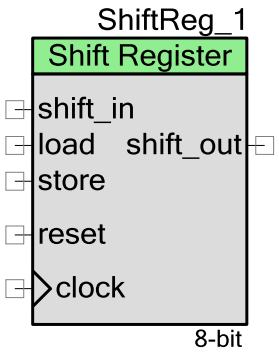
移位寄存器 (ShiftReg)

2.0

特性



- 移位寄存器长度可调：2 到 32 位
- 同时移入和移出
- 右移或左移
- 复位输入强制清零移位寄存器
- CPU 或 DMA 可读取移位寄存器值
- CPU 或 DMA 可写入移位寄存器值

概述

移位寄存器 (ShiftReg) 组件提供基于并行寄存器的数据同步移入和移出操作。CPU 或 DMA 可读/写并行寄存器的值。移位寄存器组件提供与标准 74xxx 系列逻辑移位寄存器类似的通用功能，该系列包括：74164、74165、74166、74194、74299、74595 和 74597。在大多数应用中，移位寄存器组件与其他组件和逻辑配合使用，以创建更高级的特定应用功能，例如用于对移位位数进行计数的计数器。

在一般的使用中，移位寄存器组件用作 2 到 32 位长度的移位寄存器，在时钟输入的上升沿移位数据。组件的移位方向可配置。可进行右向移位，即最高有效位（MSB）移入输入端，最低有效位（LSB） 移出输出端； 或进行左向移位，即最低有效位（LSB）移入输入端，且最高有效位（MSB）移出输出端。

移位寄存器的值可通过 CPU 或 DMA 随时写入。当检测到装载（load） 输入端的上升沿时，组件时钟的上升沿会将待处理的数据从 FIFO（之前由 CPU 或 DMA 写入）传输到移位寄存器。当检测到存储（store）输入端的上升沿时，组件时钟的上升沿会将当前移位寄存器的值传输到 FIFO，之后可由 CPU 读取此值。

装载（load）信号、存储（store）信号和复位信号的任意组合产生的信号均可触发移位寄存器组件的中断操作。

何时使用移位寄存器

移位寄存器的一种最常见用法是在串行接口和并行接口之间进行转换操作。这很有用，因为很多电路以并行方式对比特组进行处理，但串行接口更容易构建。

移位寄存器也可用作简单的延迟电路。在大部分情况下，移位寄存器需要配合额外的特定应用电路才可以实现用户的应用要求。一个典型的例子就是：在发生一些事件之后，计数器或状态机会存储移位的数据。

移位寄存器的常见用法是在时钟驱动下移入或移出八位数据，就像在 SPI 协议中实现的那样。如果您在构建一个通信协议，请检查 Creator 是否已经提供了针对该通信协议的用户组件。

输入/输出连接

本节介绍移位寄存器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。

shift_in — 输入*

串行数据移入到移位寄存器 MSB 或 LSB，这取决于移位方向。如果选中了 **Use Shift In**（使用移入）复选框，则显示此终端。

装载 — 输入*

装载输入信号触发待处理数据由 FIFO （之前由 CPU 或 DMA 写入到 FIFO）到移位寄存器的传输。在检测到装载上升沿之后，组件时钟的上升沿将触发传输事件。如果选中了 **Use Load**（使用装载）复选框，则显示此终端。注意，装载脉冲的占空比是任意值，但宽度至少为一个组件时钟周期。在检测到另一个上升沿之前，装载信号至少在一个组件时钟周期内保持低电平状态。

存储 — 输入*

存储输入信号触发当前移位寄存器值到输出 FIFO 的传输。在检测到存储信号上升沿之后，组件时钟的第一个上升沿触发传输事件。注意，负载脉冲的占空比是任意值，但宽度至少为一个组件时钟周期。在发生下一个存储事件之前，存储信号至少在一个组件时钟周期内保持为低电平状态。然后，可使用 ShiftReg_ReadData() API 子程序从 FIFO 中读取数据。如果选中了 **Use Store**（使用存储）复选框，则显示此终端。

复位 — 输入

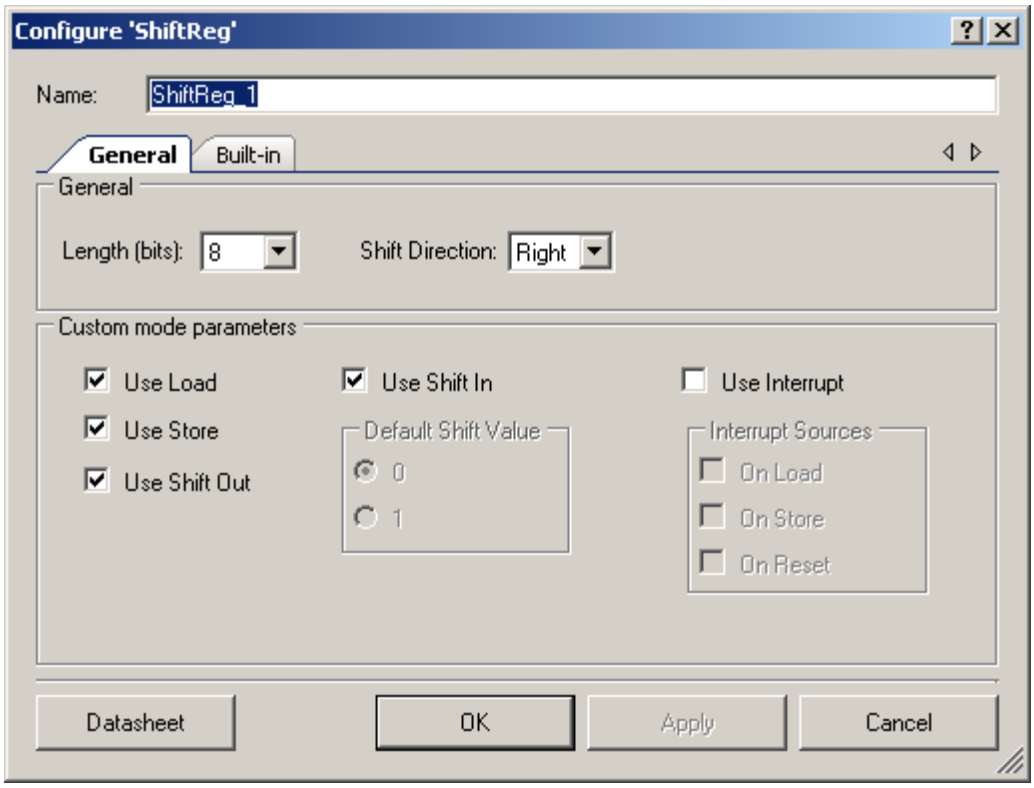
复位输入（高电平有效）将清除整个整个移位寄存器的值。此输入不影响 FIFO 的内容。复位输入与时钟输入同步。复位输入可以不连接外部信号，至于悬空状态。如果复位线上无任何连接，组件会给它分配一个常量逻辑 0。

时钟 — 输入
组件的时钟源。在某些配置中，此信号作为使能信号而非时钟信号。

shift_out — 输出*
从移位寄存器 MSB 或 LSB 中移出串行数据，这取决于移位方向。如果选中了 **Use Shift Out**（使用移出）复选框，则显示此终端。

中断 — 输出*
移位寄存器组件生成的中断信号。基于指定参数生成中断。如果选中了 **Use Interrupt**（使用中断）复选框，则显示此终端。

元件参数
将一个移位寄存器组件拖放到您的设计上，并双击以打开 **Configure**（配置）对话框。



Length (bits)（长度（位））
此参数确定移位寄存器的长度（单位为位）。有效值为 2 到 32 位。默认值为 **8**。

Shift Direction（移位方向）

此参数确定移位方向，即 **Right**（右）或 **Left**（左）。默认值为 **Right**（右）（最低有效位优先）。

Use Load（使用装载）

选择了此选项时，移位寄存器符号上显示装载输入终端。装载信号从内部路由到控制逻辑，以便在检测到装载信号上升正沿之后，在组件时钟的上升沿上将输入 FIFO 中的数据传输到移位寄存器。

如果选定了 **Use Load**（使用装载），将生成 ShiftReg_WriteRegValue()、ShiftReg_ReadRegValue() 和 ShiftReg_GetIntStatus() API 以使用输出 FIFO。*component.h* 文件包含必要的 API 原型和 #define 常量。

如果未选定 **Use Load**（使用装载），组件符号上不显示装载终端，且不生成相关联 API 子程序。

Use Store（使用存储）

选择了此选项时，移位寄存器符号上显示存储输入终端。存储信号从内部路由到控制逻辑，以便在检测到存储信号上升沿之后，在组件时钟的上升沿上将移位寄存器中的当前数据传输到输出 FIFO。

如果选定了 **Use Store**（使用存储），将生成 ShiftReg_WriteRegValue()、ShiftReg_ReadRegValue() 和 ShiftReg_GetIntStatus() API 以使用输出 FIFO。*component.h* 文件包含必要的 API 原型和 #define 常量。

警告

将 ShiftReg_ReadRegValue() API 子程序与 **Use Store**（使用存储）输出 FIFO 功能配合使用时必须谨慎。ShiftReg_ReadRegValue() API 实现将当前移位寄存器 ALU 值传输到输入 FIFO 中，然后从 FIFO 中读取此数据。之前任何使用存储信号在输出 FIFO 中捕获但尚未被应用程序读取的数据将丢失。

如果未选定 **Use Store**（使用存储），组件符号上不显示存储终端，且不生成相关联 API 子程序。

Use Shift Out（使用移出）

此参数决定移位寄存器组件符号是否显示的 shift_out 输出端。默认选择此参数。

Use Shift In（使用移入）

此参数决定移位寄存器组件符号是否显示 shift_in 输入端。默认选择此参数。

Default Shift Value（默认移位值）

此参数允许您定义移位寄存器的输入默认值。仅在未选定 **Use Shift In**（使用移入）参数时才可使用此参数。**Default Shift Value**（默认移位值）参数的有效值为 **0** 和 **1**。

Use Interrupt（使用中断）

如果选择了此参数，中断输出终端将显示在符号上。在这种情况下，可使用由移位寄存器生成的中断。

如果未选择 **Use Interrupt**（使用中断），中断终端将不显示在符号上，同时生成相关联的 API 子程序。

Interrupt Source（中断源）

如果选择了 **Use Interrupt**（使用中断），则启用此参数。中断信号用于指示出现了哪一种指定情况。可启用或禁用中断生成，并指定中断事件的触发源：**On Load**（装载）、**On Store**（存储）或 **On Reset**（复位）。

时钟选择

任何信号都可用作移位寄存器组件时钟输入。在时钟输入信号的上升沿上移位数据。

放置

使用 UDB 阵列资源实现移位寄存器组件。UDB 资源的需求量由工具放置算法分配。

资源

分辨率	数字模块				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/ Counter7 单元	Flash（闪存）	RAM	
8 位	1	1	1	1	554	4	6
16 位	2	1	1	1	630	6	6
24 位	3	1	1	1	668	8	6
32 位	4	1	1	1	668	10	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“ShiftReg_1”分配给指定设计中组件的第一个实例。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“ShiftReg”。

函数	说明
ShiftReg_Start()	启动移位寄存器并启用所有选定的中断
ShiftReg_Stop()	禁用移位寄存器
ShiftReg_EnableInt()	启用移位寄存器中断
ShiftReg_DisableInt()	禁用移位寄存器中断
ShiftReg_SetIntMode()	设置中断的中断源。
ShiftReg_GetIntStatus()	获取移位寄存器中断状态
ShiftReg_WriteRegValue()	直接将值写入移位寄存器
ShiftReg_ReadRegValue()	从移位寄存器中读取当前值
ShiftReg_WriteData()	将数据写入移位寄存器输入 FIFO
ShiftReg_ReadData()	从移位寄存器输出 FIFO 中读取数据
ShiftReg_GetFIFOStatus ()	返回输入 FIFO 或输出 FIFO 的当前状态
ShiftReg_Sleep()	停止组件，并保存所有非保留寄存器
ShiftReg_Wakeup()	恢复所有非保留寄存器，并启动组件
ShiftReg_Init()	初始化或恢复默认移位寄存器配置
ShiftReg_Enable()	启用移位寄存器
ShiftReg_SaveConfig()	保存移位寄存器配置
ShiftReg_RestoreConfig()	恢复移位寄存器配置

全局变量

变量	说明
ShiftReg_initVar	指示是否已初始化移位寄存器。该变量初始化为 0，并在第一次调用 ShiftReg_Start() 时设置为 1。这样，第一次调用 ShiftReg_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，则可在调用 ShiftReg_Start() 或 ShiftReg_Enable() 函数之前调用 ShiftReg_Init() 函数。

void ShiftReg_Start(void)

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

注意，对于 PSoC3 Production 芯片，需要一个组件时钟脉冲，以在调用此函数之后启动组件逻辑。

参数： None（无）

Return Value（返回值）： None（无）

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

void ShiftReg_Stop(void)

说明： 禁用移位寄存器。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void ShiftReg_EnableInt(void)

说明： 启用移位寄存器中断。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void ShiftReg_DisableInt(void)

说明： 禁用移位寄存器中断。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void ShiftReg_SetIntMode(uint8 interruptSource)

说明： 设置中断的中断源。多个源是“或”运算集合。

参数： uint 8 InterruptSource：包含选定中断源的常量的比特字段。
选择多个中断源时，多个源是“或”运算集合。

返回值	说明
ShiftReg_LOAD_INT_EN	启用装载中断
ShiftReg_STORE_INT_EN	启用存储中断
ShiftReg_RESET_INT_EN	启用复位中断

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

uint 8 ShiftReg_GetIntStatus(void)

说明： 获取移位寄存器中断的中断状态。

参数： None（无）

Return Value（返回值）： 包含选定中断源的状态的比特字段。

返回值	说明
ShiftReg_LOAD	已发生装载中断
ShiftReg_STORE	已发生存储中断
ShiftReg_RESET	已发生复位中断

Side Effects（副作用）： 清除中断状态寄存器。

void ShiftReg_WriteRegValue(uint 8/16/32 shiftData)

说明： 直接将值写入移位寄存器。

参数： uint 8/16/32 shiftData：要写入的数据。数据类型由“Shift Register Length”（移位寄存器长度）参数确定。

Return Value（返回值）： None（无）

Side Effects（副作用）： 使用此 API 函数之前必须停止组件。

注意： 一个组件时钟周期之后才可读取写入的值。

uint 8/16/32 ShiftReg_ReadRegValue(void)

说明： 从移位寄存器中返回当前值。

参数： None（无）

Return Value（返回值）： uint 8/16/32 移位寄存器值。数据类型由“Length”（长度）参数确定

Side Effects（副作用）： 清除移位寄存器输出 FIFO。调用 ShiftReg_WriteRegValue() 之后等待至少一个组件时钟周期，然后调用此函数。

警告

将 ShiftReg_ReadRegValue() API 子程序与 **Use Store**（使用存储）输出 FIFO 功能配合使用时必须谨慎。ShiftReg_ReadRegValue() API 实现将当前移位寄存器 ALU 值传输到输入 FIFO 中，然后从 FIFO 中读取此数据。之前任何使用存储信号在输出 FIFO 中捕获，但尚未被应用程序读取的数据将丢失。

cystatus ShiftReg_WriteData(uint8/16/32 shiftData)

说明： 将数据写入移位寄存器输入 FIFO。在装载输入的上升沿上将数据字传输到移位寄存器中

参数： uint 8/16/32 shiftData：要写入的数据。数据类型由“Shift Register Length”（移位寄存器长度）参数确定。

Return Value（返回值）： cystatus：如果 FIFO 为满或在成功操作时为 CYRET_SUCCESS，则返回错误。如果输入 FIFO 已满，则数据将不会被写入 FIFO。

返回值	说明
CYRET_SUCCESS	成功操作
CYRET_INVALID_STATE	输入 FIFO 已满

Side Effects（副作用）： None（无）

uint 8/16/32 ShiftReg_ReadData(void)

说明： 从移位寄存器输出 FIFO 中读取数据。在存储输入的上升沿上将数据字传输到输出 FIFO。

参数： None（无）

Return Value（返回值）： uint 8/16/32：下一个可用数据字。数据类型由“Shift Register Length”（移位寄存器长度）参数确定。

Side Effects（副作用）： None（无）

uint8 ShiftReg_GetFIFOStatus (uint8 fifoId)

说明： 返回输入或输出 FIFO 的当前状态。

参数： uint8 FifoID：确定读取的 FIFO 状态。

FifoID 值	说明
ShiftReg_IN_FIFO	用于读取输入 FIFO 的状态
ShiftReg_OUT_FIFO	用于读取输出 FIFO 的状态

Return Value（返回值）： uint 8：其中一个已定义的 FIFO 状态。

返回值	说明
ShiftReg_RET_FIFO_FULL	FIFO 已满
ShiftReg_RET_FIFO_NOT_FULL	FIFO 未满
ShiftReg_RET_FIFO_EMPTY	FIFO 为空

Side Effects（副作用）： None（无）

void ShiftReg_Sleep(void)

说明： 这是准备组件睡眠的首选子程序。ShiftReg_Sleep() 子程序保存当前组件的状态。然后它调用 ShiftReg_Stop() 函数，并调用 ShiftReg_SaveConfig() 以保存硬件配置。

在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 ShiftReg_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 ShiftReg_Wakeup(void)

说明： 该函数是将组件恢复到调用 ShiftReg_Sleep() 时状态的首选子程序。ShiftReg_Wakeup() 函数调用 ShiftReg_RestoreConfig() 函数以恢复配置。如果组件在调用 ShiftReg_Sleep() 函数前已启用，则 ShiftReg_Wakeup() 函数也将重新启用组件。

注意，对于 PSoC3 Production 芯片，需要一个组件时钟脉冲，以在调用此函数之后返回到正常操作。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： 调用 ShiftReg_Wakeup() 函数前未调用 ShiftReg_Sleep() 或 ShiftReg_SaveConfig() 函数可能会产生意外后果。

void ShiftReg_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 ShiftReg_Init()，因为 ShiftReg_Start() 子程序会调用该函数并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“Configure”（配置）对话框设置为相应的值。

void ShiftReg_Enable(void)

说明： 激活硬件并开始执行组件操作。无需调用 ShiftReg_Enable()，因为 ShiftReg_Start() 子程序会调用该函数，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）： None（无）

void ShiftReg_SaveConfig(void)

说明： 此函数会保存组件配置和非保留寄存器。此函数还将保存当前“Configure”（配置）对话框中定义的或通过相应 API 修改的组件参数值。该函数由 ShiftReg_Sleep() 函数调用。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（无）

void ShiftReg_RestoreConfig(void)

说明： 此函数会恢复组件配置和非保留寄存器。该函数还会将组件参数值恢复为调用 ShiftReg_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 ShiftReg_SaveConfig() 函数之后才可调用此子程序。调用此子程序与调用 ShiftReg_SaveConfig() 函数无关，并将用初始设置覆盖当前设置。

定义

ShiftReg_SR_SIZE — 定义移位寄存器长度（单位为位）。

ShiftReg_USE_INPUT_FIFO — 指示在项目中定义了输入 FIFO。

注意：始终定义输出 FIFO，因为它用于软件捕获。

ShiftReg_FIFOSize — 定义移位寄存器字中输入 FIFO 的大小。移位寄存器字大小由 **Length**（长度）（单位为字节）参数值确定。

ShiftReg_DIRECTION — 定义移位方向（0 = 左向移位、1 = 右向移位）。

固件源代码示例

PSoC Creator 在“查找示例项目”对话框中提供了很多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。

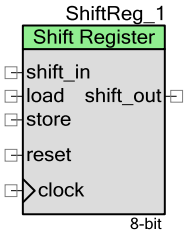
有关更多信息，请参见 PSoC Creator 帮助中的“Find Example Project（查找示例项目）”主题。

功能描述

移位寄存器参数在组件配置上提供极高的灵活性。本节提供对移位寄存器操作的附加说明，以及如何为您的应用自定义组件使用参数。可独立使用移位寄存器，或与其他组件配合使用以创建特定的应用功能。

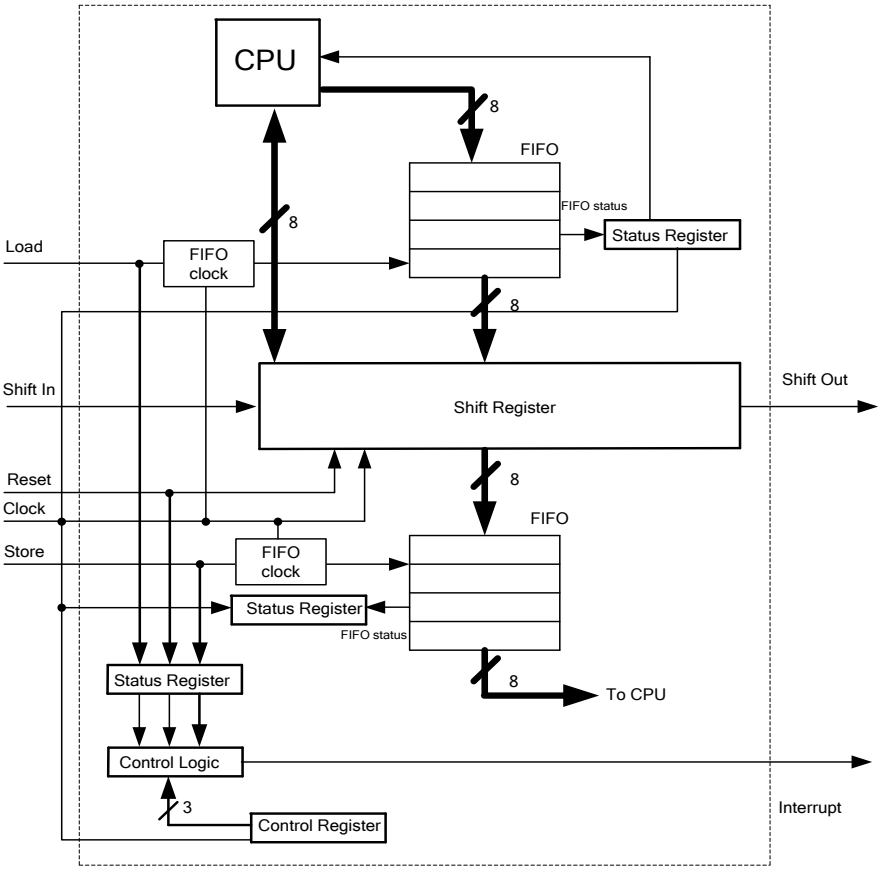
默认配置

移位寄存器组件的默认配置提供与标准 7400 系列逻辑移位寄存器类似的基本并行移位寄存器功能。此功能包括在时钟输入的上升沿将数据在并行寄存器中的同步移入和移出。将串行比特流数据移入 shift_in 终端，并从 shift_out 输出终端中移出。



框图和配置

图 1. 移位寄存器框图



移位寄存器是基于 UDB 的组件，包含一个输入 FIFO (F0)、一个直接移位寄存器（A0 和 A1 以及重复值，用于提供软件捕获）、一个输出 FIFO (F1)、控制寄存器和状态寄存器。

输入 FIFO F0 配置为输入模式。这意味着可由 CPU（使用 ShiftReg_WriteData() API 函数）写入此 FIFO，且可将此值加载到 A0 寄存器以便移位。此函数在每个周期之前使用 ShiftReg_GetFIFOStatus() 函数检查当前 FIFO 状态。

也可通过调用 ShiftReg_WriteRegValue() 将待移位的值直接写入到 A0 寄存器。由于有内部硬件实现，强烈建议在使用 ShiftReg_WriteRegValue() 时停止组件操作（通过使用 ShiftReg_Stop() 函数或停止输入时钟）。否则，在移位操作过程中写入 A0 寄存器将导致写入错误的数据。

加载操作具有硬件限制，即仅可在输入 FIFO 非空时提供加载事件。

要提供移位功能，在以下配置中使用 UDB 数据路径：

State == 100 (4)	移位操作（左或右）
State == 101 (5)	复位 (XOR A0 A0)
State == 110 (6)	负载 A0 <=F0
State == 111 (7)	复位 (XOR A0 A0)

除了存储之外的所有操作都是从数据路径控制存储器中控制的。移位是默认操作 (cs_addr = “000”)。负载输入连接到 cs_addr[1] 线路，复位输入连接到 cs_addr[0]。如果这些线路中的某些线路更改了其电平，则将导致控制存储地址立即更改。在数据路径时钟（在此例中是组件时钟）的上升沿上，将执行对应的操作。负载导致负载值从 F0 更改为 A0。复位命令导致清除 A0。在此情况下，负载值被忽略。

使用两种机制读取移位值：硬件捕获和软件捕获。硬件捕获事件在存储输入的每个正沿上发生。它导致 “Shift Register”（移位寄存器）值被写入到输出 FIFO。可使用 ShiftReg_ReadData() API 函数读取此值。存储输入有一个硬件限制，即存储输入仅在输出 FIFO 未满时才有效。

每次调用 ShiftReg_ReadRegValue() 函数时都会发生软件捕获。此函数读取 A1 值，它复制 A0 的值。此操作减小 A1 值，以便使其自动写入到输出 FIFO F1（因为 F1 被从 A1 配置为软件捕获）。提供软件捕获之前，ShiftReg_ReadRegData() 函数清除输出 FIFO。因此，您应谨慎使用此函数。

注意：使用此函数，A1 中的实际值在写入移位寄存器之后可用于下一个时钟周期。

使用状态寄存器实现中断生成机制。该机制有三位，代表三个中断源。装载、存储和复位。当其中一位的值从 0 更改为 1 时，将自动生成相关状态寄存器输出上的中断脉冲。这三位都处于“读取清除”模式。

第二个状态寄存器用于存储当前输入和输出 FIFO 的状态。所有状态位都处于“粘滞”模式下（读取后不清除）。

当移位寄存器大小超过 8 时，提供数据路径的连锁连接以将 2、3、或 4 个数据路径彼此连接起来，从而实现 16、24 或 32 的组件大小。要实现与数据路径的测量值不符合的移位寄存器大小，使用仿真模型控制的 MSB 进行数据路径的配置。

使用控制寄存器的 CLK_EN 位启动和停止组件。

寄存器

ShiftReg_SR_CONTROL

位	7	6	5	4	3	2	1	0
值	Reserved							clk_en

- clk_en：启用移位寄存器操作

ShiftReg_SR_STATUS

位	7	6	5	4	3	2	1	0
值		F1_not_ empty	F1_full	F0_not_full	F0_empty	reset	store	load

- load：装载状态位
- store：存储状态位
- reset：复位状态位
- F0_empty：输入 FIFO 为空
- F0_not_full：输入 FIFO 未满
- F1_full：输出 FIFO 已满
- F1_not_empty：输出 FIFO 非空

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	–	–	66	MHz
		24 位	–	–	58	MHz
		32 位	–	–	55	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，引脚到同步输入 [2]	1	–	–	STA [3]	ns
t_{PD_ps}	输入路径延迟，引脚到同步输入 [4]	2	–	–	8.5	ns
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_si}	同步输出到输入路径延时（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_{IH}	输入高电平时间	1,2	t_{CY_clock}	–	–	ns
t_{IL}	输入低电平时间	1,2	t_{CY_clock}	–	–	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[5]	单位
f_{CLOCK}	组件时钟频率	8 位	–	–	33	MHz
		16 位	–	–	33	MHz
		24 位	–	–	29	MHz
		32 位	–	–	27	MHz
t_{CLOCKH}	输入时钟高电平时间 [6]	不可用	–	0.5	–	$1/f_{CLOCK}$
t_{clockL}	输入时钟低电平时间 [6]	不可用	–	0.5	–	$1/f_{CLOCK}$
输入						
t_{PD_ps}	输入路径延迟，引脚到同步输入 [7]	1	–	–	STA [8]	ns
t_{PD_ps}	输入路径延迟，引脚到同步输入 [9]	2	–	–	8.5	ns
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_si}	同步输出到输入路径延时（路由）	1,2,3,4	–	–	STA [8]	ns
t_{I_clk}	clockX 与时钟的对齐	1,2,3,4	0	–	1	t_{CY_clock}
t_{IH}	输入高电平时间	1,2	t_{CY_clock}	–	–	ns
t_{IL}	输入低电平时间	1,2	t_{CY_clock}	–	–	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

[5] “所有路由”时序号的最大值是通过将“额定路由”时序号减去因子 2 而计算得出的。如果您的组件实例的运行速度没有超过这些速度，则应无须担心此组件会遇到时序问题。

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

[7] 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。

[8] 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。

[9] 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

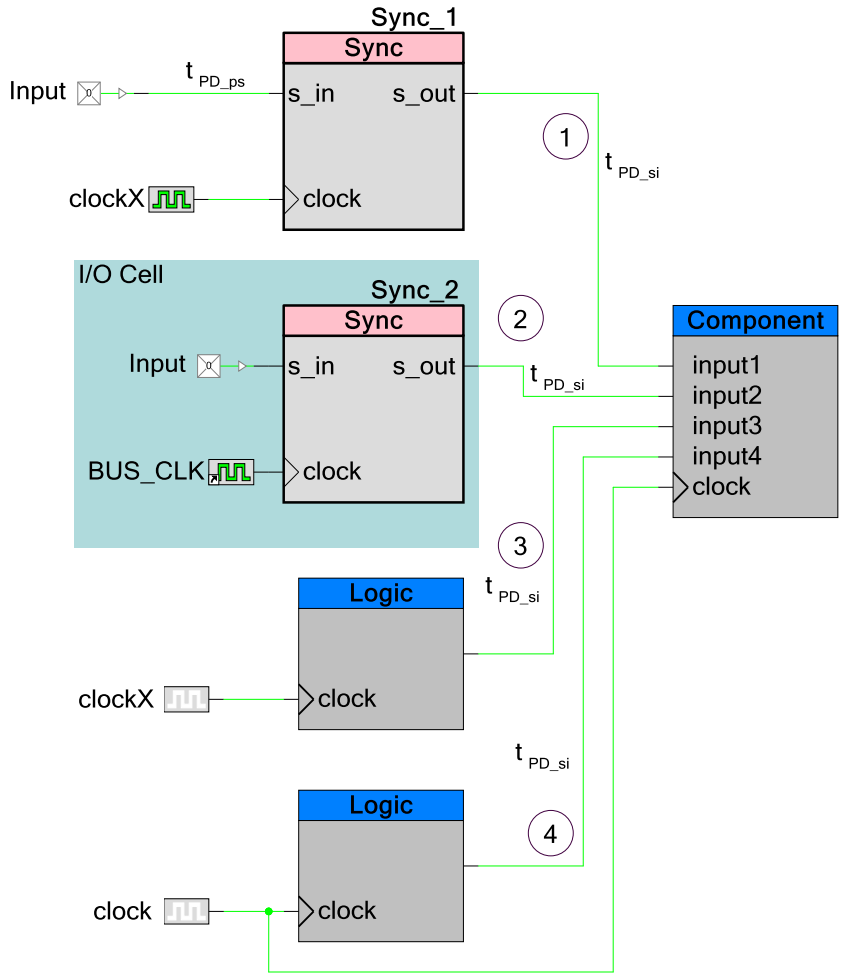
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

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 2 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 2. 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 7
1	clock	master_clock	图 5
1	clock	clockX = 时钟 [10]	图 3
1	clock	clockX > 时钟	图 4
1	clock	clockX < 时钟	图 6
2	master_clock	master_clock	图 7

[10] 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
2	clock	master_clock	图 5
3	master_clock	master_clock	图 12
3	clock	master_clock	图 10
3	clock	clockX = 时钟 [10]	图 8
3	clock	clockX > 时钟	图 9
3	clock	clockX < 时钟	图 11
4	master_clock	master_clock	图 12
4	clock	clock	图 8

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。它还可以等于 master_clock，该时钟生成如图 3、图 4、图 6 和图 7 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 4 和图 7 所示的特性参数。

图 3. 输入配置 1 和 2；同步器时钟频率 = 组件时钟频率（不保证时钟和 clockX 的边沿对齐）

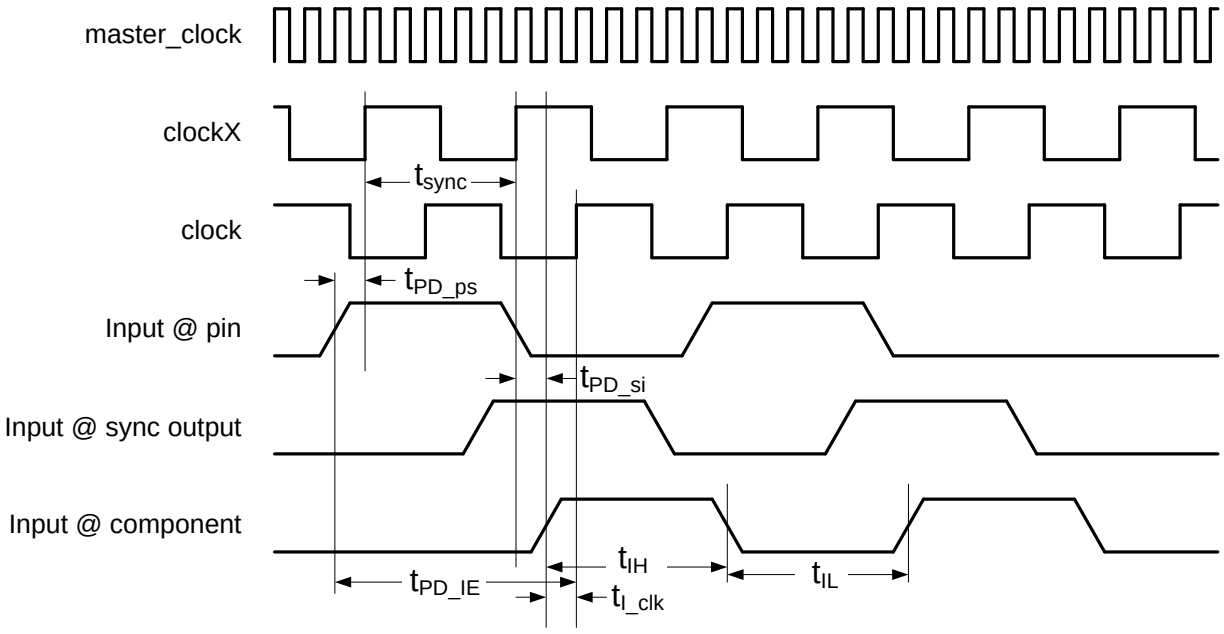


图 4. 输入配置 1 和 2：同步器时钟频率 > 组件时钟频率

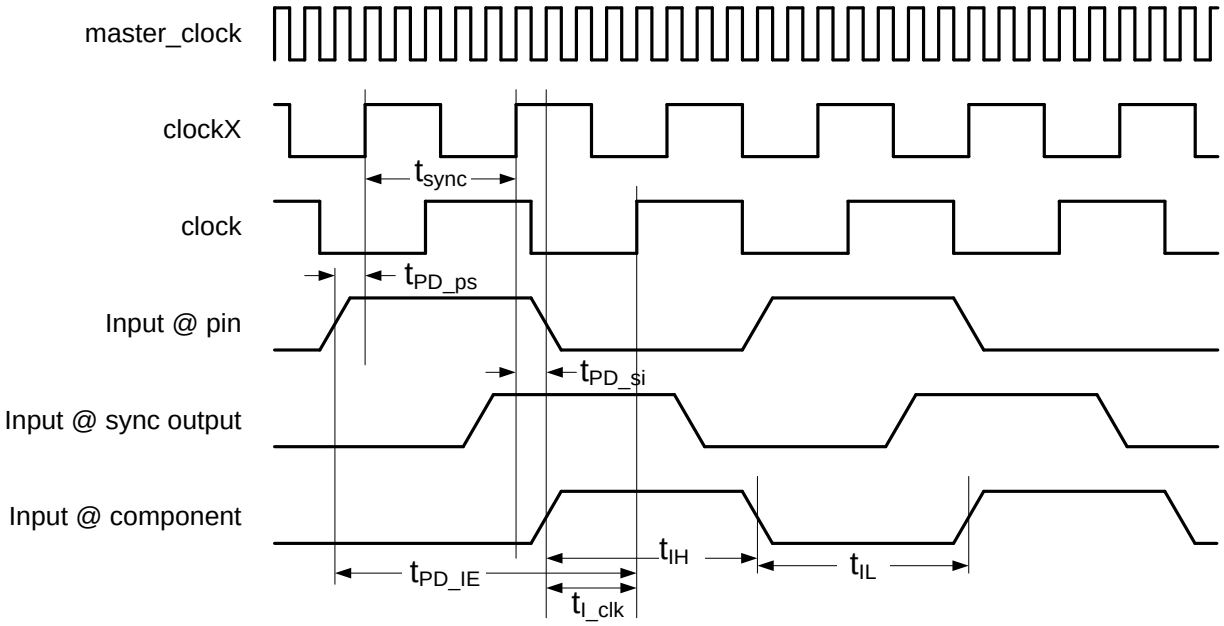


图 5. 输入配置 1 和 2：[同步器时钟频率 = master_clock] > 组件时钟频率

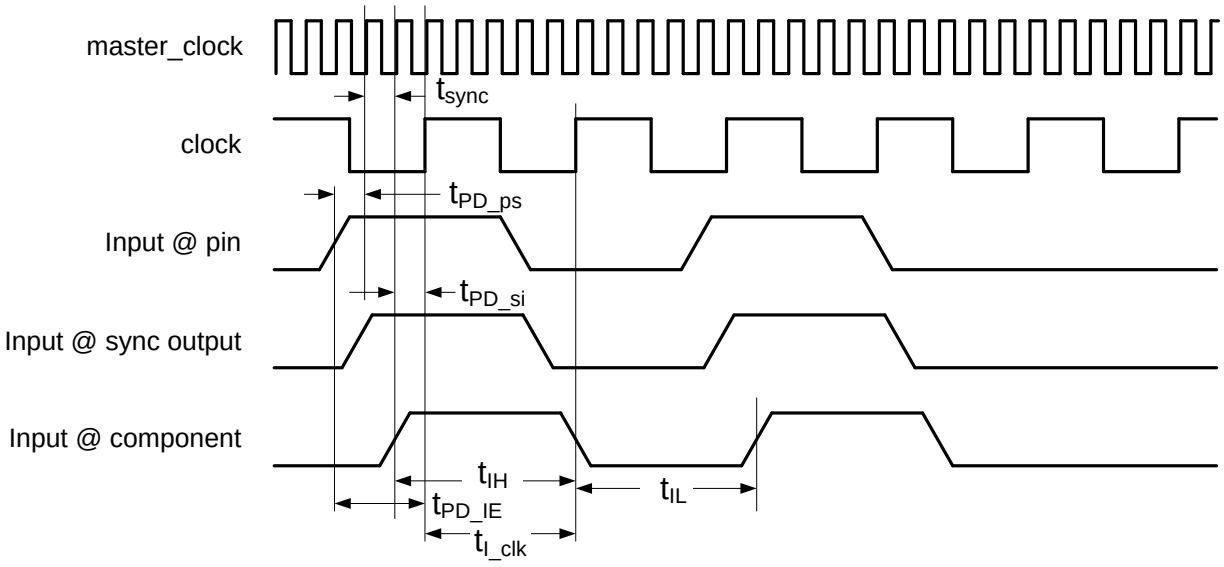


图 6. 输入配置 1：同步器时钟频率 < 组件时钟频率

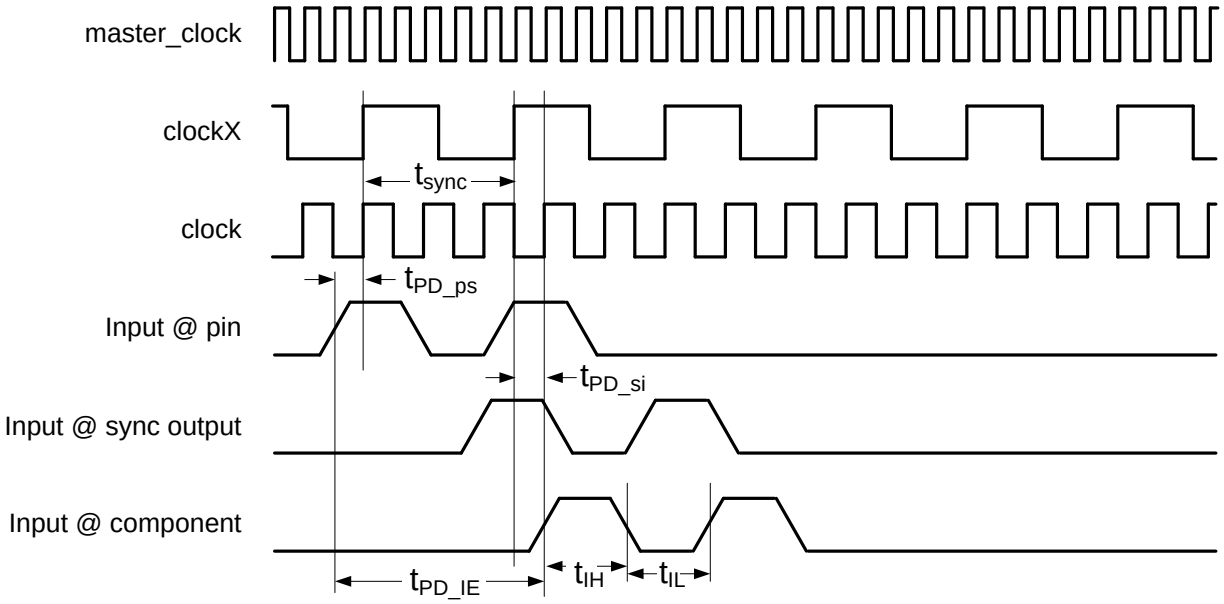
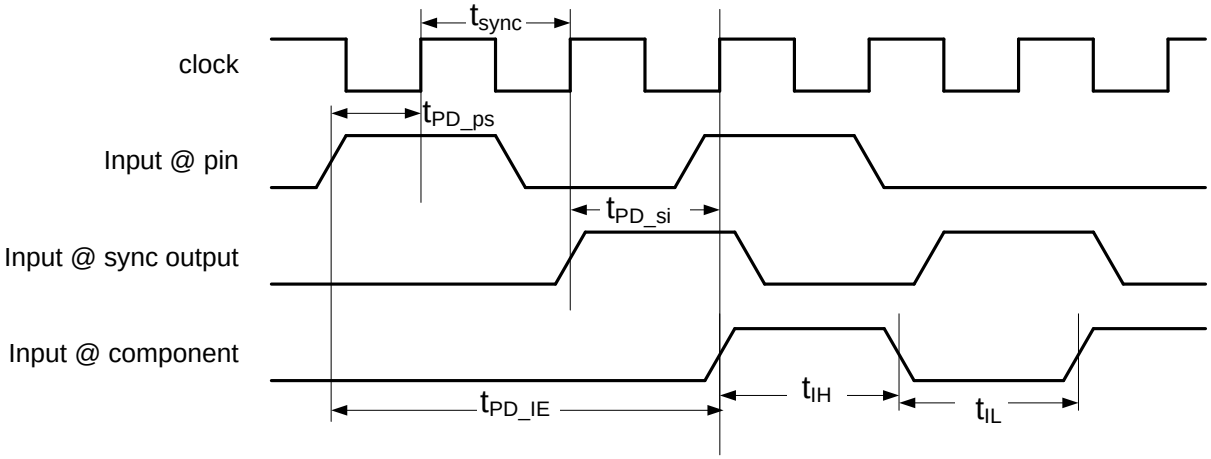


图 7. 输入配置 1 和 2：同步时钟 = 组件时钟 = master_clock



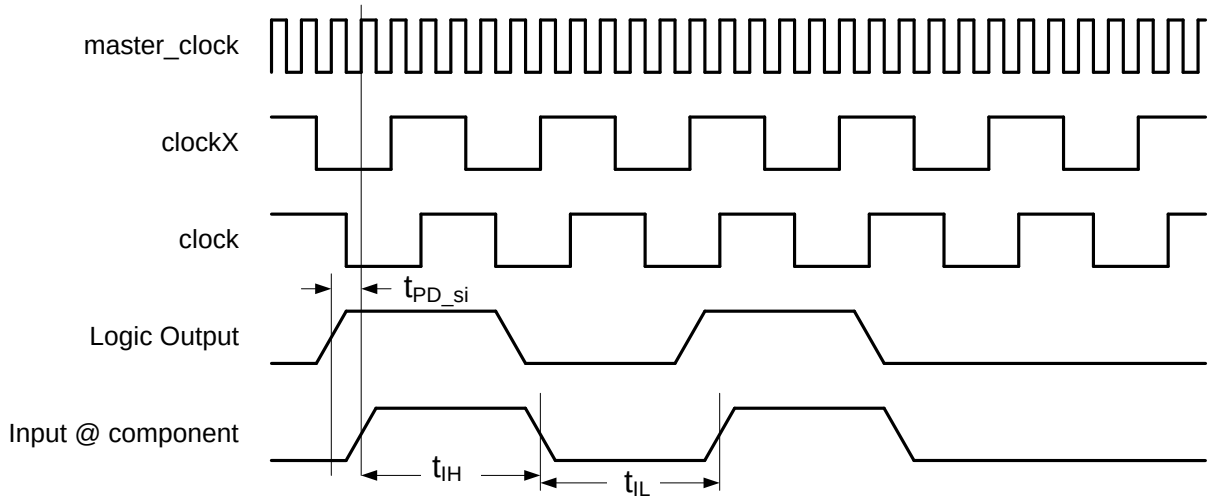
1. 输入由 PSoC 内部逻辑驱动，它基于与组件所使用的时钟不同的时钟同步（所有内部时钟都派生自 master_clock）。

   当表现按此方法配置的输入的特性时，同步器时钟快于、慢于或等于组件时钟，该时钟生成如图 8、图 9 和图 11 所示的特性参数。

2. 输入由 PSoC 内部逻辑驱动，它基于与组件所使用的时钟同步。

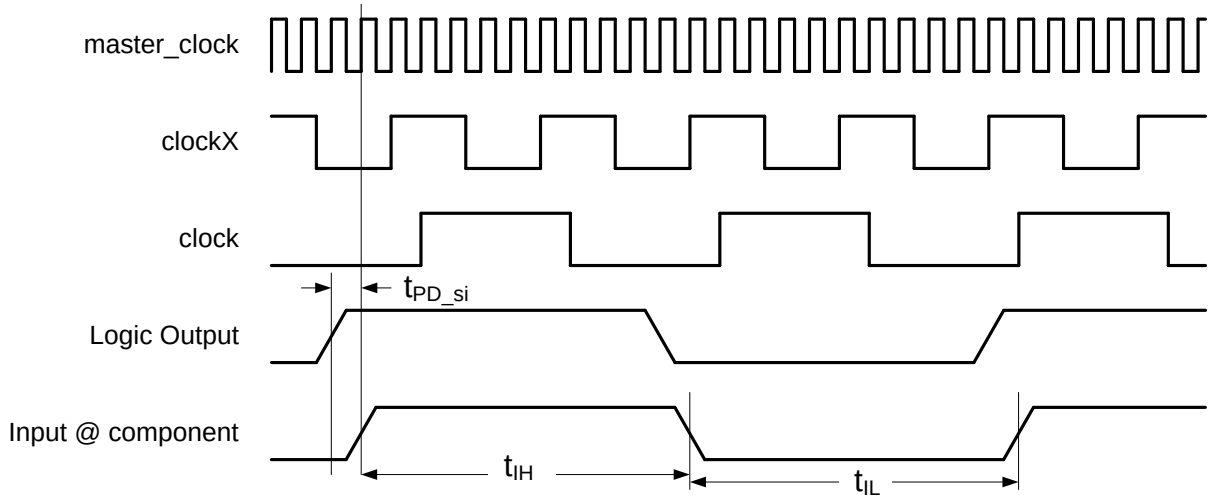
   当表现按此方法配置的输入的特性时，同步器时钟等于组件时钟，该时钟将生成如图 12 所示的特性参数。

图 8. 输入配置 3：同步器时钟频率 = 组件时钟频率（不保证时钟和 clockX 的边沿对齐）



此图表明静态时序分析保持时钟。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} 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 9. 输入配置 3：同步器时钟频率 < 组件时钟频率



与图 8 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 tPD_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 10. 输入配置 3：同步器时钟频率 = master_clock > 组件时钟频率

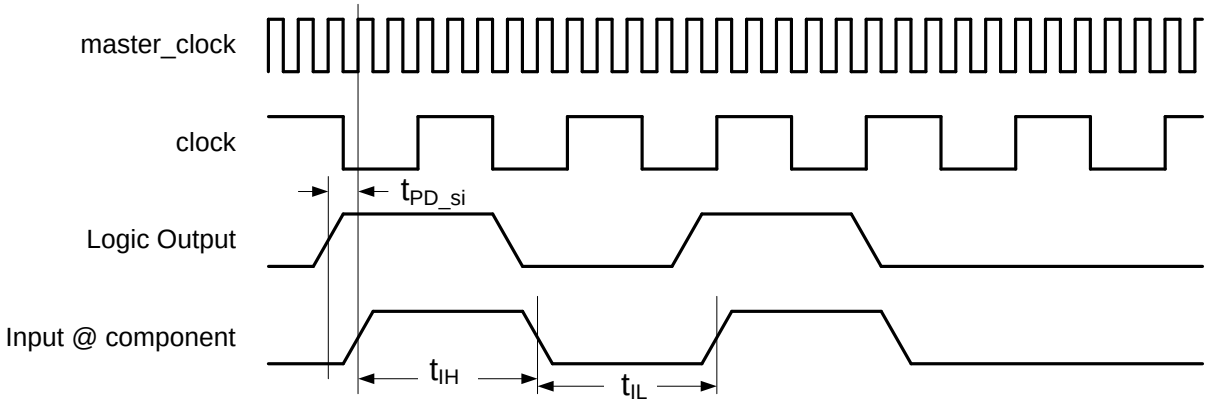
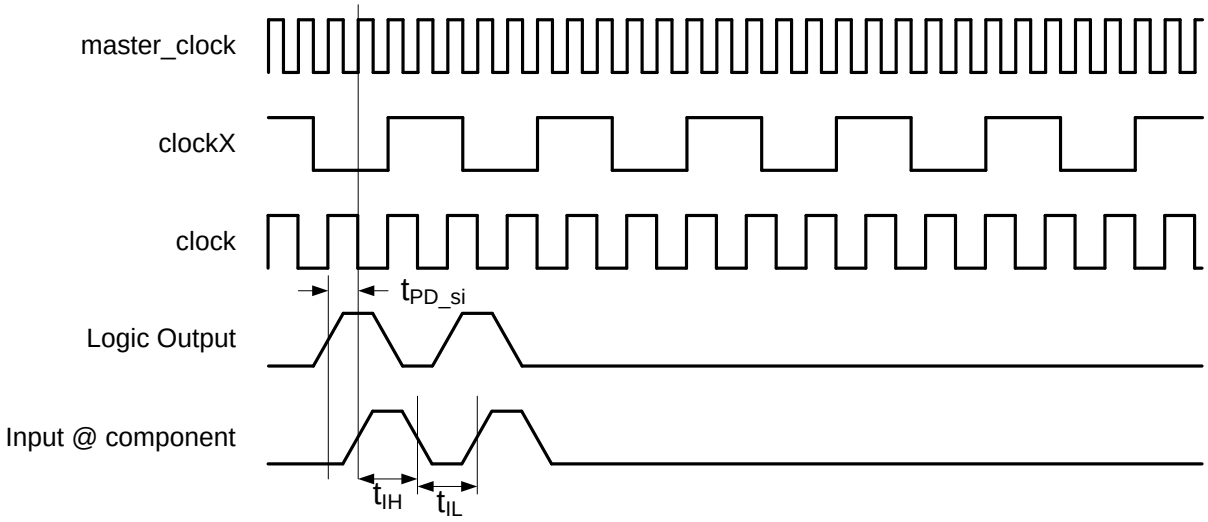
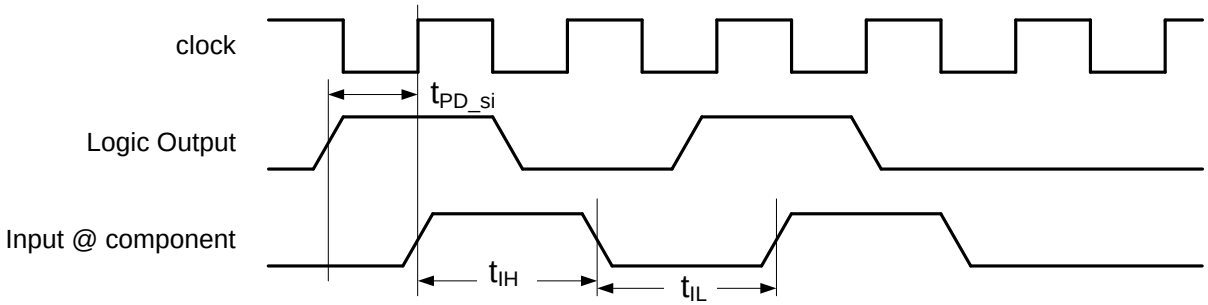


图 11. 输入配置 3：同步器时钟频率 < 组件时钟频率



与图 8 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 tPD_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 12. 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} 将在“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	将装载逻辑实现从电平触发更改为边沿触发。	装载功能根据要求进行了修改。使用电平敏感型装载功能的应用与此版本的组件不兼容。
	数据手册更改 ■ 更新了资源使用表。 ■ 更正了加载信号和存储信号的描述。 ■ 更新了 ShiftReg_Start() 函数和 ShiftReg_Wakeup() API 函数的描述。	
1.60.a	数据手册纠正	
1.60	将 FIFO 模块状态信号重新采样到 DP 时钟。	允许组件针对所有 PSoC 3 和 PSoC 5 芯片使用相同的时序结果。
	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	
1.50	添加了睡眠/唤醒和初始化/启用 API。	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“配置”对话框。	更改了“使用移出”和“使用移入”的位置并更改了“Use interrupt”（使用中断）复选框的默认值，以增强功能。
	更改了 ShiftReg_ReadRegValue() 实现。	从而提供了更快的软件捕获运行速度。
1.20	未使用加载和存储时，将禁用用于选择 FIFO 大小的选项。 更新了“配置”对话框。 删除了未使用参数的生成码。	做出了各种更改，以解决功能并不全面的版本 1.10 的各种问题。

© 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。