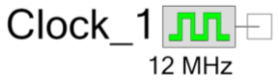
时钟
2.20

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，它支持创建本地时钟，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“PSoC Creator 设计范围资源（DWR）时钟编辑器”这部分。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：

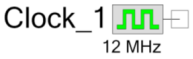
- 数字 — 波形颜色与数字连线的颜色相同，均为黑色外形。
- 模拟 — 波形颜色与模拟连线的颜色相同，均为黑色外形。
- 中间 — 波形颜色为白色，无外形。

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*

如果选中 **Force clock to be Analog Clock**（强制时钟充当模拟时钟）项，通过该可选输出，模拟时钟可访问数据域输出。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，双击它打开 **Component** 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，例如降低功耗，您可能希望以编程的方式来控制时钟。在这种情况下，取消选择 Start on Reset 选项，并在您的代码中插入 Clock_Start()函数。更多有关信息，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。

Basic（基本）选项卡

Basic 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>）

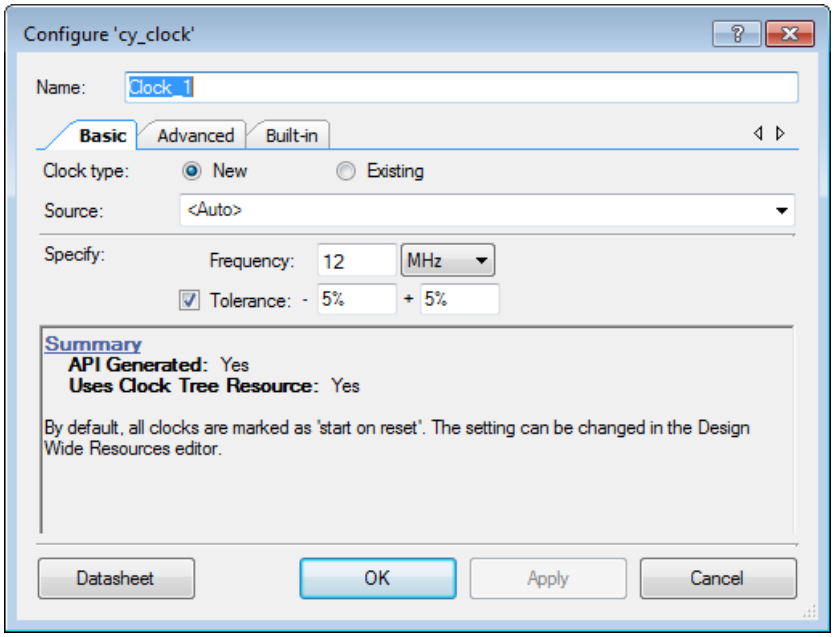


图 2. Clock Type: New / Source: Specific Clock （时钟类型：新建/源：特定时钟）

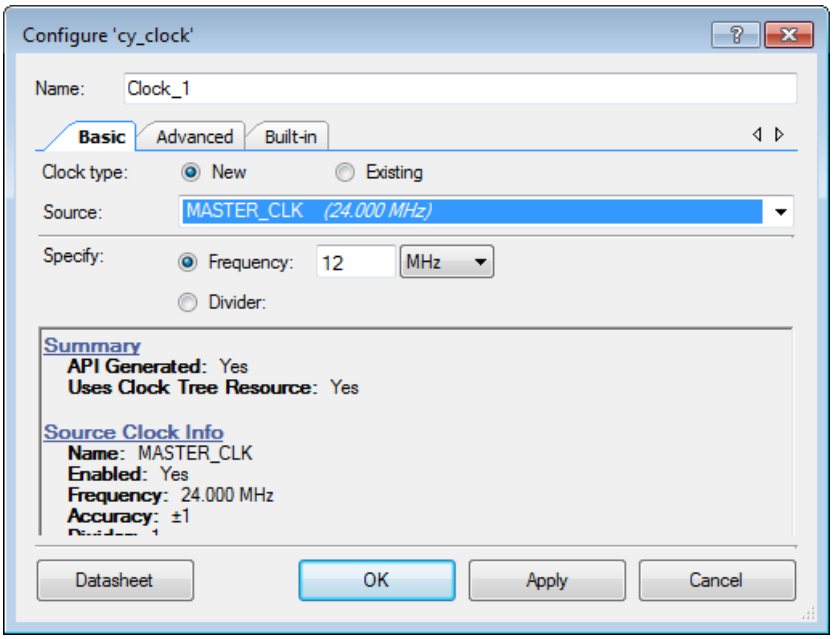


图 3. Clock Type: New （时钟类型：新建）（位于带相位对齐时钟特性的器件上）

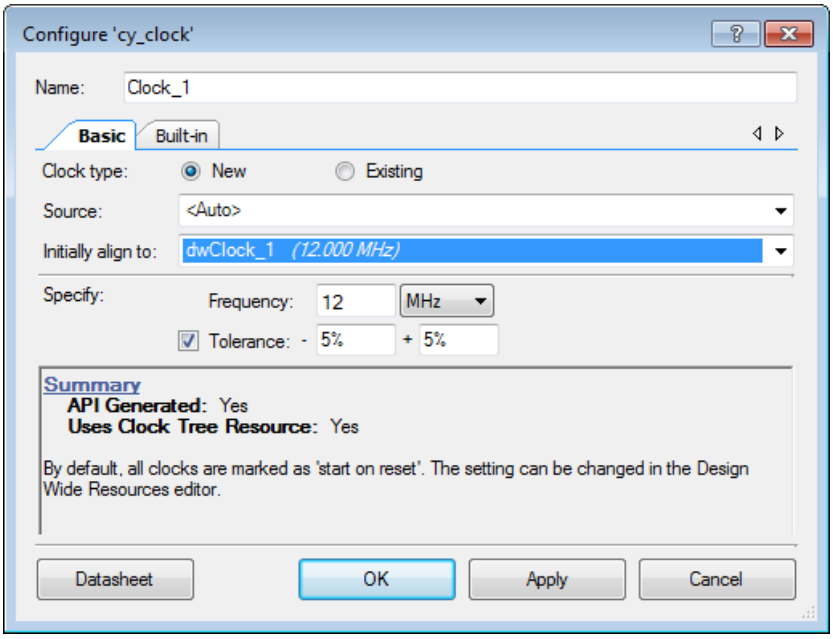
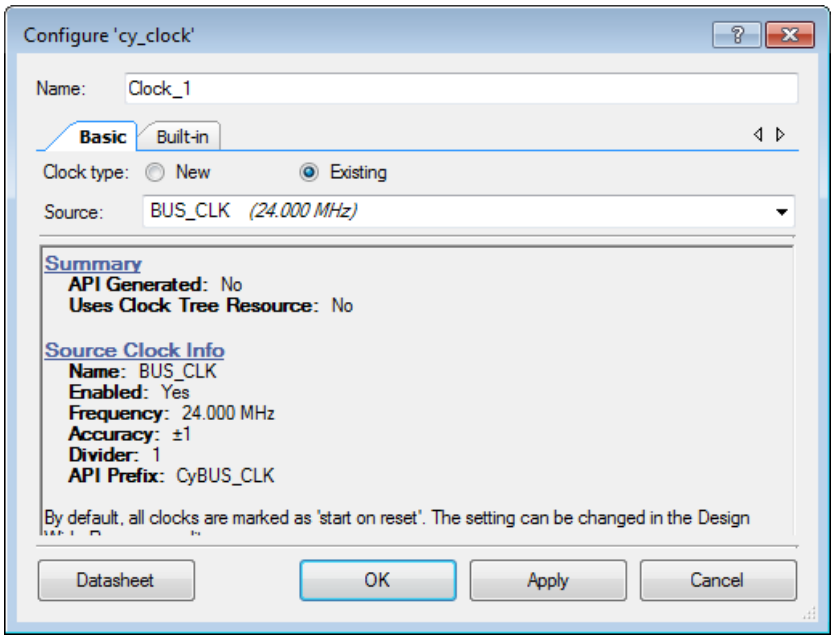


图 4. Clock Type: Existing（时钟类型：现有）

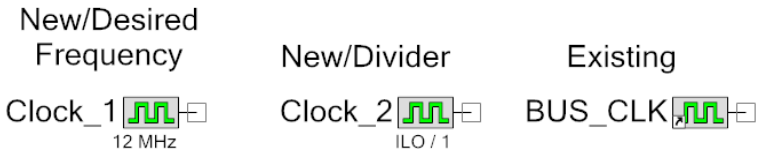


以下各节介绍了时钟组件参数：

时钟类型

有两种时钟类型：**New**（新建）和 **Existing**（现有）。对于新建的时钟，您可指定要使用的时钟**源**，或允许 PSoC Creator 通过选中**<Auto>**项完成选择。如果选中**<Auto>**，仍可以输入特定的 **Frequency**（频率）和可选的 **Tolerance**（容差）。如果指定某个**源**，可以选择 **Frequency**（频率）或选择 **Divider**（分频器）。对于现有的时钟，仅可以选择时钟 **Source**（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。



配置为 **New**（新建）的时钟组件会消耗器件中的时钟源，并为它们生成了 API。配置到系统或设计范围时钟的 **Existing**（现有）时钟组件不会消耗器件中的任何物理资源，而且不生成 API。相反，它们使用选定系统或设计范围时钟。

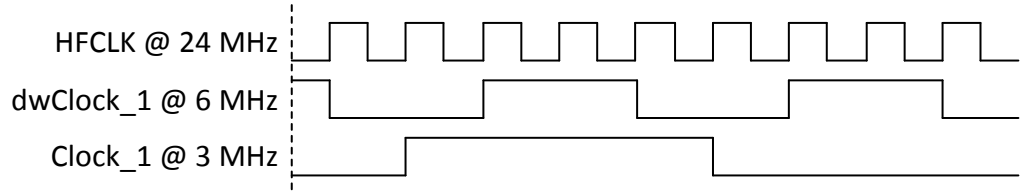
源

如果您需要 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）。

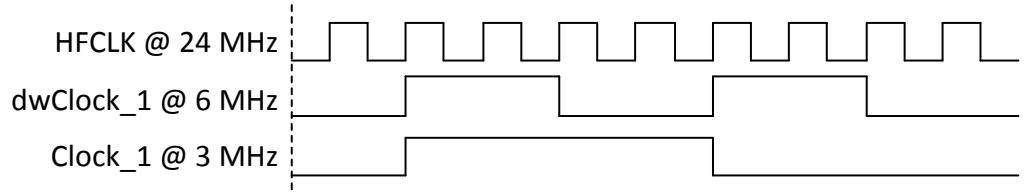
从供应列表中选择一个系统或设计范围时钟，用来强制 PSoC Creator 使用那个时钟作为源。

初始对齐

在带有相位对齐时钟特性的器件上，启动时，通过该参数，您可以指定与该时钟对齐的设计范围时钟。默认情况下，启动时，所有时钟都与 HFCLK 对齐，但指定其他对齐关系是有好处的。例如，刚开始时，6 MHz 时钟和 3 MHz 时钟默认与 HFCLK（所显示的频率为 24 MHz）对齐。这有产生相位范围外的波形的可能性，如下图所示。



通过设置 3 MHz 时钟，使之在启动时与 6 MHz 设计范围时钟对齐，您可确保这些时钟在同一个相位内启动，如下图所示。



注意：如果在 DWR 时钟编辑器内未将时钟设置为“Start on Reset”（复位时启动），则该参数处于无效状态。在这种情况下，需要使用 StartEx() API 来对时钟进行对齐。

频率

输入所需频率和单位（默认值为 **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

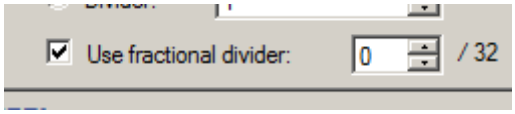
如果将**<Auto>**选作时钟源，则可以输入该时钟所需的容差值（默认值为 5%）。PSoC Creator 将确保结果时钟的精确度介于指定容差范围之内，如果所需时钟无法实现，则会生成警告。时钟容差被指定为百分比。（**注意**：如果输入 ppm 值，该值将被转换为相应的百分比）。如果没有所需容差值，需要取消选中容差旁边的选框，并且不会对该时钟生成警告。

Divider（分频值）

如果选择特定的 **Source**（源），则可以为 **Divider**（分频器）输入明确的值。否则，如果保留 **Source** 设置为**<Auto>**，则 **Divider** 选项不可用（默认）。

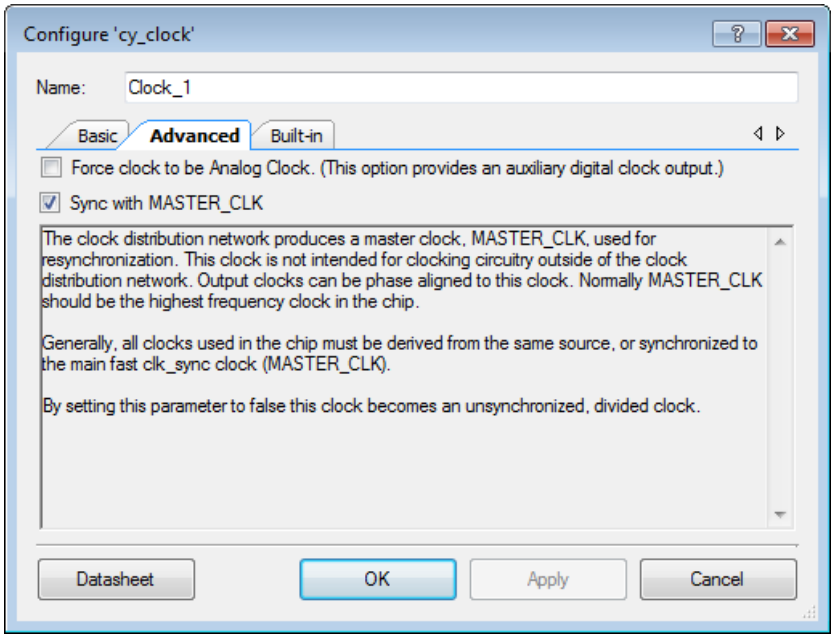
如果选择 **Divider** 选项，则 **Frequency**（频率）选项不可用。

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

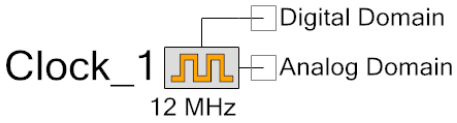
Advanced 选项卡包含两个参数。



注意：PSoC 4 器件不包含 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），此选项为模拟时钟版本添加了一个终端，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，新添加的终端位于数字域。



与 MASTER_CLK 同步

若选择该选项（默认值 = 未选中），时钟与 MASTER（主控）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），您可以使用软件对组件进行配置。下表列出并说明了每个函数的接口。以下各节将更加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、变量和符号常量的前缀。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，如果对本地时钟的 **Clock Type** 设置为 **Existing**，将不生成任何 API。

函数	说明
Clock_Start()	使能时钟。
Clock_StartEx()	启动时钟，且其相位与指定时钟对齐。
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，并立即重启该时钟分频器。
Clock_SetDividerRegister() [2]	设置时钟分频器，并（可选地）立即重启该时钟分频器。
Clock_SetDividerValue()	设置时钟分频器，并立即重启该时钟分频器。

[1] 不适用于 PSoC 4 器件

[2] PSoC 4 器件不支持参数“复位”

函数	说明
Clock_GetDividerRegister()	获取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]	获取时钟模式寄存器值。
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]	获取时钟源。
Clock_SetPhase() [1]	设置模拟时钟的相位延迟（仅为模拟时钟生成该延迟）。
Clock_SetPhaseRegister() [1]	设置模拟时钟的相位延迟（仅为模拟时钟生成该延迟）。
Clock_SetPhaseValue() [1]	设置模拟时钟的相位延迟（仅为模拟时钟生成该延迟）。
Clock_GetPhaseRegister() [1]	获得模拟时钟的相位延迟（仅为模拟时钟生成该延迟）。
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]	获取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。

[3] 仅适用于 PSoC 4 器件

void Clock_StartEx(uint32 alignClkDiv)

说明： 启动时钟，且其相位与指定时钟分频器对齐。该API函数要求目标相位对齐时钟已经运行。因此，正确的流程为先启动目标对齐时钟，然后调用该API函数以与目标时钟对齐。

如果停止并重启目标相位对齐时钟，则将失去相位对齐，应该再一次调用API以重新对齐。

注意： 仅在具有相位对齐时钟特性的PSoC 4器件上，该API才可用。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： uint32 alignClkDiv：所需相位对齐时钟的DIV_ID。比如，如果使用时钟来与Clock_1进行对齐，则Clock_1_DIV_ID值将被传递到该函数。

返回值： Void

其他影响： 无。

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，时钟可能产生跳变。为避免产生时钟信号跳变，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（禁用时钟且API返回前，周期中高电平的部分将终止）。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，因为要停止的时钟不能被关闭。

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，且不会为其他器件生成该函数。

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使能处于备用活动模式中的时钟，Clock_Stop和ClockStopBlock APIs则禁用处于该模式的时钟。

如果已使能时钟，但需要在备用活动模式下禁用它，应在Clock_Start() 之后调用Clock_StandbyPower(0)。如果已禁用时钟，但需要在备用活动模式下使能它，应在Clock_Stop() 之后调用 Clock_StandbyPower(1)。

参数： uint8 state：0值用于在备用活动模式下禁用时钟，而非零值用于使能该时钟。

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被设置为零或更改为非零值时，将暂时禁用该时钟，以便更改模式位。如果调用Clock_SetDivider()时使能该时钟，则必须运行源时钟。当前时钟周期将被截断，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；时钟硬件由clkDivider分频，然后加1。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被设置为零或更改为非零值时，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；时钟硬件由clkDivider分频，然后加1。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，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被设置为零或被更改为非零值时，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，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，则时钟为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： 获取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。必须在更改模式前禁用该时钟。

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。当分频器计时器达到分频值的 1/2 时，将发生输出时钟的上升沿。
- 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：使能主控时钟的输出同步化。应该针对所有同步时钟使能该功能。

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。必须在更改模式前禁用该时钟。

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。当分频器计时器达到分频值的 1/2 时，将发生输出时钟的上升沿。
- 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：使能主控时钟的输出同步化。应该针对所有同步时钟使能该功能。

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

uint8 Clock_GetModeRegister(void)

说明： 获取时钟模式寄存器值。

参数： void

返回值： 位掩码表示已使能模式位。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。必须在更改模式前禁用该时钟。

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。当分频器计时器达到分频值的 1/2 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：使能主控时钟的输出同步化。应该针对所有同步时钟使能该功能。

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。必须在更改源之前禁用该时钟。必须运行新的和旧的时钟源。

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：相位延迟主控时钟
- CYCLK_SRC_SEL_IMO：内部主要振荡器
- 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《 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。必须在更改源之前禁用该时钟。必须运行新的和旧的时钟源。

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：相位延迟主控时钟
- CYCLK_SRC_SEL_IMO：内部主要振荡器
- 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《 技术参考手册》。

返回值： void

其他影响： 无

uint8 Clock_GetSourceRegister(void)

说明： 获取时钟的输入源。

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：时钟相位的延时，递增步长为1.0 ns。clkPhase的范围值必须为1至11（包含1和11在内）。其他值（包括0）将禁用时钟。

clkPhase的值	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
12-15	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：时钟相位的延时，递增步长为1.0 ns。clkPhase的范围值必须为1至11（包含1和11在内）。其他值（包括0）将禁用时钟。

clkPhase的值	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
12-15	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。该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。该函数与Clock_SetPhase()函数相同，区别在于Clock_SetPhaseValue()的值加上1，然后通过该值调用Clock_SetPhaseRegister()。

参数： uint8 clkPhase：时钟相位的延时，递增步长为1.0 ns。clkPhase的范围值必须为0至10（包括0和10在内）。其他值将禁用该时钟。

clkPhase的值	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
11-15	禁用时钟

返回值： void

其他影响： 无

uint8 Clock_GetPhaseRegister(void)

说明： 获取模拟时钟的相位延迟。该函数仅用于模拟时钟。

参数： void

返回值： 模拟时钟的相位（单位为纳秒）。更多信息，请参见Clock_SetPhaseRegister()。

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。小数分频器无法处理按1分频的整数分频器值。

参数： uint16 clkDivider：整数分频器寄存器值（0至65,535）。该值不是分频器；时钟硬件由clkDivider 分频，然后加 1。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）。该值表示增量为1/32的小数时钟分频值。例如，要想按3/32分频时钟，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： 获取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”章节中介绍了项目偏差以及有关 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，包括原理图和示例代码。要获取组件示例，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“Find Example Project”（查找样例项目）的内容。

资源

资源使用情况因配置和连接的不同而异。

- 配置为 **Existing** 的时钟组件不消耗芯片上的任何资源。
- 配置为 **New** 的时钟组件消耗一个单一时钟源。PSoC Creator 自动发现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 存储器使用情况

根据编译器、器件、所使用的 API 数量以及组件的配置情况的不同，组件所用的存储器大小也不一样。下表提供了在某一器件配置中的所有可用的 APIs 使用的存储器大小。

下表中的存储器大小是在将相应编译器设置为 Release（释放）模式并且优化选项为 Size 的情况下测得的。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使用情况。

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

组件更新

本节列出了各版本的主要组件更改内容。

版本	更改说明	更改原因/影响
2.20.a	添加了StartEx()的更多信息。	说明未完整。

版本	更改说明	更改原因/影响
2.20	在定制器中添加了“初始对齐”的参数。 新增StartEx() API。	支持具有相位对齐时钟特性的PSoC 4时钟。
2.10	对符号的图像进行少量更新	在Existing类型的时钟组件符号上提供了最新的源时钟信息。
2.0.a	新增PSoC 4支持	修改了GUI和API，以便与PSoC 4正确运行
	新增Clock_SetFractionalDividerRegister() 和 Clock_GetFractionalDividerRegister() API	允许固件设置/获取当前的小数分频器值
2.0	新增MISRA合规性一节。	该组件没有任何特定偏差。
	更新了Clock_Start、Clock_Stop 和 Clock_StopBlock API	目前，API可在活动模式和备用活动模式下启动/停止时钟。该功能与其他PSoC Creator组件的功能一致。更多信息，请参考Clock_Standby API说明。
1.70	新增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	确定了API，以便与PSoC 5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，以说明硅片支持的欠缺。	
	对数据手册进行了少量编辑和更新	
1.50	新增Clock_StopBlock() API	该函数用来停止时钟，并等待时钟进入禁用状态。这样，在更改设置并重新启动时钟时，可阻止时钟跳变。
	新增Clock_GetPhaseRegister() API（仅限模拟）	允许固件读取电流相位值。
	新增Clock_SetPhaseValue() API（仅限模拟）	该宏返转Clock_SetPhaseRegister()，并自动将1添加到相位值，以提供更加直观的接口。
	将Clock_SetPhase()重命名为 Clock_SetPhaseRegister()（仅限模拟）	重命名的目的在于与其他名称一致。为满足兼容性要求，SetPhase作为宏使用，且其作用与 Clock_SetPhaseRegister()的作用相同。
	新增Clock_GetSourceRegister() API	允许固件读取电流时钟源。
	将Clock_SetSource()重命名为 Clock_SetSourceRegister()	重命名的目的在于与其他名称一致。为满足兼容性的要求，SetSource作为宏使用，且其作用与 Clock_SetSourceRegister()的作用相同。
	新增Clock_GetModeRegister() API	允许固件读取电流模式标志。

版本	更改说明	更改原因/影响
	新增Clock_SetModeRegister() API	该函数用来替换Clock_SetMode()。为满足兼容性要求，SetMode作为宏使用，且其作用与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（如SYNC）。
	新增Clock_ClearModeRegister() API	该函数类似于Clock_SetModeRegister()，但只将模式标志从1改为0。
	新增Clock_GetDividerRegister() API	允许固件读取电流分频器的值。
	新增Clock_SetDividerRegister() API	Clock_SetDivider() API无条件地复位时钟分频器。Clock_SetDividerRegister()允许固件控制是否复位分频器。
	新增Clock_SetDividerValue() API	此宏返转 Clock_SetDividerRegister()，并自动从分频器减去1，以提供更加直观的接口。
	设置SSS in Clock_SetDividerRegister()	进行1分频（分频值为0）时，必须设置SSS位，以便旁路通过分频器。Clock_SetDividerRegister()函数将自动设置/清除SSS，并根据要求暂时禁用时钟。
	更改寄存器定义	更新这些定义，以符合组件编码指南。
	纠正了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“与主控同步”，并更改在Configure对话框中与其相关联的工具提示。	更新该内容，以符合器件的工作方式。该更新操作仅为外观更改。
	新增参数，以从模拟时钟使能数字域输出。	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。

版本	更改说明	更改原因/影响
	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()	如需重入，请允许用户将这些API设置为重新进入。
1.0.a	将CYCLK_ constants移到 *cydevice.h*/*cydevice_trm.h*内。	此时，从所选器件寄存器映像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。
	在数据手册中新增CYCLK_ constants说明。	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，2014-2015。此处，所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路以外，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，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准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：赛普拉斯不针对该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。赛普拉斯保留在不另行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。对于合理预计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，则表示制造商将承担因此类使用而导致的所有风险，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。