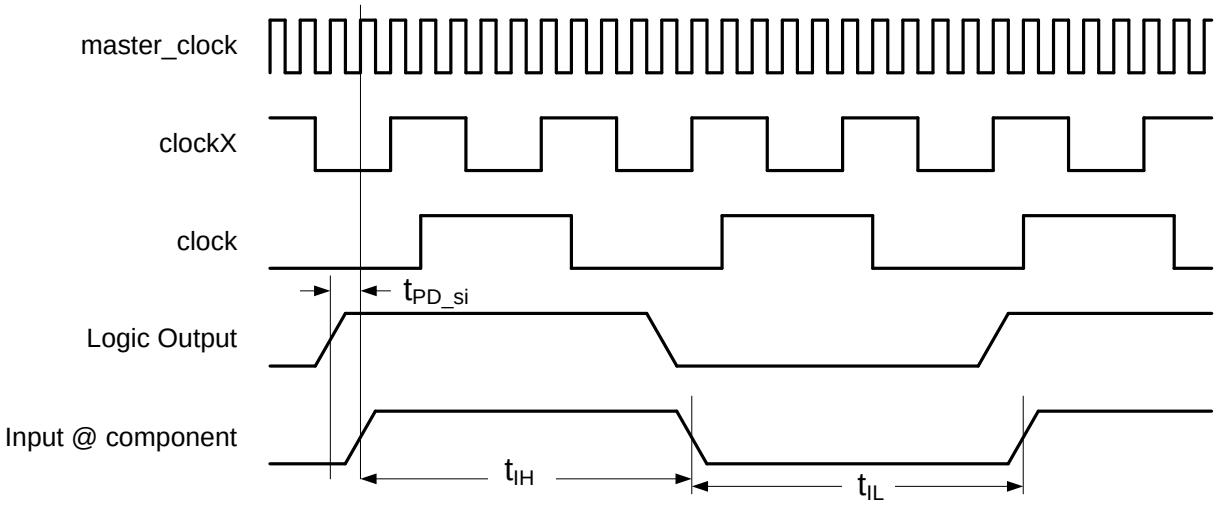
当表现按此方法配置的输入的特性时，同步器时钟等于组件时钟。这会生成如图 11 所示的特性参数。

图 7. 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）



此图显示了静态时序分析中针对时钟的信息。数字时钟域中的所有时钟与 master_clock 同步。但是，同一频率的两个时钟可以上升沿不对齐。因此，静态时序分析工具不了解时钟同步到哪个边沿，必须假设最小值为 1 个 master_clock 循环。这意味着 t_{PD_si} 现在对系统的 master_clock 的影响有限。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 8. 输入配置 3：同步 时钟频率 > 组件时钟频率



与图 7 中的方法几乎相同，所有时钟都派生自 master_clock。STA 在此配置中指明了对一个 master_clock 周期的 master_clock 的 t_{PD_si} 限制。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 9. 输入配置 3：同步器时钟频率 = master_clock > 组件时钟频率

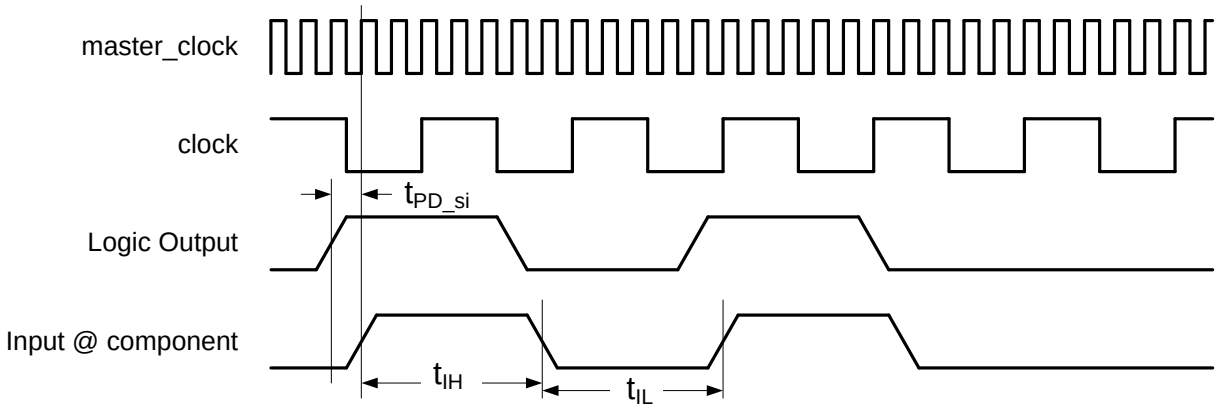
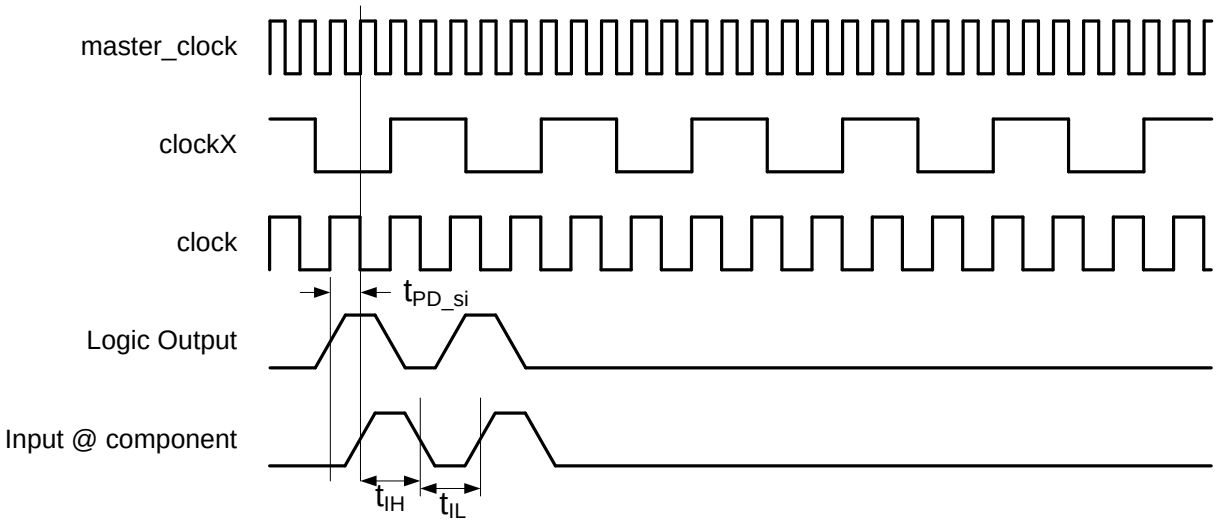
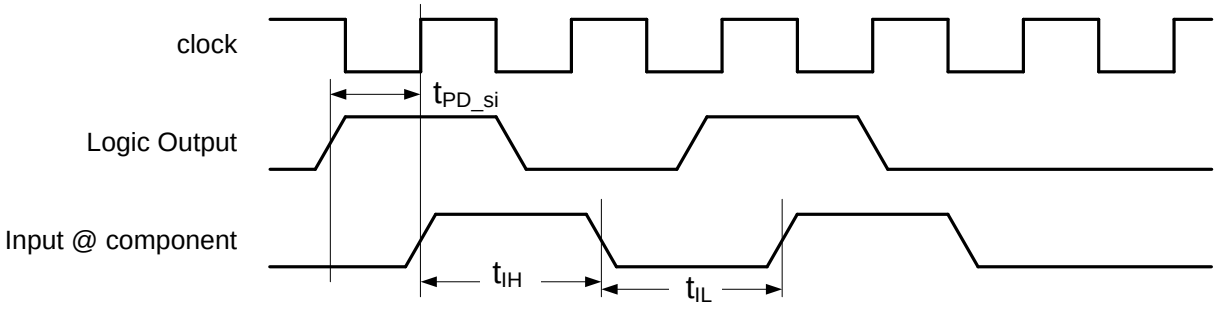


图 10. 输入配置 3：同步器时钟频率 < 组件时钟频率



与图 7 中的方法几乎相同，所有时钟都派生自 master_clock。STA 在此配置中指明了对一个 master_clock 周期的 master_clock 的 t_{PD_si} 限制。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 11. 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，在了解实现时使用的最关键参数是 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。最重要的是注意 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余留的余量。

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。此输入的输出端可以大于 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。此路径的输出端可以大于 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，必须查看为上面“逻辑至输入”说明显示的“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/影响原因
2.0.a	对数据表进行了少量编辑和更新	
2.0	已寄存的脉冲密度输出，用于消除短时脉冲。	任何组合输出都可能出现短时脉冲，具体取决于放置和信号之间的延迟。要消除短时脉冲，应将输出寄存。
	已寄存的使能输入和复位输入，以提高最大运行。	这些输入已组合使用，因此不会被 Creator 自动寄存，且有违规行为。寄存用于提高最大速度，并避免可能的短时脉冲。
	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。