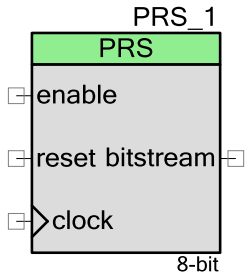
伪随机序列 (PRS)

2.0

特性



- 2-64 位 PRS 序列长度
- 时分复用模式
- 串行输出位流
- 连续或单步运行模式
- 标准或自定义多项式
- 标准或自定义种子值
- 启用输入可以与其他组件同步运行
- 可以从线性反馈移位寄存器 (LFSR) 直接读取计算后得出的伪随机数字

概述

伪随机序列 (PRS) 组件使用 LFSR 生成伪随机序列，由此输出一个伪随机位流。LFSR 是 Galois 格式（有时也称为模块格式），并使用所提供的最大代码长度或周期。使能输入保持在高电平时，PRS 组件便可以在启动后连续运行。使用除 0 以后的任意有效种子值，可以启动 PRS 数字发生器。

何时使用 PRS

LFSR 可以在硬件中实现。这使得 LFSR 在要求非常快速地生成伪随机序列的应用程序中非常有用，例如，直接序列扩频无线电。

全球定位系统使用 LFSR 快速传输相对时间偏移的高精度序列。此外，某些视频游戏控制器也将 LFSR 用作音响系统的一部分。

用作计数器

可以接受非二进制序列时，通过 LFSR 状态的重复序列，可以将其用作分频器，或计数器。与自然二进制计数器或格雷码计数器相比，LFSR 计数器具有简单的反馈逻辑，因此可以在较高的时间

频率下运行。然而，必须确保 LFSR 从未进入全零状态，例如，在启动时，通过将其预设为序列中其他任何状态。

输入/输出连接

本节介绍 PRS 组件的各种输入和输出连接。I/O 列表中的星号 (*) 表示该 I/O 是可隐藏 I/O， 其隐藏条件在该 I/O 的说明中。

clock – Input *（时钟 – 输入）*

时钟输入定义要计算 PRS 的信号。当选择 API 单步运行模式时，此输入不可用。

复位 – 输入*

复位输入用来定义要同步复位 PRS 的信号。此输入在选择时钟模式时不可用。只有使能输入保持在高电平时，才能复位 PRS。

使能 – 输入

使能输入保持在高电平时，PRS 组件便可以在启动后运行。此输入可以与其他组件同步运行。

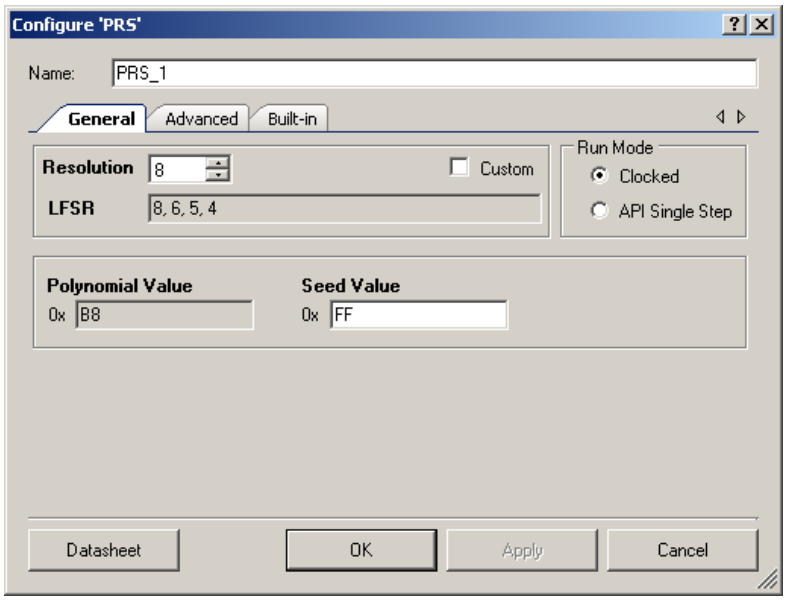
位流 - 输出

LFSR 输出。

元件参数

将 PRS 组件拖入设计中，双击该组件，打开 **Configure**（配置）对话框。该对话框有若干选项卡，可引导您完成 PRS 组件的设置过程。

一般选项卡



分辨率

这用来定义 PRS 序列的长度。该值设置介于 2-64 之间。默认值为 **8**。

默认情况下，**分辨率**用来定义 **LFSR** 系数**和多项式值**。这些系数取自下表。此外，此参数还用来定义最大代码长度或周期，如下表所示。

分辨率	LFSR	周期（$2^{分辨率} - 1$）		分辨率	LFSR	周期（$2^{分辨率} - 1$）
2	2, 1	3		34	34, 31, 30, 26	17179869183
3	3, 2	7		35	35, 34, 28, 27	34359738367
4	4, 3	15		36	36, 35, 29, 28	68719476735
5	5, 4, 3, 2	31		37	37, 36, 33, 31	137438953471
6	6, 5, 3, 2	63		38	38, 37, 33, 32	274877906943
7	7, 6, 5, 4	127		39	39, 38, 35, 32	549755813887
8	8, 6, 5, 4	255		40	40, 37, 36, 35	1099511627775
9	9, 8, 6, 5	511		41	41, 40, 39, 38	2199023255551
10	10, 9, 7, 6	1023		42	42, 40, 37, 35	4398046511103

分辨率	LFSR	周期（$2^{分辨率}-1$）		分辨率	LFSR	周期（$2^{分辨率}-1$）
11	11, 10, 9, 7	2047		43	43, 42, 38, 37	8796093022207
12	12, 11, 8, 6	4095		44	44, 42, 39, 38	17592186044415
13	13, 12, 10, 9	8191		45	45, 44, 42, 41	35184372088831
14	14, 13, 11, 9	16383		46	46, 40, 39, 38	70368744177663
15	15, 14, 13, 11	32767		47	47, 46, 43, 42	140737488355327
16	16, 14, 13, 11	65535		48	48, 44, 41, 39	281474976710655
17	17, 16, 15, 14	131071		49	49, 45, 44, 43	562949953421311
18	18, 17, 16, 13	262143		50	50, 48, 47, 46	1125899906842623
19	19, 18, 17, 14	524187		51	51, 50, 48, 45	2251799813685247
20	20, 19, 16, 14	1048575		52	52, 51, 49, 46	4503599627370495
21	21, 20, 19, 16	2097151		53	53, 52, 51, 47	9007199254740991
22	22, 19, 18, 17	4194303		54	54, 51, 48, 46	18014398509481983
23	23, 22, 20, 18	8388607		55	55, 54, 53, 49	36028797018963967
24	24, 23, 21, 20	16777215		56	56, 54, 52, 49	72057594037927935
25	25, 24, 23, 22	33554431		57	57, 55, 54, 52	144115188075855871
26	26, 25, 24, 20	67108863		58	58, 57, 53, 52	288230376151711743
27	27, 26, 25, 22	134217727		59	59, 57, 55, 52	576460752303423487
28	28, 27, 24, 22	268435455		60	60, 58, 56, 55	1152921504606846975
29	29, 28, 27, 25	536870911		61	61, 60, 59, 56	2305843009213693951
30	30, 29, 26, 24	1073741823		62	62, 59, 57, 56	4611686018427387903
31	31, 30, 29, 28	2147483647		63	63, 62, 59, 58	9223372036854775807
32	32, 30, 26, 25	4294967295		64	64, 63, 61, 60	18446744073709551615
33	33, 32, 29, 27	8589934591				

要手动设置 LFSR 系数：

定义**分辨率**。

选中 **Custom**（自定义）复选框。

在 **LFSR** 文本框中输入系数，用逗号分隔，然后按 **[Enter]**。系统将自动重新计算多项式值。

多项式值显示为十六进制格式。

注意：任何 LFSR 系数的值均不大于**分辨率**的值。

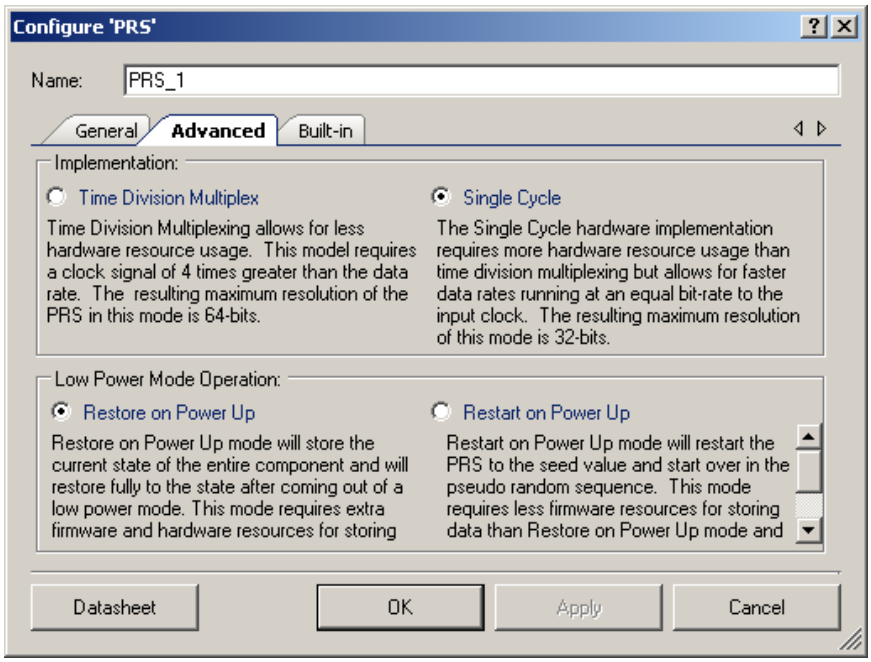
默认情况下，种子值设置为最大可能值（$2^{分辨率} - 1$）。其值可以更改为除 0 以外的任何值。种子值显示为十六进制格式。

注意：更改**分辨率**时，可以将**种子值**复位到默认值。

运行模式

此参数用于将组件运行模式定义为连续运行或单步运行模式。您可以选择 **Clocked**（时钟）（默认值）或 **API Single Step**（API 单步执行）。如果连续读取 PRS 的值，或需要读取一个值，则必须停止时钟或在时钟模式下将使能设置为低电平。

“高级”选项卡



PRS **Advanced**（高级）选项卡包含以下设置：

实现

这用来定义 PRS 组件的实现：包含或不包含时间复用（**单周期**）。默认值为**单周期**。

低功耗模式运行

这用来定义低功耗模式下的 PRS 行为。默认值为 **Restore on Power Up**（加电时存储）。

局部参数（供 API 使用）

这些参数用于 API， 而非在 GUI 中显示：

- **PolyValueLower(uint32) –** 包含下半部分多项式值（十六进制格式）。默认值为 0xB8h (**LFSR**= [8,6,5,4])，因为默认分辨率值为 8。
- **PolyValueUpper(uint32) –** 包含下半部分多项式值（十六进制格式）。默认值为 0x00h，因为默认分辨率为 8。
- **SeedValueLower (uint32) –** 包含下半部分种子值（十六进制格式）。默认值为 0xFFh，因为默认分辨率为 8。
- **SeedValueUpper (uint32) –** 包含上半部分种子值（十六进制格式）。默认值为 0，因为默认分辨率为 8。

时钟选择

如果选择 **Run Mode**（运行模式）参数的 **Clocked**（时钟）选项，则必须连接时钟源。

注意：如果为 **Implementation**（实现）参数选择 **Time Division Multiplex**（时间分频复用），则生成分辨率的正确 PRS 序列大于 8，则要求比数据速率大 4 倍的时钟信号。

放置

PRS 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

单周期，API 单步执行

资源	资源类型				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	
1..8-位分辨率	1	1	0	1	152	3	2
9..16-位分辨率	2	1	0	1	188	4	2
17..24-位分辨率	3	1	0	1	244	6	2
25..32-位分辨率	4	1	0	1	240	6	2

时间分频复用，API 单步执行

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	
9..16-位分辨率	1	3	1	1	302	6	2
17..24-位分辨率	2	4	1	1	548	8	2
25..32-位分辨率	2	3	1	1	624	8	2
33..40-位分辨率	3	4	1	1	753	12	2
41..48-位分辨率	3	3	1	1	870	12	2
49..56-位分辨率	4	4	1	1	956	12	2
57..64-位分辨率	4	3	1	1	1035	12	2

单周期，时钟

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	
1..8-位分辨率	1	1	0	1	180	3	4
9..16-位分辨率	2	1	0	1	244	4	4
17..24-位分辨率	3	1	0	1	319	6	4
25..32-位分辨率	4	1	0	1	302	6	4

时分复用，时钟

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	
9..16-位分辨率	1	3	1	1	318	6	4
17..24-位分辨率	2	4	1	1	512	8	4
25..32-位分辨率	2	3	1	1	686	8	4
33..40-位分辨率	3	4	1	1	855	12	4
41..48-位分辨率	3	3	1	1	990	12	4
49..56-位分辨率	**4**	4	1	1	1096	12	4
57..64-位分辨率	**4**	3	1	1	1175	12	4

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“PRS_1”分配给指定设计中组件的第一个实例。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。出于可读性考虑，下表中使用的实例名称为 "PRS"。

函数	说明
PRS_Start()	初始化由自定义程序提供的种子和多项式寄存器。在输入时钟上升沿启动 PRS 计算功能。
PRS_Stop()	停止 PRS 计算功能。
PRS_Sleep()	停止 PRS 计算功能并保存 PRS 配置。
PRS_Wakeup()	存储 PRS 配置，并在输入时钟上升沿启动 PRS 计算功能。
PRS_Init()	使用初始值初始化种子和多项式寄存器。
PRS_Enable()	在输入时钟上升沿启动 PRS 计算功能。
PRS_SaveConfig()	保存种子和多项式寄存器。
PRS_RestoreConfig()	存储种子和多项式寄存器。

函数	说明
PRS_Step()	使用 API 单步执行模式时，PRS 增量为 1。
PRS_WriteSeed()	写入种子值。
PRS_WriteSeedUpper()	写入上半部分种子值。仅为 33-64 位 PRS 生成。
PRS_WriteSeedLower()	写入下半部分种子值。仅为 33-64 位 PRS 生成。
PRS_Read()	读取 PRS 值。
PRS_ReadUpper()	读取上半部分 PRS 值。仅为 33-64 位 PRS 生成。
PRS_ReadLower()	读取下半部分 PRS 值。仅为 33-64 位 PRS 生成。
PRS_WritePolynomial()	写入 PRS 多项式值。
PRS_WritePolynomialUpper()	写入上半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。
PRS_WritePolynomialLower()	写入下半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。
PRS_ReadPolynomial()	读取 PRS 多项式值。
PRS_ReadPolynomialUpper()	读取上半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。
PRS_ReadPolynomialLower()	读取下半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。

全局变量

变量	说明
PRS_initVar	表示是否完成初始化 PRS。变量将初始化为 0，并在第一次调用 PRS_Start() 时设置为 1。这样，第一次调用 PRS_Start() 子程序后，组件不用重新初始化即可重启。 如需重新初始化组件，可在 PRS_Start() 或 PRS_Enable() 函数前调用 PRS_Init() 函数。

void PRS_Start(void)

说明： 初始化种子和多项式寄存器。在输入时钟上升沿启动 PRS 计算功能。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void PRS_Stop(void)

说明：	停止 PRS 计算功能。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_Sleep(void)

说明：	停止 PRS 计算功能，然后保存 PRS 配置。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_Wakeup(void)

说明：	存储 PRS 配置，并在输入时钟上升沿启动 PRS 计算功能。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_Init(void)

说明：	使用初始值初始化种子和多项式寄存器。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_Enable(void)

说明：	在输入时钟上升沿启动 PRS 计算功能。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_SaveConfig(void)

说明：	保存种子和多项式寄存器。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_RestoreConfig(void)

说明：	存储种子和多项式寄存器。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_Step(void)

说明：	使用 API 单步执行模式时，PRS 增量为 1。
参数：	None（无）
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

void PRS_WriteSeed(uint8/16/32 seed)

说明： 写入种子值。

参数： uint8/16/32 seed: 种子值

Return Value（返回值）： None（无）

Side Effects（副作用）： 根据掩码 = $2^{分辨率} – 1$ 截取种子值。
例如，如果 PRS 分辨率为 14 位，则掩码值为：掩码 = $2^{14} – 1$ = 0x3FFFu。
截取种子值 = 0xFFFFu：种子与掩码 = 0xFFFFu AND 0x3FFFu = 0x3FFFu。

void PRS_WriteSeedUpper(uint32 seed)

说明： 写入上半部分种子值。仅为 33-64 位 PRS 生成。

参数： uint32 seed: 上半部分种子值

Return Value（返回值）： None（无）

Side Effects（副作用）： 根据掩码 = $2^{(分辨率 – 32)} – 1$ 截取上半部分种子值。
例如，如果 PRS 分辨率为 35 位，则掩码值为：
$2^{(35 – 32)} – 1 = 2^{\wedge 3} – 1$ = 0x0000 0007u.
截取上半部分种子值 = 0x0000 00FFu：
上半部分种子与掩码 = 0x0000 00FFu AND 0x0000 0007u = 0x0000 0007u。

void PRS_WriteSeedLower(uint32 seed)

说明： 写入下半部分种子值。仅为 33-64 位 PRS 生成。

参数： uint32 seed: 下半部分种子值

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

uint8/16/32 PRS_Read(void)

说明：	读取 PRS 值。
参数：	None（无）
Return Value（返回值）：	uint8/16/32: 返回 PRS 值。
Side Effects（副作用）：	None（无）

uint32 PRS_ReadUpper(void)

说明：	读取上半部分 PRS 值。仅为 33-64 位 PRS 生成。
参数：	None（无）
Return Value（返回值）：	uint32: 返回上半部分 PRS 值。
Side Effects（副作用）：	None（无）

uint32 PRS_ReadLower(void)

说明：	读取下半部分 PRS 值。仅为 33-64 位 PRS 生成
参数：	None（无）
Return Value（返回值）：	uint32: 返回下半部分 PRS 值。
Side Effects（副作用）：	None（无）

void PRS_WritePolynomial(uint8/16/32 polynomial)

说明：	写入 PRS 多项式值。
参数：	uint8/16/32 polynomial: PRS 多项式。
Return Value（返回值）：	None（无）
Side Effects（副作用）：	根据掩码 = $2^{分辨率} – 1$ 截取多项式值。 例如，如果 PRS 分辨率为 14 位，则掩码值为：掩码 = $2^{14} – 1$ = 0x3FFFu。 截取多项式值 = 0xFFFFu： 多项式与掩码 = 0xFFFFu AND 0x3FFFu = 0x3FFFu。

void PRS_WritePolynomialUpper(uint32 polynomial)

说明：	写入上半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。
参数：	uint32 polynomial: 上半部分 PRS 多项式值。
Return Value（返回值）：	None（无）
Side Effects（副作用）：	根据掩码 = $2^{(分辨率-32)} - 1$ 截取上半部分多项式值。 例如，如果 PRS 分辨率为 35 位，则掩码值为： $2^{(35-32)} - 1 = 2^3 - 1$ = 0x0000 0007u. 截取上半部分多项式值 = 0x0000 00FFu： 上半部分多项式与掩码 = 0x0000 00FFu AND 0x0000 0007u = 0x0000 0007u。

void PRS_WritePolynomialLower(uint32 polynomial)

说明：	写入下半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。
参数：	uint32 polynomial: 下半部分 PRS 多项式值
Return Value（返回值）：	None（无）
Side Effects（副作用）：	None（无）

uint8/16/32 PRS_ReadPolynomial(void)

说明：	读取 PRS 多项式值。
参数：	None（无）
Return Value（返回值）：	uint8/16/32: 返回 PRS 多项式值。
Side Effects（副作用）：	None（无）

uint32 PRS_ReadPolynomialUpper(void)

说明：	读取上半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。
参数：	None（无）
Return Value（返回值）：	uint32: 返回上半部分 PRS 多项式值。
Side Effects（副作用）：	None（无）

uint32 PRS_ReadPolynomialLower(void)

说明：	读取下半部分 PRS 多项式值。仅为 33-64 位 PRS 生成。
参数：	None（无）
Return Value（返回值）：	uint32: 返回下半部分 PRS 多项式值。
Side Effects（副作用）：	None（无）

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

PRS 运行模式：时钟

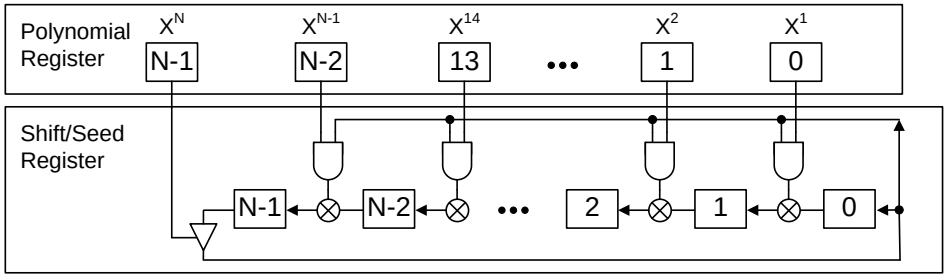
在此模式下，只要使能输入保持在高电平时，PRS 组件便可以在启动后连续运行。

PRS 运行模式：API 单步执行

在此模式下，通过 API 调用增加 PRS。

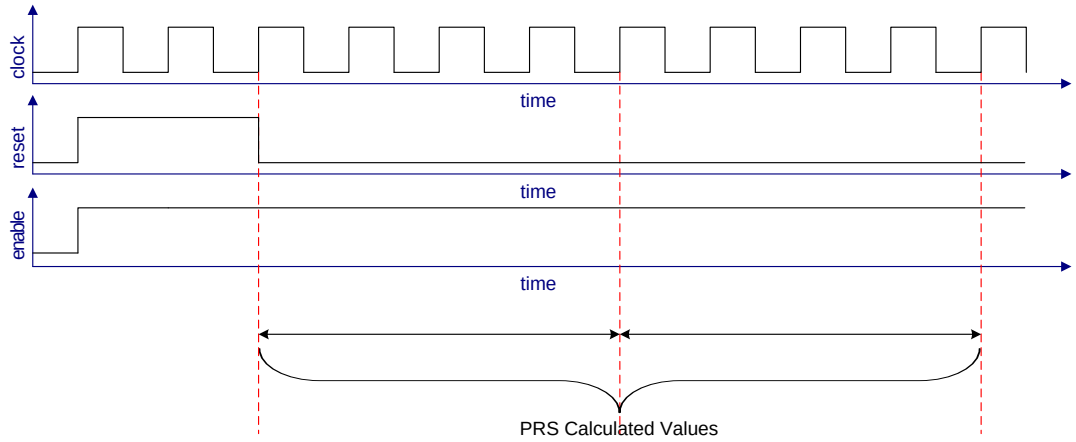
框图和配置

PRS 作为一组 UDB 配置得以实现。在以下框图中显示该实现。

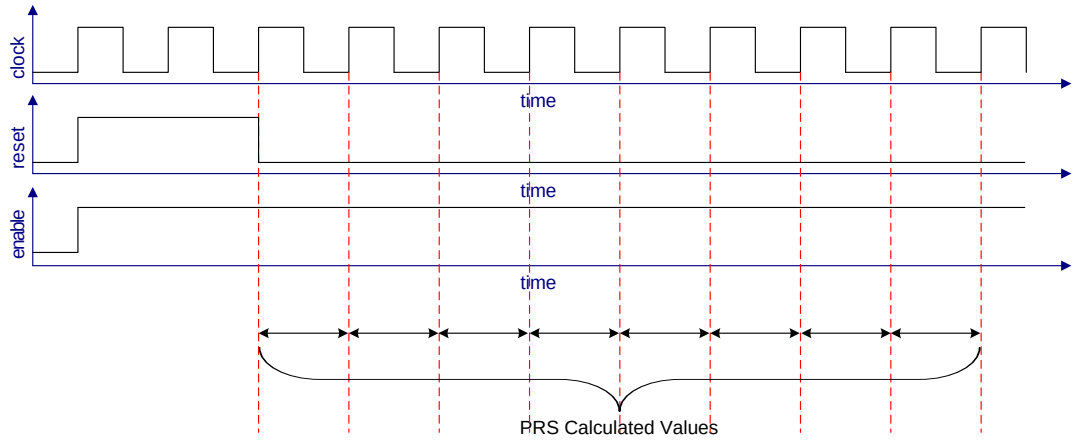


时序图

时分复用实现模式



单周期实现模式



寄存器

多项式寄存器（2-64 位，根据分辨率）

多项式寄存器包含多项式值。通过 PRS_WritePolynomial()、PRS_WritePolynomialUpper() 或 PRS_WritePolynomialLower() 函数，可以对该值进行更改。此外，还可以使用 PRS_ReadPolynomial()、PRS_ReadPolynomialUpper() 或 PRS_ReadPolynomialLower() 读取当前多项式值。

移位/种子寄存器（2-64 位，根据寄存器）

移位/种子寄存器包含种子值。可以使用 PRS_WriteSeed()、PRS_WriteSeedUpper() 或 PRS_WriteSeedLower() 函数，可以对该值进行更改。此外，还可以使用 PRS_ReadSeed()、PRS_ReadSeedUpper() 或 PRS_ReadSeedLower() 读取当前种子值。

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 [1]	最小值	典型值	最大值	单位
f_{CLOCK}	组件时钟频率 [2]	配置 1	–	–	57	MHz
		配置 2	–	–	57	MHz
		配置 3	–	–	35	MHz

[1] 配置：

配置 1：
分辨率：8 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：单周期

配置 2：
分辨率：8 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：单周期

配置 3：
分辨率：16 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：时分复用

配置 4：
分辨率：16 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：时分复用

配置 5：
分辨率：32 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：单周期

配置 6：
分辨率：32 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：单周期

配置 7：
分辨率：64 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：时分复用

配置 8：
分辨率：64 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：时分复用

[2] 如果选择了“时分复用实现”，则组件时钟频率必须比数据速率大 4 倍。

参数	说明	配置 [1]	最小值	典型值	最大值	单位
		配置 4	–	–	30	MHz
		配置 5	–	–	43	MHz
		配置 6	–	–	40	MHz
		配置 7	–	–	25	MHz
		配置 8	–	–	30	MHz
t_{clockH}	输入时钟高电平时间 [3]	不可用	–	0.5	–	$1/f_{clock}$
t_{clockL}	输入时钟低电平时间 [3]	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 [4]	1	–	–	STA [5]	ns
t_{PD_ps}	输入路径延迟，要同步的引脚 [6]	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 [5]	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} [7]	–	–	ns
t_{IL}	输入低电平时间	1,2,3,4	t_{CY_clock} [7]	–	–	ns
输出						
	输出寄存器到寄存器的时间		–	–	–	
	引脚路径输出延迟		–	–	–	

[3] $t_{CY_clock} = 1/f_{CLOCK}$。这是一个时钟周期的周期时间。

[4] 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 STA 分析的许多输入的额定值。

[5] t_{PD_ps} 和 t_{PD_si} 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。在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。此处列出的数字是该器件上可用的所有引脚的额定值。

若选择时分复用实现，则 [7] $t_{CY_clock} = 4\times [1/f_{CLOCK}]$。

时序特性“所有路由的最大值”

参数	说明	配置[1]	最小值	典型值	最大值[2]	单位
f_{CLOCK}	组件时钟频率 [3]	配置 1	–	–	29	MHz

[1] 配置：

配置 1：
分辨率：8 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：单周期

配置 2：
分辨率：8 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：单周期

配置 3：
分辨率：16 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：时分复用

配置 4：
分辨率：16 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：时分复用

配置 5：
分辨率：32 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：单周期

配置 6：
分辨率：32 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：单周期

配置 7：
分辨率：64 位
运行模式：API 单步执行
低功耗模式运行：加电时存储
实现：时分复用

配置 8：
分辨率：64 位
运行模式：时钟
低功耗模式运行：加电时存储
实现：时分复用

[2] “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。此值提供了一个基础，因此用户不必担心在以该频率或低于频率运行该组件时遇到时序问题。

[3] 如果选择了“时分复用实现”，则组件时钟频率必须比数据速率大 4 倍。

参数	说明	配置[1]	最小值	典型值	最大值[2]	单位
		配置 2	–	–	29	MHz
		配置 3	–	–	18	MHz
		配置 4	–	–	15	MHz
		配置 5	–	–	22	MHz
		配置 6	–	–	20	MHz
		配置 7	–	–	13	MHz
		配置 8	–	–	15	MHz
t_{clockH}	输入时钟高电平时间 [4]	不可用	–	0.5	–	$1/f_{clock}$
t_{clockL}	输入时钟低电平时间 [4]	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 [5]	1	–	–	STA [6]	ns
t_{PD_ps}	输入路径延迟，要同步的引脚 [7]	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 [6]	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} [8]	–	–	ns
t_{IL}	输入低电平时间	1,2,3,4	t_{CY_clock} [8]	–	–	ns
输出						
	输出寄存器到寄存器的时间		–	–	–	
	引脚路径输出延迟		–	–	–	

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

[5] 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 STA 分析的许多输入的额定值。

[6] t_{PD_ps} 和 t_{PD_si} 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。在静态时序分析结果中一定能够找到这些值。

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

若选择时分复用实现，则 [8] $t_{CY_clock} = 4\times [1/f_{CLOCK}]$。

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。您可以使用下列方法通过 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

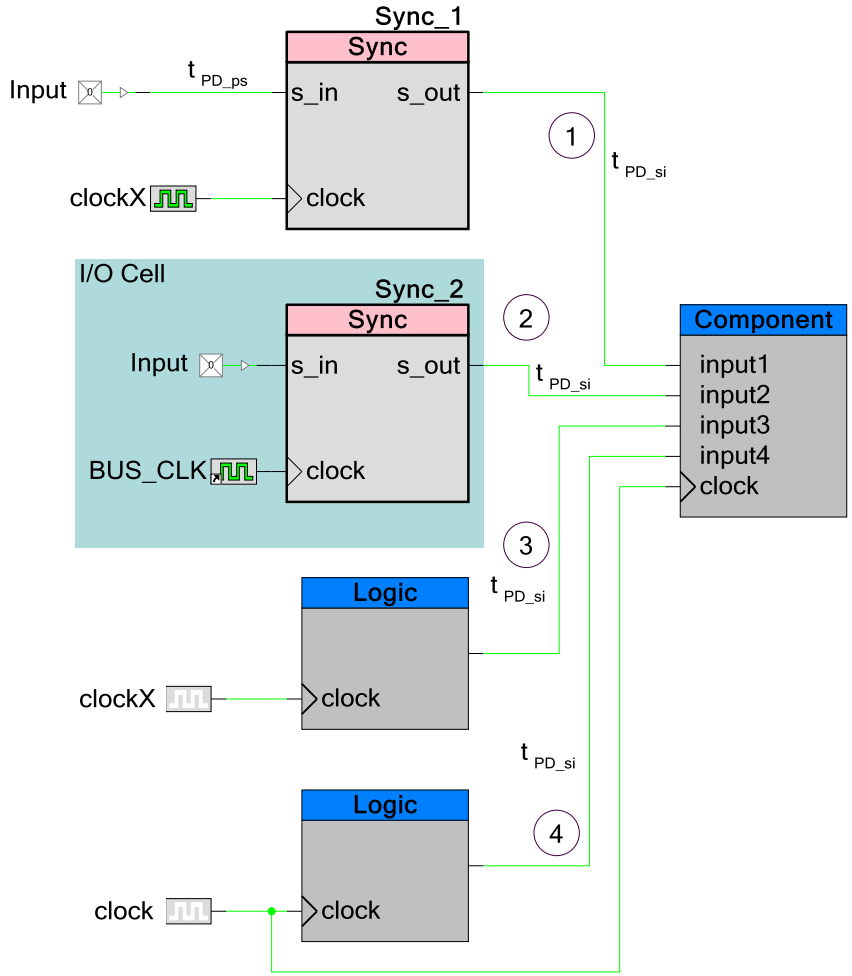
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	66.000 MHz	63.295 MHz	VIOLATION
CharComp_clock	33.000 MHz	57.594 MHz	

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。要完全解析系统如何工作，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. 组件时序的输入配置说明



配置	组件时钟	同步器时钟（频率）	图形
1	master_clock	master_clock	图 6
1	clock	master_clock	图 4
1	clock	clockX = 时钟 [1]	图 2
1	clock	clockX > 时钟	图 3
1	clock	clockX < 时钟	图 5
2	master_clock	master_clock	图 6
2	clock	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	clock	master_clock	图 9
3	clock	clockX = 时钟 [1]	图 7
3	clock	clockX > 时钟	图 8
3	clock	clockX < 时钟	图 10
4	master_clock	master_clock	图 11
4	clock	clock	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的时钟不同的内部时钟（所有内部时钟均派生于 master_clock）。

   当表现按此方法配置的输入特性时，clockX 可以快于、等于或慢于组件时钟。此外，它还可以等于 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）

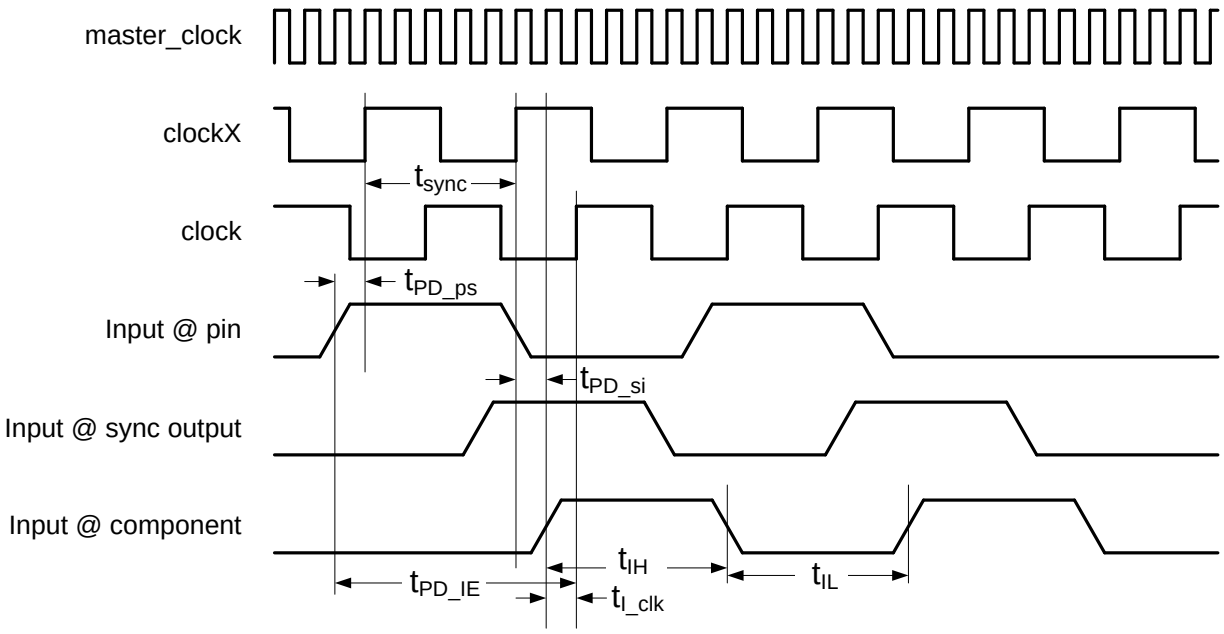


图 3. 输入配置 1 和 2：同步 时钟频率 > 组件时钟频率

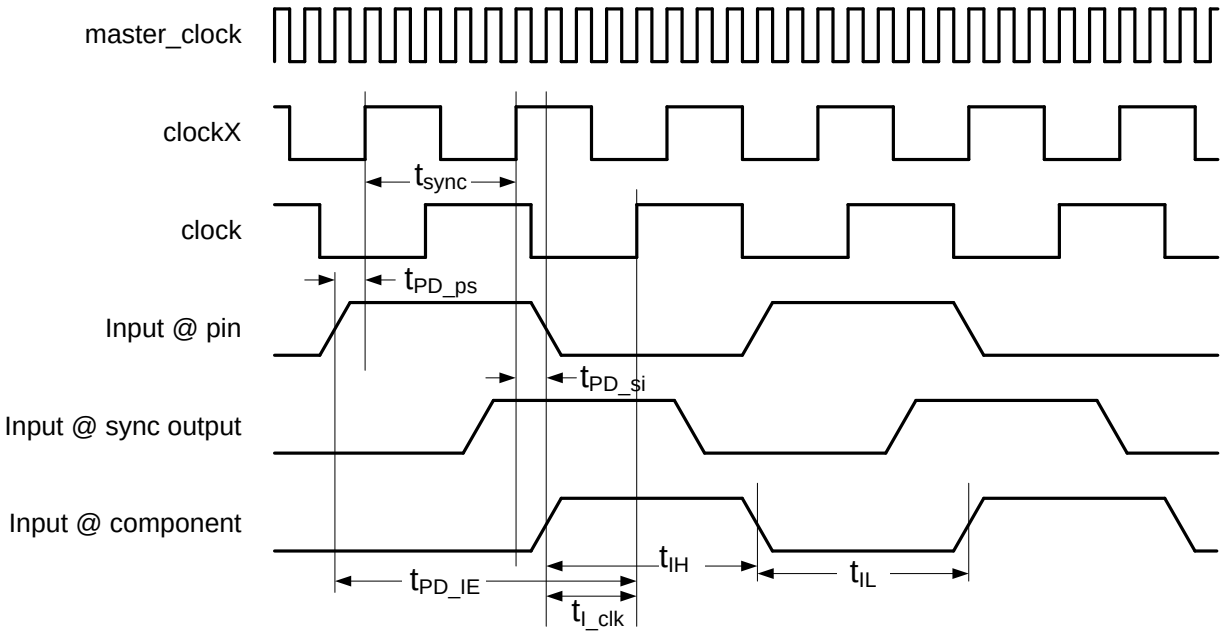


图 4. 输入配置 1 和 2：[同步 时钟频率 == master_clock] > 组件时钟频率

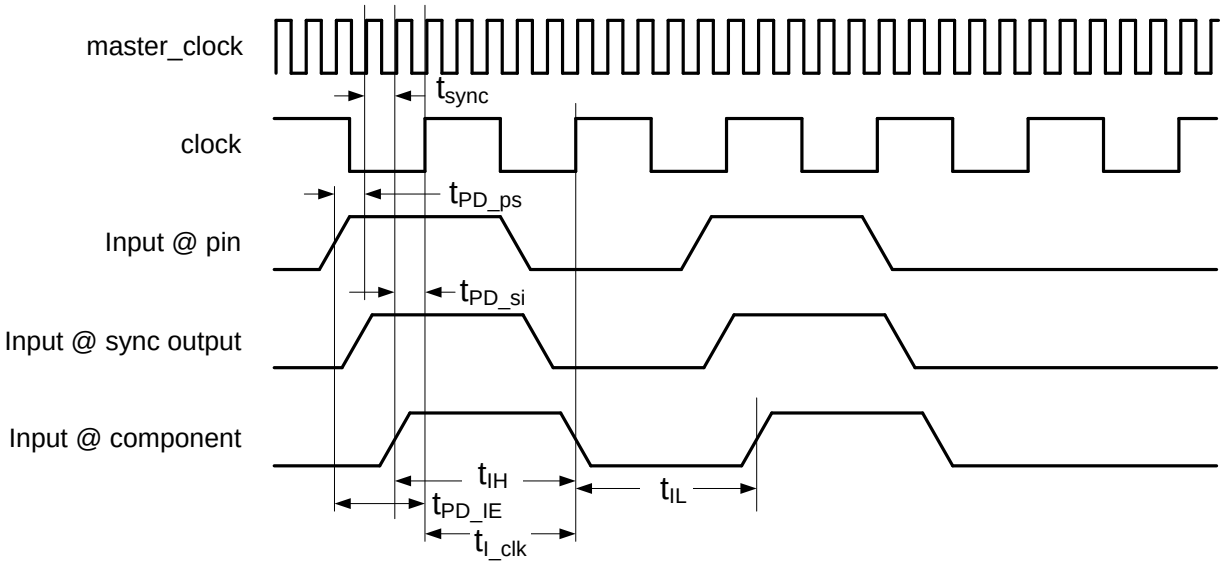


图 5. 输入配置 1：同步 时钟频率 > 组件时钟频率

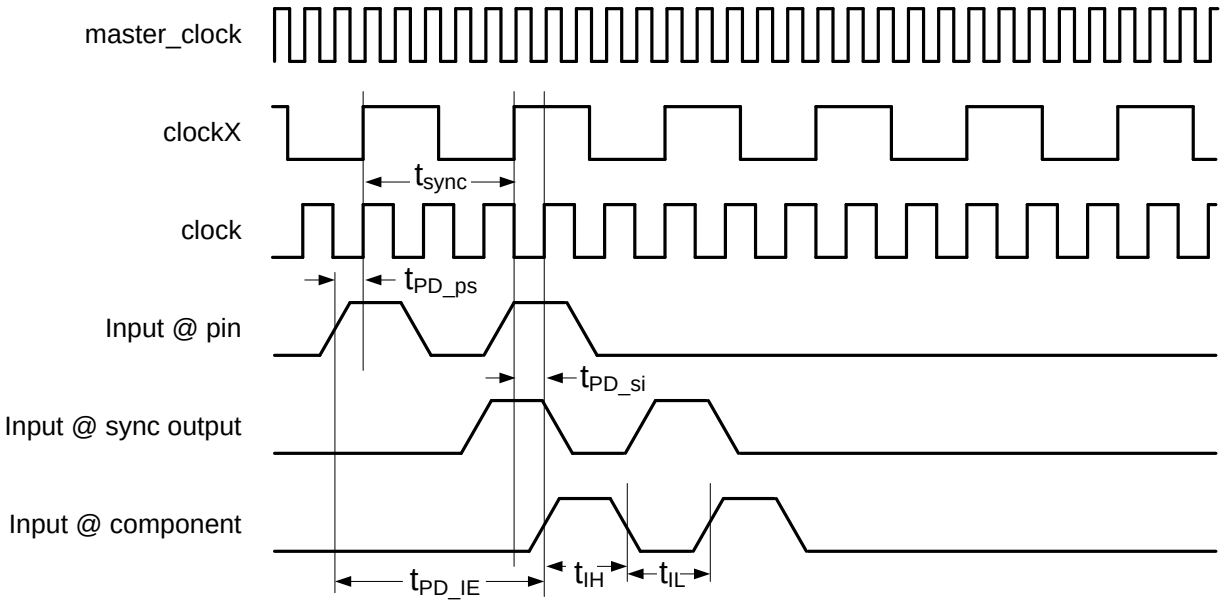
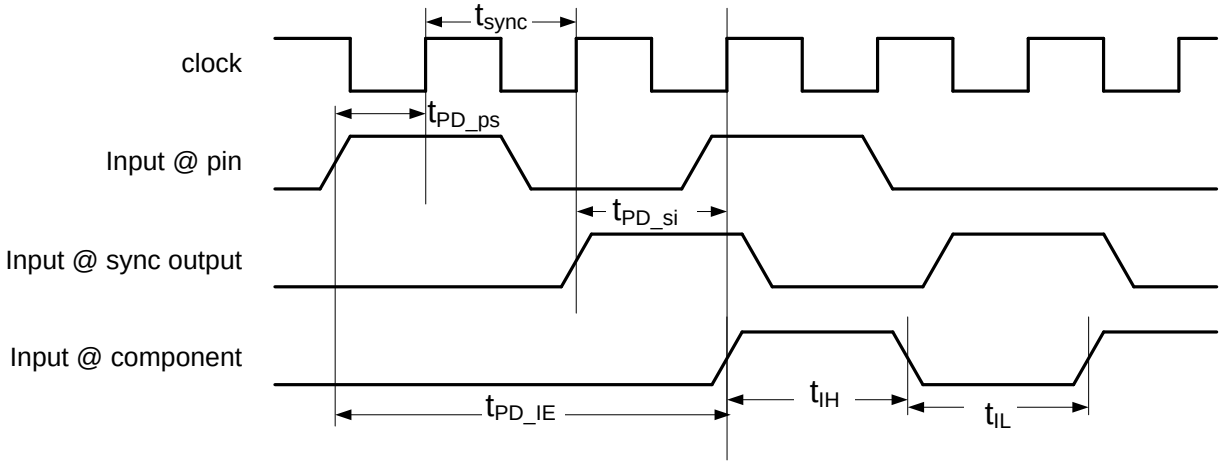


图 6. 输入配置 1 和 2：同步 时钟 = 组件时钟 = master_clock



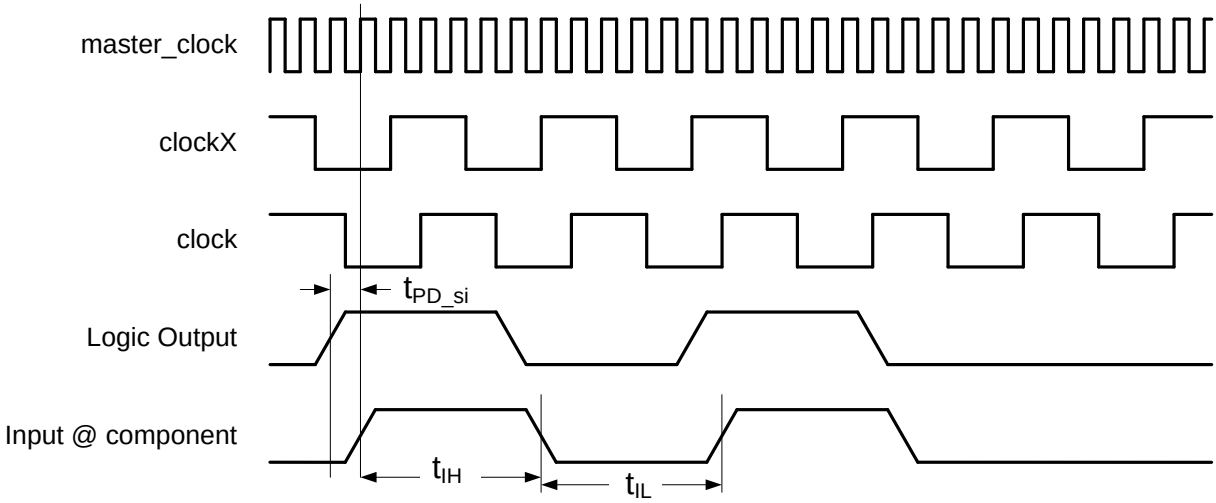
3. 输入由 PSoC 内部逻辑驱动，它基于与组件所使用的时钟不同的时钟同步（所有内部时钟都派生自 master_clock）。

   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 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。

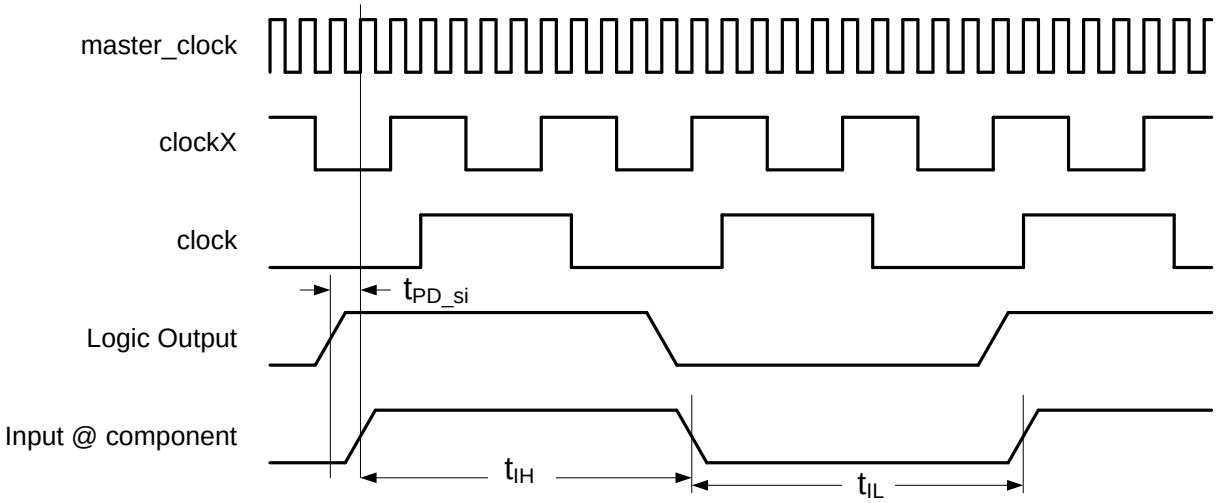
   当表现按此方法配置的输入特性时，同步器时钟等于组件时钟。这会生成如图 11 所示的特性参数。

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）



此图表述静态时序分析所具备的时钟的相关信息。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，必须假设最小值为 1 个 master_clock 循环。这意味着 t_{PD_si} 此时对系统 master_clock 造成有限影响。如果此路径延迟时间过长，则显示 master_clock 建立时间不满足要求。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

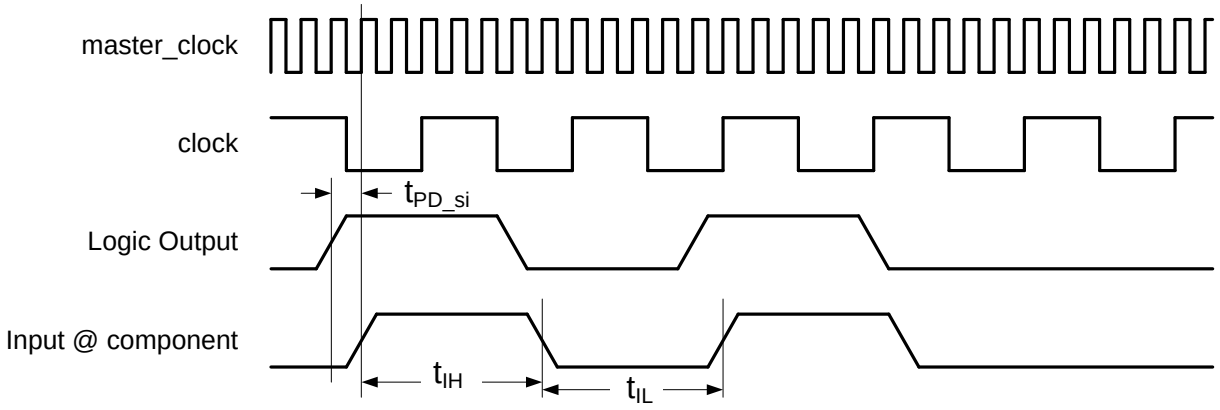
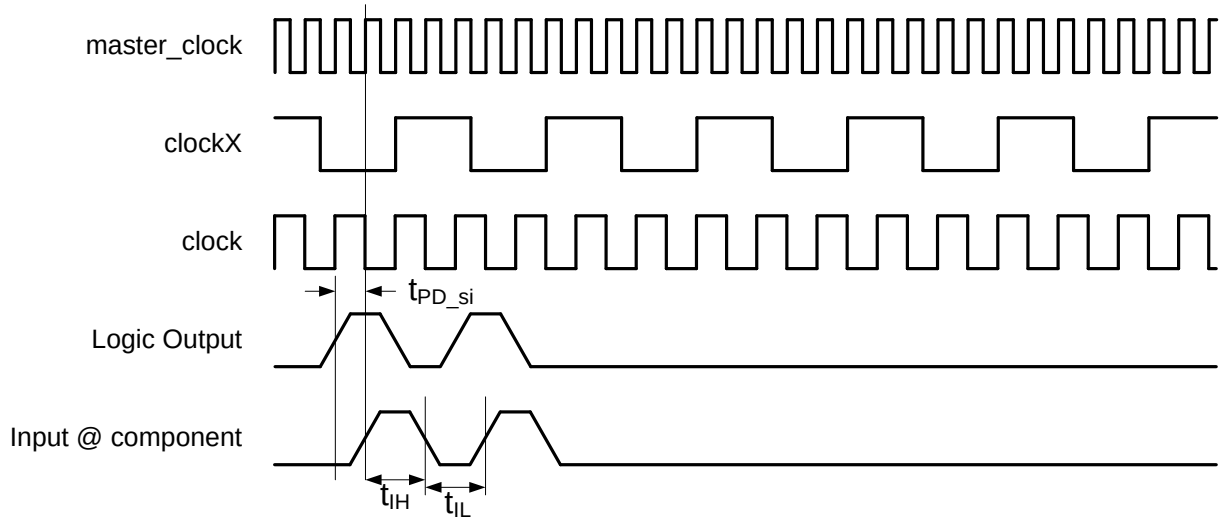
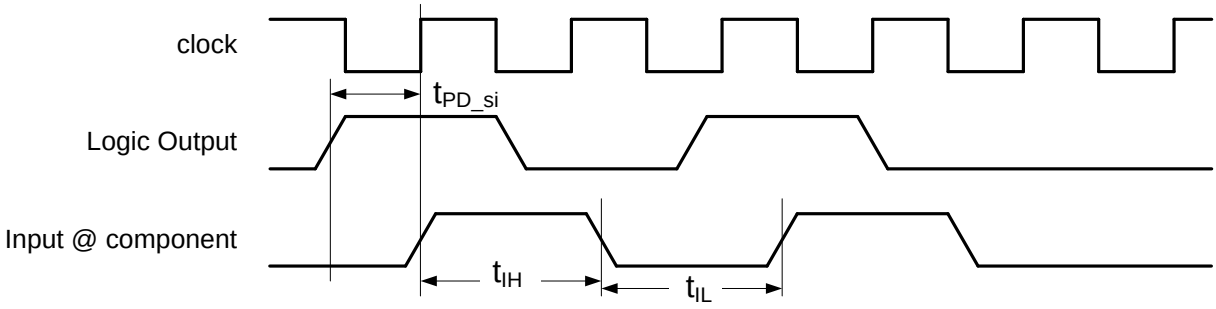


图 10. 输入配置 3：同步器时钟频率 < 组件时钟频率



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 t_{PD_si} 定义最大组件时钟频率。t_{I_Clk} 不源自 STA 结果，但用于表示何时寄存 t_{PD_IE}。这是同步器与组件时钟之间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)_SYNC:synccell.syncd	BUS_CLK	13.246

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 CharComp_clock

Destination clock CharComp_clock (Actual freq: 33.000 MHz)

+Source clock BUS_CLK

Source clock BUS_CLK (Actual freq: 66.000 MHz)

Start	End	Period (ns)	Max Freq	Frequency	Violation
input1(0)_SYNC:synccell.syncq	\CharComp:sC8:PRSdp:u0\:datapathcell.sync_ov_msb	15.799	63.295 MHz	66.000 MHz	SETUP

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。

+Clock to output times

+Clock to output times from clock CharComp_clock

Start	Register	End	Delay (ns)
CharComp_clock	\CharComp:sC8:PRSdp:u0\:datapathcell.regq_a0	CharComp_bitstream(0):iocell.pad_out	27.253

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.b	更新了数据表中的资源信息	
2.0.a	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	
2.0	添加了对 PSoC 3 Production 的芯片的支持。变更内容包括： ■ 补充了时分复用实现的 4 倍时钟 ■ 在 1 倍时钟上实现的单周期此时可用于 1-32 位。 ■ 在 4 倍时钟上实现的时分复用此时可用于 9-64 位。 ■ 补充了同步输入信号复位。 ■ 补充了同步输入信号使能。 ■ 在 Configure（配置）对话框新增实现与低功耗模式参数的 'Advanced'（‘高级’）页面	新要求支持 PSoC 3 Production 器件，由此创建 PRS 组件新的 2.0 版本
	补充了 PRS_Sleep()/PRS_Wakeup() 和 PRS_Init()/PRS_Enable() APIs。	为支持低功耗模式并提供常用接口，以单独控制大多数组件的初始化和启用。
	更新了函数 PRS_WriteSeed() 和 PRS_WriteSeedUpper()。	掩码参数用于在写入时截取种子值以定义分辨率。
	将复位 DFF 触发添加到多项式写入函数：PRS_WritePolynomial()、PRS_WritePolynomialUpper() 和 PRS_WritePolynomialLower()。	开始计算前，必须在正确状态下设置 DFF 触发（多项式最重要的位始终为 1）。要满足此条件，任何写入种子或多项式寄存器中的值均可以复位 DFF 触发。
	更新了 Configure（配置）对话框以用来支持某些参数的“表达式查看”。	“表达式查看”用于直接访问符号参数。通过此查看，您可以将组件参数与外部所需参数进行连接。
	更新了 Configure（配置）对话框以用来添加各种参数的错误图标。	如果在文本框中输入的值不正确，则显示该错误图标，包含问题描述的工具提示。这比使用单独错误消息更易于使用。

© 赛普拉斯半导体公司，2010-2012。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。