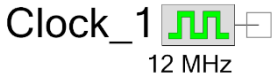
时钟
2.10

特性



- 快速定义新时钟
- 请参见系统或设计范围时钟
- 配置时钟频率容差

概述

时钟组件具有两个重要功能：首先，它支持本地时钟的创建，其次，它支持把系统时钟或者其它更宽设计的时钟连接到你的设计上。有关所有时钟的详细信息，请参见“设计范围资源（DWR）时钟编辑器”这部分。有关更多信息，请参见“PSoC Creator 帮助”的“时钟编辑器”这部分。

定义时钟有多种方法，例如：

- 作为自动选择时钟源的频率
- 作为用户选择时钟源的频率
- 作为分频器和用户选择的时钟源

如果指定了频率，PSoC Creator 将自动选择能够产生最精确结果频率的分频器。若允许，PSoC Creator 还检测所有系统和设计范围时钟，并选择能够产生最精确结果频率的源和分频器对。

外观

时钟组件波形符号的颜色随时钟域的变化而变化（参见“DWR 时钟编辑器”），如下所示：

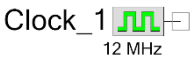
- 数字 — 波形颜色与数字连线的颜色相同，均为黑色外形。
- 模拟 — 波形颜色与模拟连线的颜色相同，均为黑色外形。
- 中间 — 波形颜色为白色，无外形。

输入/输出连接

本节介绍时钟的各种输入和输出连接。I/O 列表中的星号（*）表示：在 I/O 说明部分中所列出的情况下，该 I/O 可能不可见。

时钟 – 输出

时钟具有标准的输出终止信号，可以通过此终止信号访问时钟信号。



数字域 – 输出*

如果选择**强制时钟充当模拟时钟**，此可选输出可以接入模拟时钟的数据域输出。在 **Configure**（配置）对话框中，使用 **Advanced**（高级）选项卡中的选项使能此输出。



组件参数

将时钟拖入设计中，双击时钟，打开 **Component**（配置）对话框。

注意：对于您添加到设计中的任何本地时钟，DWR 时钟编辑器均包含“Start on Reset”（复位启动）选项，此选项默认为使能状态。在某些情况下，您需要以编程的方式来控制时钟，例如，降低功耗。在这种情况下，取消选择“Start on Reset”选项，在代码中插入 Clock_Start()函数。有关更多信息，请参见此数据手册中的应用编程接口部分或“PSoC Creator 帮助”中的“时钟编辑器”部分的内容。

配置时钟选项卡

Configure Clock（配置时钟）选项卡包含 **Clock Type**（时钟类型）和 **Source**（源）参数。根据您的选择，此选项卡将包含其他各种参数，如下图所示：

图 1. Clock Type: New / Source: <Auto>（时钟类型：新建/源：<自动>）

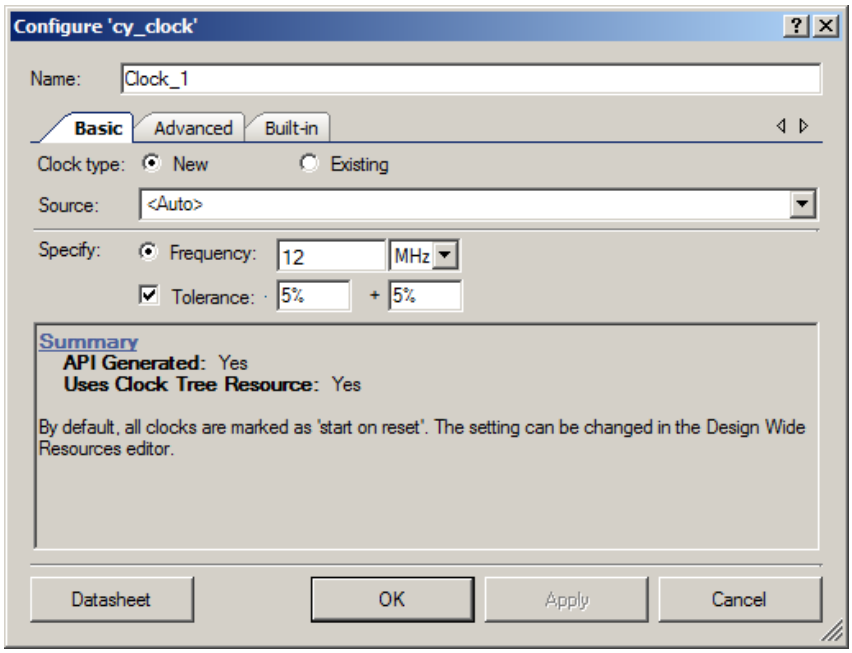


图 2. Clock Type: New / Source：特定时钟（时钟类型：新建/源：特定时钟）

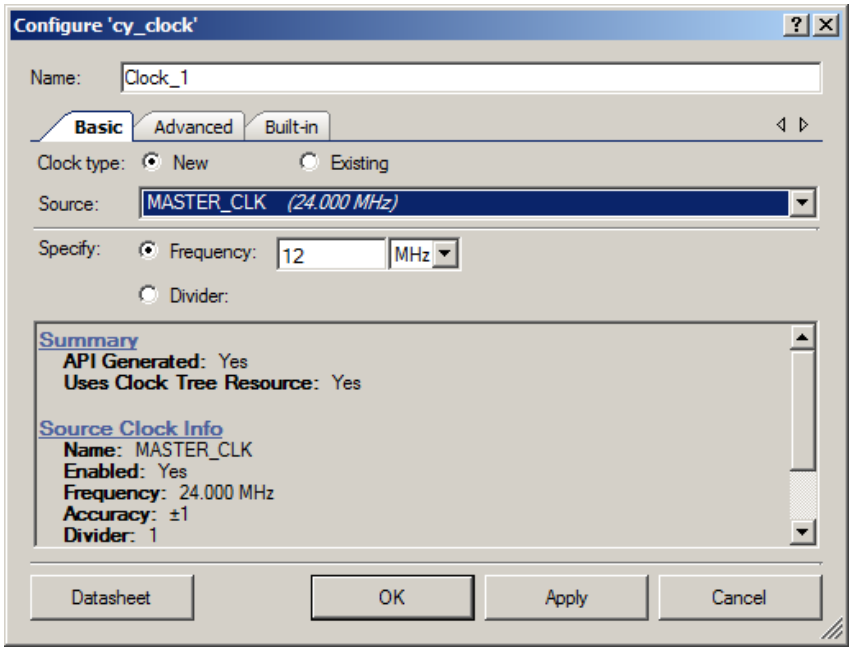
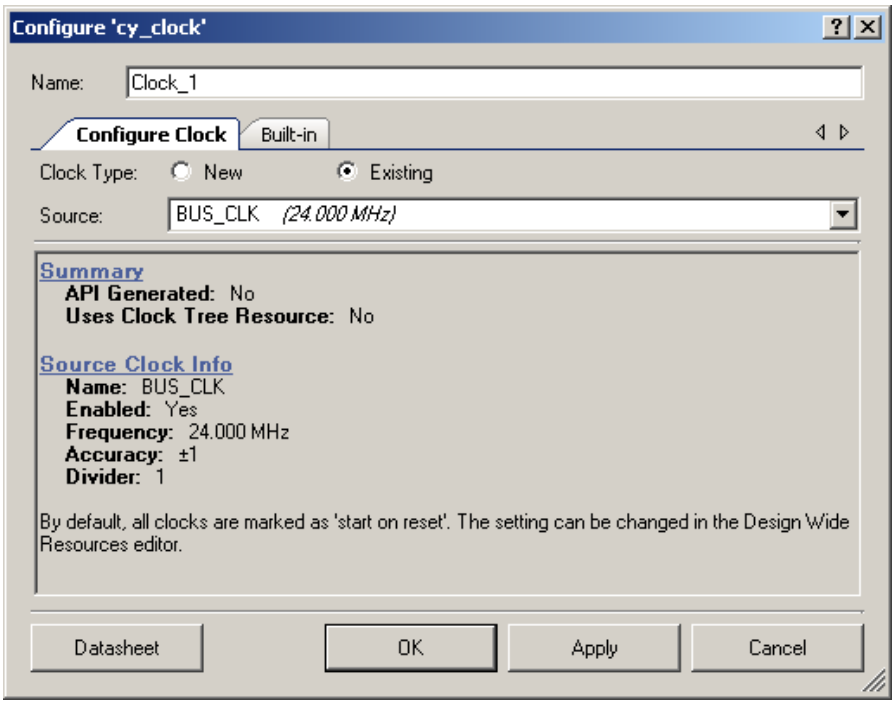


图 3. Clock Type: Existing（时钟类型：现有）



以下各节介绍时钟组件参数：

Clock Type（时钟类型）

有两种时钟类型：**New**（新建）和 **Existing**（现有）。对于新时钟而言，可以指定时钟**源**以使用或支持 PSoC Creator 通过选择**<Auto>（<自动>）**来完成选择。如果选择**<Auto>**，还可以输入特定**频率**和可选**容差**。如果指定一个**源**，可以指定**频率**或选择**分频器**。对于现有时钟而言，仅可以选择时钟**源**。

在原理图中，不同的配置显示不同的时钟符号，如以下示例所示。



在器件中，配置为 **New**（新建）的时钟组件会消耗时钟源，并具有适时生成的 API。在器件中，配置到系统或设计范围时钟的 **Existing**（现有）时钟组件不会消耗任何物理源，而且不具有适时生成的 API。相反，它们使用选定系统或设计范围时钟。

源

如果 PSoC Creator 自动定位可用源时钟，而该源时钟在向下分频时提供最精确的结果频率，则选择**<Auto>**（默认）。带有**<Auto>**源的时钟仅可以输入所需频率。此外，还可以提供一个容差（可选）。

从供应列表中选择一个系统或设计范围时钟，用来强制 PSoC Creator 使用该时钟源。

频率

输入所需频率和单位（默认值 = **12 MHz**）。然后，PSoC Creator 将计算用来创建时钟信号的分频器，其结果尽可能接近于所需频率。

在 PSoC 4 中，如果指定 **Source** 为<Auto>，并且所需频率导致分频器的值大于 65536，PSoC Creator 将自动链接两个 16 位分频器，以获取尽可能接近的分频器，它是两个 16 位数字的乘积。在这种情况下，SetDivider 和 SetFractionalDivider API 不可用。为了灵活修改链接分频时钟的分频器，必须明确指出设计范围时钟的链接：从 DWR 中的时钟编辑器添加“设计范围时钟”，并将时钟组件的**源**指定为刚创建的设计范围时钟。配置这两个时钟的**分频器**值，通过它们的乘积得到预期的分频器值。

容差

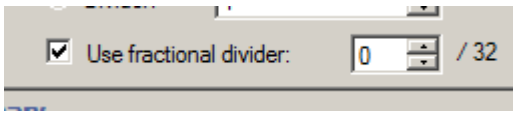
如果将**<Auto>**选作时钟源，则可以输入该时钟所需的容差值（默认值为 5%）。PSoC Creator 将确保结果时钟的精确度介于指定容差范围之内，如果所需时钟无法实现，则会生成警告。时钟容差被指定为百分比。（**注意：**如果输入 ppm 值，该值将被转换为相应的百分比）。如果没有所需容差值，需要取消选中容差旁边的选框，并且不会对该时钟生成警告。

分频器

如果选择特定的 **Source**（源），则可以为 **Divider**（分频器）输入显式值。另外，如果保留 **Source** 设置为**<Auto>**，则 **Divider** 选项不可用（默认）。

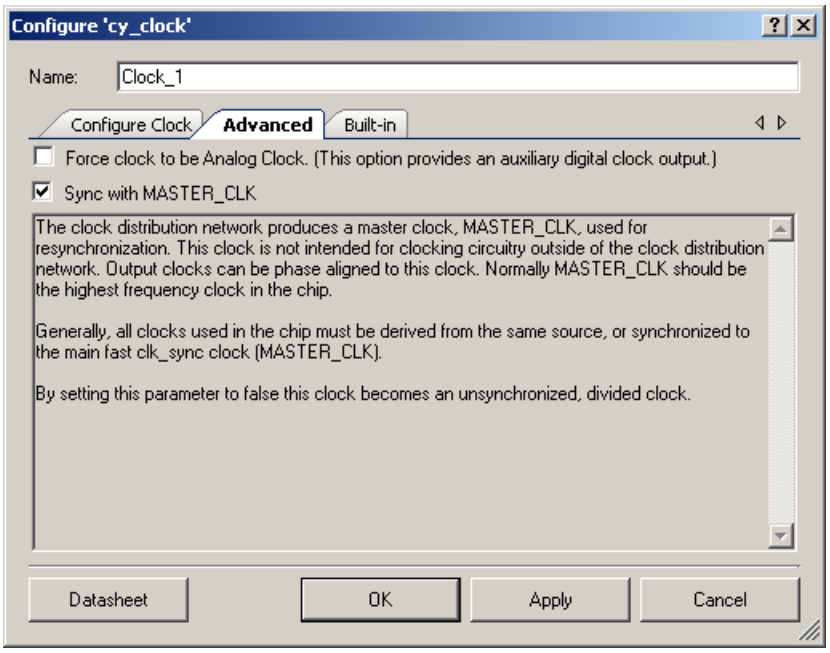
如果选择 **Divider** 选项，则 **Frequency**（频率）选项不可用。

在 PSoC 4 上，可以选择 **Use fractional divider**（使用小数分频器），以允许小数值作为分频器值。如果您指定了某一频率，时钟解算器将使用小数分频器来尝试获得所需频率。如果您指定了某一分频器，您可以输入从 0 到 31 之间的一个小数分频值。



Advanced（高级）选项卡

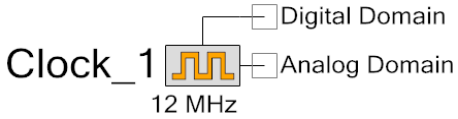
Advanced 选项卡包含两个参数。



注意：PSoC 4 上不存在“Advanced”选项卡。

强制该时钟充当模拟时钟

若勾选此选项（默认值 = 未勾选），此选项添加模拟时钟版本的终止信号，并且，该模拟时钟使用主数字同步时钟作为重新同步时钟。若使用该时钟，则系统强制该时钟进入模拟域；然而，新添加的终止信号位于数字域。



与 MASTER_CLK 同步

若选择该选项（默认值 = 未选中），时钟与 MASTER（主器件）时钟保持同步；否则，该时钟不同步。

应用编程接口

通过应用编程接口（API）子程序，您可以使用软件对组件进行配置。下表列出了每个函数的接口，并对它们进行了说明。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“Clock_1”分配给指定设计中组件的第一个实例。您可以将其重新命名为任何一个符合标识符语法规则的值。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“Clock”（时钟）。

注意：在 **Configure** 对话框中，配置 **Clock Type** 为 **Existing** 的本地时钟均不生成任何 API。

函数	说明
Clock_Start()	使能时钟。
Clock_Stop()	禁用时钟。
Clock_StopBlock()[1]	禁用时钟，然后等待，直到该时钟被禁用时为止。
Clock_StandbyPower()[1]	选择待机（备用活动）操作模式的功耗。
Clock_SetDivider()	设置时钟分频器，并立即重新启动该时钟分频器。
Clock_SetDividerRegister()[2]	设置时钟分频器，并立即重启该时钟分频器（可选）。
Clock_SetDividerValue()	设置时钟分频器，并立即重启该时钟分频器。
Clock_GetDividerRegister()	获得时钟分频器寄存器值。
Clock_SetMode()[1]	设置用于控制时钟操作模式的标志。
Clock_SetModeRegister()[1]	设置用于控制时钟操作模式的标志。
Clock_GetModeRegister()[1]	获得时钟模式寄存器值。
Clock_ClearModeRegister()[1]	清除用于控制时钟操作模式的标志。
Clock_SetSource()[1]	设置时钟源。
Clock_SetSourceRegister()[1]	设置时钟源。
Clock_GetSourceRegister()[1]	获得时钟源。
Clock_SetPhase()[1]	设置模拟时钟的相位延迟（仅为模拟时钟生成）。
Clock_SetPhaseRegister()[1]	设置模拟时钟的相位延迟（仅为模拟时钟生成）。
Clock_SetPhaseValue()[1]	设置模拟时钟的相位延迟（仅为模拟时钟生成）。
Clock_GetPhaseRegister()[1]	获得模拟时钟的相位延迟（仅为模拟时钟生成）。

[1] 不适用于 PSoC 4 器件

[2] PSoC 4 器件不支持“reset”（复位）参数

函数	说明
Clock_SetFractionalDividerRegister()[3]	设置时钟的小数分频器，并立即重启该时钟分频器。
Clock_GetFractionalDividerRegister()[3]	获得小数时钟分频器寄存器值。

void Clock_Start(void)

说明： 启动时钟。

注意：启动时，如果在DWR Clock（DWR时钟）编辑器中使能“Start on Reset”（复位启动）选项，则时钟已经运行。

参数： void

返回值： void

其他影响： 使能时钟。

void Clock_Stop(void)

说明： 停止时钟并立即返回。此API不需要运行源时钟，但在实际禁用硬件之前可以返回。如果调用此函数之后更改时钟设置，则在启动时，时钟可能产生短时脉冲。为避免产生时钟短时脉冲，请使用Clock_StopBlock()函数。

参数： void

返回值： void

其他影响： 禁用时钟。输出将为逻辑0。

void Clock_StopBlock(void)

说明： 返回前，停止时钟并等待硬件实际被禁用。这样确保时钟从未被截断（禁用时钟和API返回前，周期中高逻辑的部分将终止）。注意：源时钟必须保持运行，否则调用这个API将永远不会返回结果，因为不能禁用停止的时钟。

参数： void

返回值： void

其他影响： 禁用时钟。输出将为逻辑0。

注意：仅在 PSoC 3 和 PSoC 5 LP 上支持 Clock_StopBlock() API，且其不会为其他器件生成。

[3] 仅适用于 PSoC 4 器件

void Clock_StandbyPower(uint8 state)

说明： 选择待机（备用活动）操作模式的功耗。

注意：Clock_Start API使能时钟处于备用活动模式，而Clock_Stop和ClockStopBlock API则禁用时钟处于备用活动模式。

如果使能了时钟，但需要在备用活动模式下禁用，则应调用Clock_Start()之后才调用Clock_StandbyPower(0)。如果禁用了时钟，但需要在备用活动模式下使能，则应调用Clock_Stop()之后才调用Clock_StandbyPower(1)。

参数： uint8 state：0值用于在备用活动模式期间禁用时钟，而非零值用于使能时钟。

返回值： void

其他影响： 无

void Clock_SetDivider(uint16 clkDivider)

说明： 修改时钟分频器，由此修改频率。当时钟分频器寄存器设置为零值或从零值更改为其他值时，将暂时禁用该时钟，以便更改模式位。如果调用Clock_SetDivider()时使能时钟，则必须运行源时钟。当前时钟周期将被截断，并新的分频值将立即生效。

该函数与Clock_SetDividerValue之间的区别是该API 必须考虑+1因子。

参数： uint16 clkDivider：分频器寄存器值（0至65,535）。该值不是分频器；对时钟硬件进行clkDivider加1分频。例如，要对时钟进行二分频，则此参数应设置为1。

返回值： void

其他影响： 无

void Clock_SetDividerRegister(uint16 clkDivider, uint8 reset)

说明： 修改时钟分频器，由此修改频率。当时钟分频器寄存器设置为零值或从零值更改为其他值时，将暂时禁用该时钟，以便更改模式位。如果调用Clock_SetDivider()时使能时钟，则必须运行源时钟。

参数： uint16 clkDivider：分频器寄存器值（0至65,535）。该值不是分频器；对时钟硬件进行clkDivider加1分频。例如，要对时钟进行二分频，则此参数应设置为1。

uint8 reset：如果为非零值，重新启动时钟分频器；当前时钟周期将被截断，并新的分频值立即生效。如果为零值，新的分频值将在当前时钟周期结束时生效。

返回值： void

其他影响： 无

void Clock_SetDividerValue(uint16 clkDivider)

说明： 修改时钟分频器，由此修改频率。当时钟分频器寄存器设置为零值或从零值更改为其他值时，将暂时禁用该时钟，以便更改SSS模式位。如果调用Clock_SetDivider()时使能时钟，则必须运行源时钟。当前时钟周期将被截断，并新的分频值将立即生效。

参数： uint16 clkDivider：分频值（1至65535）或零值。如果clkDivider为零值，则将对时钟进行65,536分频。

该函数与Clock_SetDivider()之间的区别是该API不必考虑+1 因子。

返回值： void

其他影响： 无

uint16 Clock_GetDividerRegister(void)

说明： 获得时钟分频器寄存器值。

参数： void

返回值： 时钟的分频值减去1。例如，如果对时钟进行2频时，返回值将为1。

其他影响： 无

void Clock_SetMode(uint8 clkMode)

说明： 设置用于控制时钟操作模式的标志。此函数仅将标志从0改为1；而已经为1的标志保持不变。要清除标志，请使用Clock_ClearModeRegister()函数。该时钟必须在更改模式前禁用。

该API提供的功能与SetModeRegister API提供的功能相同。

参数： uint8 clkMode：位掩码包含要设置的位。 对于PSoC 3，clkMode应是一组进行“OR”（或）逻辑运算后的可选位：

- CYCLK_EARLY：使能初期相位模式。当分频器计数器达到分频值的一半时，输出时钟将出现上升沿。
- CYCLK_DUTY：使能 50%占空比输出。当使能时，输出时钟激活时间约为半个周期。当禁用时，输出时钟激活时间为一个源时钟周期。
- CYCLK_SYNC：使能输出同步化到主器件时钟。这应针对所有同步时钟而使能。

有关设置时钟模式的详细信息，请参见《技术参考手册》。有关具体内容，请参见CLKDIST.DCFG.CFG2寄存器。

返回值： void

其他影响： 无

void Clock_SetModeRegister(uint8 clkMode)

说明： 与Clock_SetMode()相同。设置用于控制时钟操作模式的标志。此函数仅将标志从0改为1；而已经为1的标志保持不变。要清除标志，请使用Clock_ClearModeRegister()函数。该时钟必须在更改模式前禁用。

该API提供的功能与SetMode API提供的功能相同。

参数： uint8 clkMode：位掩码包含要设置的位。它应是一组进行“OR”（或）逻辑运算后的可选位：

- CYCLK_EARLY：使能初期相位模式。当分频器计数器达到分频值的一半时，输出时钟将出现上升沿。
- CYCLK_DUTY：使能 50%占空比输出。当启用时，输出时钟激活时间约为半个周期。当禁用时，输出时钟激活时间为一个源时钟周期。
- CYCLK_SYNC：使能输出同步化到主器件时钟。这应针对所有同步时钟而启用。

有关设置时钟模式的详细信息，请参见《技术参考手册》。有关具体内容，请参见CLKDIST.DCFG.CFG2寄存器。

返回值： void

其他影响： 无

uint8 Clock_GetModeRegister(void)

说明： 获得时钟模式寄存器值。

参数： void

返回值： 位掩码表示已启用模式位。有关模式位的详细信息，请参见Clock_SetModeRegister()和Clock_ClearModeRegister()说明。

其他影响： 无

void Clock_ClearModeRegister(uint8 clkMode)

说明： 清除用于控制时钟操作模式的标志。此函数仅将标志从1改为0；而已经为0的标志保持不变。该时钟必须在更改模式前禁用。

参数： uint8 clkMode：位掩码包含要清除的位。它应是一组进行“OR”（或）逻辑运算后的可选位：

- CYCLK_EARLY：使能初期相位模式。当分频器计数器达到分频值的一半时，输出时钟将出现上升沿。
- CYCLK_DUTY：使能 50%占空比输出。当使能时，输出时钟激活时间约为半个周期。当禁用时，输出时钟激活时间为一个源时钟周期。
- CYCLK_SYNC：使能输出同步化到主器件时钟。这应针对所有同步时钟而使能。

有关设置时钟模式的详细信息，请参见《技术参考手册》。有关具体内容，请参见CLKDIST.DCFG.CFG2寄存器。

返回值： void

其他影响： 无

void Clock_SetSource(uint8 clkSource)

说明： 设置时钟的输入源。该时钟必须在更改源之前禁用。必须运行新和旧时钟源。

该API提供的功能与SetSourceRegister API提供的功能相同。

参数： uint8 clkSource：应是下列输入源之一：

- CYCLK_SRC_SEL_SYNC_DIG：相位延迟主时钟
- CYCLK_SRC_SEL_IMO：内部主振荡器
- CYCLK_SRC_SEL_XTALM：4 至 33 MHz 外部石英振荡器
- CYCLK_SRC_SEL_ILO：内部低速振荡器
- CYCLK_SRC_SEL_PLL：锁相环输出
- CYCLK_SRC_SEL_XTALK：32.768 kHz 外部石英振荡器
- CYCLK_SRC_SEL_DSI_G：DSI 全局输入信号
- CYCLK_SRC_SEL_DSI_D: DSI 数字输入信号
- CYCLK_SRC_SEL_DSI_A：DSI 模拟输入信号

有关时钟源的详细信息，请参见《PSoC技术参考手册》。

返回值： void

其他影响： 无

void Clock_SetSourceRegister(uint8 clkSource)

说明： 与Clock_SetSource()相同设置时钟的输入源。该时钟必须在更改源之前禁用。必须运行新和旧时钟源。

该API提供的功能与SetSource API提供的功能相同。

参数： uint8 clkSource：应是下列输入源之一：

- CYCLK_SRC_SEL_SYNC_DIG：相位延迟主器件时钟
- CYCLK_SRC_SEL_IMO：内部主振荡器
- CYCLK_SRC_SEL_XTALM：4 至 33 MHz 外部石英振荡器
- CYCLK_SRC_SEL_ILO：内部低速振荡器
- CYCLK_SRC_SEL_PLL：锁相环输出
- CYCLK_SRC_SEL_XTALK：32.768 kHz 外部石英振荡器
- CYCLK_SRC_SEL_DSI_G：DSI 全局输入信号
- CYCLK_SRC_SEL_DSI_D/CYCLK_SRC_SEL_DSI_A：DSI 输入信号（两个常量映射到相同值）。

有关时钟源的详细信息，请参见《PSoC技术参考手册》。

返回值： void

其他影响： 无

uint8 Clock_GetSourceRegister(void)

说明： 获得时钟的输入源。

参数： void

返回值： 时钟的输入源。有关详细信息，请参见Clock_SetSourceRegister()。

其他影响： 无

void Clock_SetPhase(uint8 clkPhase)

说明： 设置模拟时钟的相位延迟。此函数仅用于模拟时钟。更改相位延迟之前，必须禁用该时钟以避免产生短时脉冲。

该API提供的功能与SetPhaseRegister API提供的功能相同。

参数： uint8 clkPhase：时钟的相位延迟，以1.0 ns步长递增。clkPhase必须介于1 - 11（含）之间。其他值（包括0）禁用时钟。

clkPhase值	相位延迟
0	禁用时钟
1	0.0 ns
2	1.0 ns
3	2.0 ns
4	3.0 ns
5	4.0 ns
6	5.0 ns
7	6.0 ns
8	7.0 ns
9	8.0 ns
10	9.0 ns
11	10.0 ns
12至15	禁用时钟

返回值： void

其他影响： 无

void Clock_SetPhaseRegister(uint8 clkPhase)

说明： 与Clock_SetPhase()相同。设置模拟时钟的相位延迟。此函数仅用于模拟时钟。更改相位延迟之前，必须禁用该时钟以避免产生短时脉冲。

该API提供的功能与SetPhase API提供的功能相同。

参数： uint8 clkPhase：时钟的相位延迟，以1.0 ns步长递增。clkPhase必须介于1 - 11（含）之间。其他值（包括 0）禁用时钟。

clkPhase值	相位延迟
0	禁用时钟
1	0.0 ns
2	1.0 ns
3	2.0 ns
4	3.0 ns
5	4.0 ns
6	5.0 ns
7	6.0 ns
8	7.0 ns
9	8.0 ns
10	9.0 ns
11	10.0 ns
12至15	禁用时钟

返回值： void

其他影响： 无

void Clock_SetPhaseValue(uint8 clkPhase)

说明： 设置模拟时钟的相位延迟。此函数仅用于模拟时钟。更改相位延迟之前，必须禁用该时钟以避免产生短时脉冲。与Clock_SetPhase()相同，但Clock_SetPhaseValue()添加一个值，然后通过该值调用Clock_SetPhaseRegister()。

参数： uint8 clkPhase：时钟的相位延迟，以1.0 ns步长递增。clkPhase必须介于0 - 10（含）之间。其他值禁用时钟。

clkPhase值	相位延迟
0	0.0 ns
1	1.0 ns
2	2.0 ns
3	3.0 ns
4	4.0 ns
5	5.0 ns
6	6.0 ns
7	7.0 ns
8	8.0 ns
9	9.0 ns
10	10.0 ns
11至15	禁用时钟

返回值： void

其他影响： 无

uint8 Clock_GetPhaseRegister(void)

说明： 获得模拟时钟的相位延迟。此函数仅用于模拟时钟。

参数： void

返回值： 模拟时钟（纳秒）的相位。更多信息，请参见Clock_SetPhaseRegister()的内容。

其他影响： 无

void Clock_SetFractionalDividerRegister(uint16 clkDivider, uint8 fracDivider)

说明： 修改时钟分频器和小数时钟分频器，由此修改频率。整数分频器值为1时，小数分频器将不运行。

参数： uint16 clkDivider：整数分频器寄存器值（0至65,535）。该值不是分频器；对时钟硬件进行clkDivider加1分频。例如，要对时钟进行二分频，则此参数应设置为1。

Uint8 fracDivider：小数分频器寄存器值（0至31）。该值代表以1/32为增量的小数时钟分频值。例如，要对时钟进行3/32分频，则应将该参数设置为3。

返回值： void

其他影响： 无

uint8 Clock_GetFractionalDividerRegister (void)

说明： 获得小数时钟分频器寄存器值。

参数： Void

返回值： 时钟的小数分频值。如果未使用小数时钟分频器，则返回值为0。

其他影响： 无

MISRA 合规性

本节介绍了 MISRA-C:2004 合规性和本组件的偏差情况。定义了两种类型的偏差：

- 项目偏差 — 适用于所有 PSoC Creator 组件的偏差
- 特定偏差 — 仅适用于该组件的偏差

本节介绍了有关组件特定偏差的信息。*《系统参考指南》*的“MISRA 合规性”章节中介绍项目偏差以及有关 MISRA 合规性验证环境的信息。

此时钟组件没有任何特定偏差。

固件源代码示例

在“Find Example Project”对话框中，PSoC Creator 提供了大量的示例项目，包括原理图和代码的。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。要查看通用示例，请打开“Start Page”或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多有关信息，请参考《PSoC Creator 帮助》部分中主题为“查找样例项目”的内容。

资源

资源使用情况因配置和连通性的不同而异。

- 配置为 **Existing** 的时钟组件不消耗芯片上的任何资源。
- 配置为 **New** 的时钟组件消耗单个时钟源。PSoC Creator 自动显示时钟是与数字外设连接还是与模拟外设连接，并在必要时消耗数字时钟或模拟时钟源。

API 存储器大小

根据不同编译程序、器件、所使用的 API 数量以及组件的配置情况，组件所用的存储空间大小也不一样。下表提供了组件配置中所有 API 占用的存储器大小。

通过使用“释放”模式下相应的编译器，可以完成测量操作。在该模式下，存储器的大小得到了优化。有关特定的设计，可分析编译器生成的映射文件以确定存储器的大小。

配置	PSoC 3 （Keil_PK51）		PSoC 4（GCC）		PSoC 5LP（GCC）	
	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节
数字时钟	574	0	104	0	416	0
模拟时钟	589	0	104	0	448	0

组件更改

本节列出了各版本组件的主要更改。

版本	更改内容	更改原因/影响
2.10	对符号的图像进行少量更新。	对“Existing”类型时钟组件提供符号上的最新源时钟信息。

版本	更改内容	更改原因/影响
2.0.a	添加了PSoC 4支持。	修改了GUI和API，以便与PSoC 4一起正确运行。
	添加了Clock_SetFractionalDividerRegister()和Clock_GetFractionalDividerRegister() API。	允许固件设置/获得当前的小数分频器值。
2.0	已添加MISRA合规性章节。	该组件没有任何特定偏差。
	更新了Clock_Start、Clock_Stop和Clock_StopBlock API。	API现在开始/停止处于活动模式和备用活动模式的时钟。这与其他PSoC Creator组件一致。欲了解更多信息，请参考Clock_Standby API说明。
1.70	添加了PSoC 5LP支持。	
	对数据手册进行了少量编辑和更新。	
1.60	更新了Clock_SetDivider()和Clock_SetDividerRegister() API。	确定API以便与PSoC 5一起正确运行。
	更改“数字域 — 输出”的措辞。	
	在数据手册中补充了Clock_Stop()注解。	
1.50.a	在数据手册中补充了Clock_StopBlock()注解，以说明芯片支持的欠缺。	
	对数据手册进行了少量编辑和更新。	
1.50	补充了Clock_StopBlock() API。	此函数用来停止时钟，并等待时钟被禁用。更改设置和重新启用时钟时，要防止短时脉冲。
	补充了Clock_GetPhaseRegister() API（仅限模拟）。	允许固件读取电流相位值。
	补充了Clock_SetPhaseValue() API（仅限模拟）。	此宏返转Clock_SetPhaseRegister()，并自动添加1到相位值，以提供更多直观接口。
	重新命名Clock_SetPhase()为Clock_SetPhaseRegister()（仅限模拟）。	以便与其他名称保持一致。对于兼容性，SetPhase作为宏使用，并与Clock_SetPhaseRegister()的作用相同。
	补充了Clock_GetSourceRegister() API。	允许固件读取电流时钟源。
	重新命名Clock_SetSource()为Clock_SetSourceRegister()。	以便与其他名称保持一致。对于兼容性，SetSource用作宏，并与Clock_SetSourceRegister()的作用相同。
	补充了Clock_GetModeRegister() API。	允许固件读取电流模式标志。
	补充了Clock_SetModeRegister() API。	此函数用来替换Clock_SetMode()。对于兼容性，SetMode用作宏，并与Clock_SetModeRegister()的作用相同。Clock_SetModeRegister()仅将模式标志从0改为1。这样可以防止意外清除其他模式位，例如：SYNC。

版本	更改内容	更改原因/影响
	补充了Clock_ClearModeRegister() API。	此函数类似于Clock_SetModeRegister()，但只将模式标志从1改为0。
	补充了Clock_GetDividerRegister() API。	允许固件读取电流分频器的值。
	补充了Clock_SetDividerRegister() API 。	Clock_SetDivider() API无条件地复位时钟分频器。Clock_SetDividerRegister()支持固件作者控制是否复位了分频器。
	补充了Clock_SetDividerValue() API。	此宏返转Clock_SetDividerRegister()，并自动从分频器减去1，以提供更多直观接口。
	设置Clock_SetDividerRegister()中的SSS。	进行1分频（分频值为0）时，必须设置SSS位，以便绕过分频器。Clock_SetDividerRegister()函数将自动设置/清除SSS，如有必要，可暂时禁用时钟。
	更改寄存器定义。	更新以符合组件代码准则。
	更正了Clock_SetDivider() API文档。	Clock_SetDivider() API文档已指出clkDivider参数应为分频值+ 1。但实际上该参数应该为分频值 - 1。因此，文档错误地指出了“0为clkDivider的无效值”。
	将“与总线同步”改为“与主器件同步”，并与Configure（配置）对话框的工具提示相关联。	更新以符合器件的工作方式。这仅是外观更改。
	补充了参数以使能模拟时钟的数字域输出。	来自硬件模拟时钟的信号可用，但该信号以前未在组件上暴露。
	已向以下函数添加了 `=ReentrantKeil($INSTANCE_NAME . "_...")`： void Clock_Start() void Clock_Stop() void Clock_StopBlock() void Clock_StandbyPower() void Clock_SetDividerRegister() uint16 Clock_GetDividerRegister() void Clock_SetModeRegister() void Clock_ClearModeRegister() uint8 Clock_GetModeRegister() void Clock_SetSourceRegister() uint8 Clock_GetSourceRegister() void Clock_SetPhaseRegister() uint8 Clock_GetPhaseRegister()	如果需要重入，请允许用户将这些API设置为可重入函数。

版本	更改内容	更改原因/影响
1.0.a	将CYCLK_ constants移到 *cydevice.h*/*cydevice_trm.h*。	此时，从所选器件寄存器映射中生成适用于模式和源的CYCLK_ constants。这允许时钟组件不再依赖于器件特定的寄存器值。时钟头文件已包含了*cydevice.h*文件，因此不必更改用户代码。
	在数据手册中，补充CYCLK_ constants的描述。	Clock_SetMode()和Clock_SetSource() API的说明内容当前包含了每个值的描述。

赛普拉斯半导体公司，2013-2016 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可权）（1）在赛普拉斯特软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。