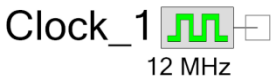
时钟

2.0

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 Help（帮助）" 的“Clock Editor（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 Clock Editor（时钟编辑器）”），如下所示：

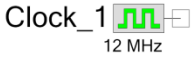
- 数字 - 波形颜色与数字连线的颜色相同，均为黑色外形。
- 数字 - 波形颜色与模拟连线的颜色相同，均为黑色外形。
- 中间 - 波形颜色为白色，无外形。

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 说明中列出的情况下，该 I/O 可能不可见。

时钟 – 输出

时钟具有标准的输出终端，可以通过此终端访问时钟信号。



数据域 – 输出 *

如果选择 **Force clock to be Analog Clock**（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）选项卡中的选项启用此输出。

组件参数

将 Clock（时钟）拖入设计中，双击 Clock（时钟），打开 **Configure**（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，默认情况下，此选项为启用状态。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”（“复位启动”）选项，在代码中插入 Clock_Start() 函数。更多信息，请参见此数据表中的“应用程序编程接口”这部分，或 PSoC Creator 帮助的“时钟编辑器”这部分。

配置时钟选项卡

Configure Clock（配置时钟）选项卡包含**时钟类型**和**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（时钟类型）：新建/源：<自动>

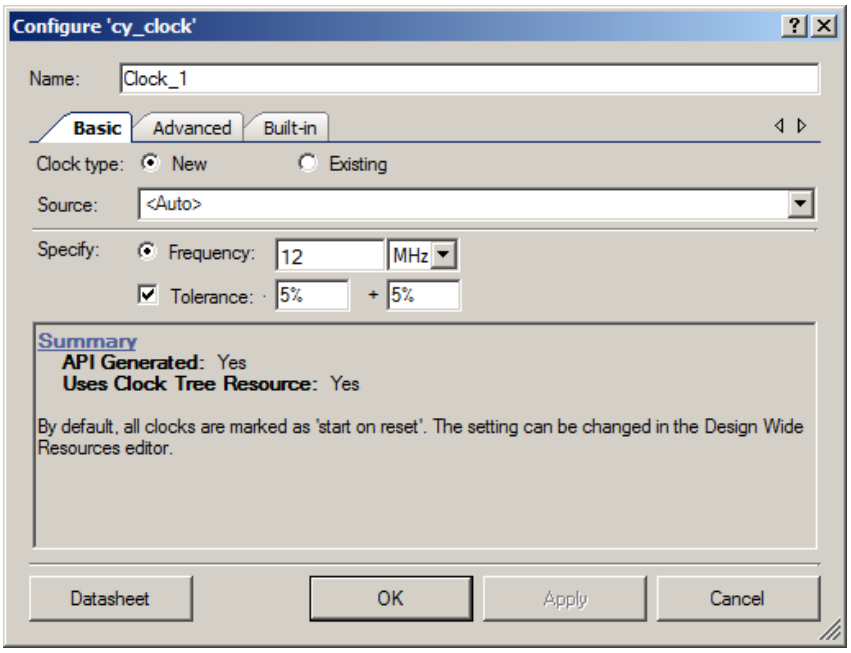


图 2. Clock Type（时钟类型）：新建/源：特定时钟

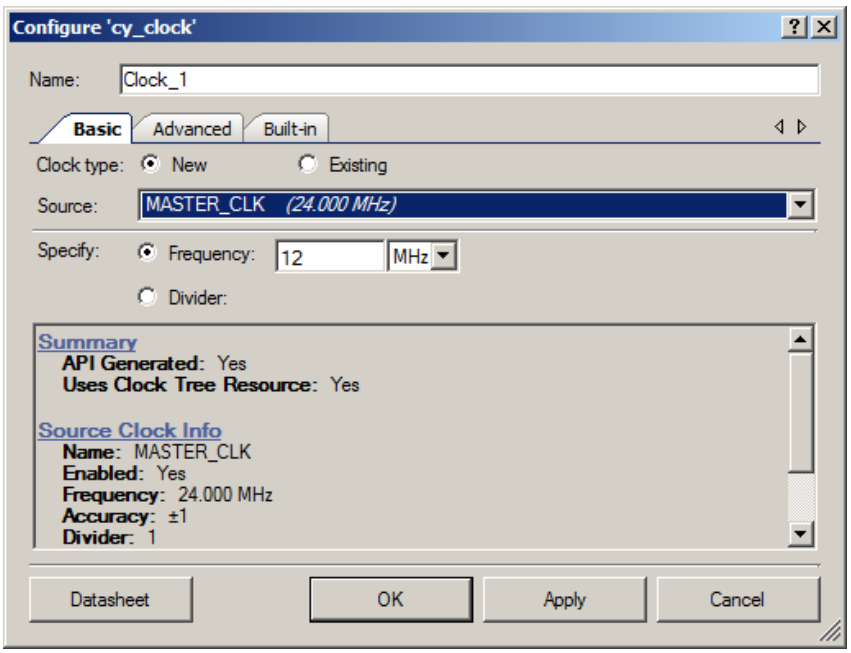
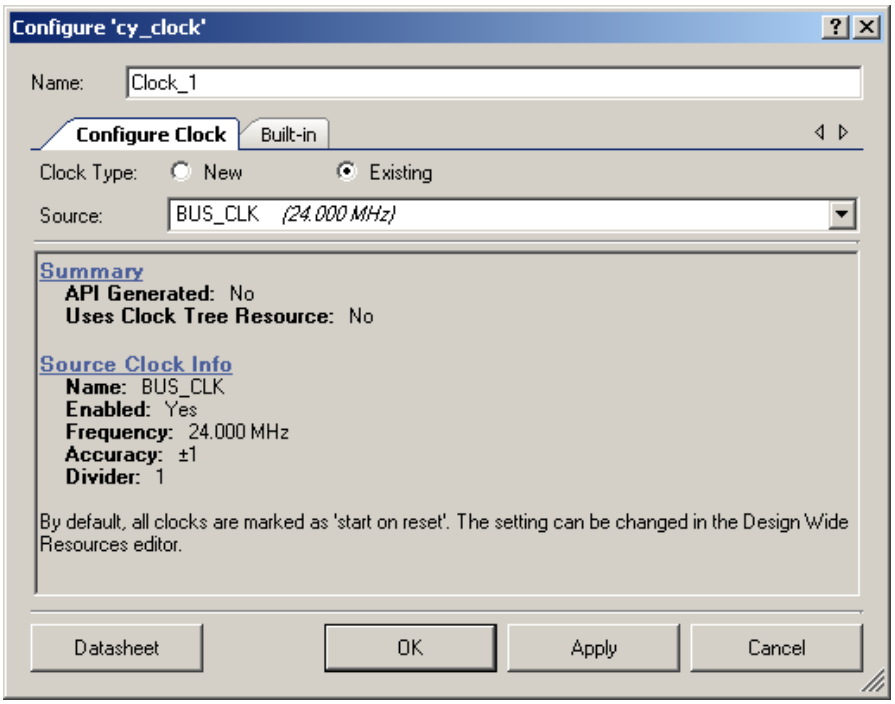


图 3. Clock Type（时钟类型）：现有的



以下各节介绍时钟组件参数：

时钟类型

共有两种时钟类型：**新建时钟**和**现有**时钟。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 **<自动>** 来完成选择。如果选择 **<Auto>** (<自动>)，还可以输入特定 **Frequency**（频率）和可选 **Tolerance**（容差）。如果指定一个 **Source**（源），则既可以指定 **Frequency**（频率），也可以选择 **Divider**（分频器）。对于现有时钟而言，仅可以选择时钟**源**。

在电路图中，不同的配置显示不同的时钟符号，如以下示例所示。

在组件中，配置为 **New**（新建）的时钟组件消耗时钟源，并具有适时生成的 API。在组件中，配置到系统或设计范围时钟的 **Existing**（现有）时钟组件不消耗任何物理源，而且不具有适时生成的 API。相反，他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**<自动>**（默认）。带有**<Auto>**<自动>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 使用作为源的时钟。

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。

容差

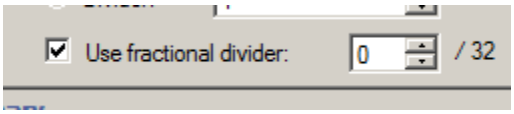
如果将**<Auto>**<自动>选作时钟源，则可以输入该时钟所需的容差值（默认值为 ± 5%）。PSoC Creator 将确保结果时钟的精确度介于指定容差范围之内，如果所需时钟无法实现，则会生成警告。时钟容差可指定为百分比。（**注意**：输入 ppm 将导致所输入的值被转换成相应的百分比值。） 如果不存在所需容差范围，则取消选择容差旁边的复选框，而此时钟不会生成任何警告。

除数

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**（源）设置为**<Auto>**<自动>，则 **Divider**（分频器）选项不可用（默认）。

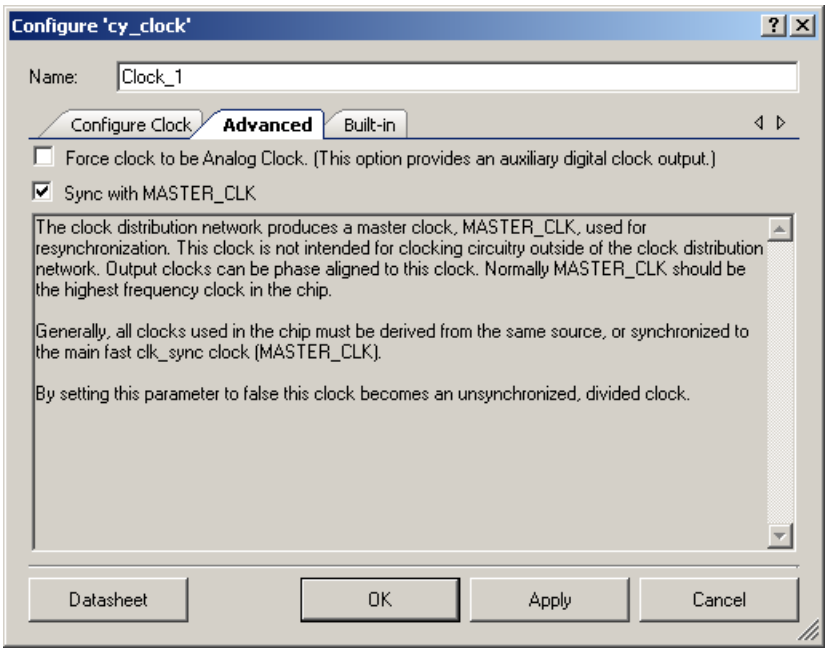
如果选择 **Divider**（分频器）选项，则 **Frequency**（频率）选项不可用。

在 PSoC 4 上，您可以选择 **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

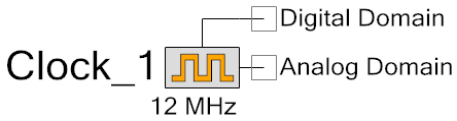
Advanced（高级）选项卡包含两个参数。



注意：PSoC 4 上不存在 Advanced（高级）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），该选项添加模拟时钟版本终端，并且，该模拟时钟使用主数据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“Clock_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“Clock”（时钟）。

Note（注意）：在 **Configure**（配置）对话框中，配置为 **Existing**（现有）**Clock Type**（时钟类型）的本地时钟均不生成任何 API。

函数	说明
Clock_Start()	启用时钟。
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，并立即重启该时钟分频器。
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）。

[1] 不适用于 PSoC 4 组件

[2] PSoC 4 组件不支持参数“重置”

函数	说明
Clock_GetPhaseRegister()1	获得模拟时钟的相位延迟（仅为模拟时钟生成）。
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：启动时，如果在 DWR 时钟编辑器中启用“复位启动”选项，则时钟已经运行。

参数： void

返回值： void

副作用： 启用时钟。

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，但可能在实际禁用硬件之前返回。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

副作用： 禁用时钟。输出将为逻辑 0。

注：使用 PSoC 5 硅片时，具有“MASTER_CLK”源的时钟和分频器 1 将始终保持运行。

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

副作用： 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，且其不会为其他组件生成。

[3] 仅适用于 PSoC 4 组件

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 启用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。

如果时钟已启用，但需要禁用备用活动模式，则应在 Clock_Start() 之后调用 Clock_StandbyPower(0)。如果时钟已禁用，但需要启用备用活动模式，则应在 Clock_Stop() 之后调用 Clock_StandbyPower(1)。

参数： uint8 状态：0 值用于在备用活动模式期间禁用时钟，而非零用于启用时钟。

返回值： void

副作用： 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或被更改为除零以外的值时，将暂时禁用该时钟，以便更改模式位。如果调用 Clock_SetDivider() 时启用时钟，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 (0-65,535)。该值不是分频器；时钟硬件由 clkDivider 分频，然后加 1。例如，要按 2 对时钟进行分频，则此参数应设置为 1。

返回值： void

副作用： 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或被更改为除零以外的值时，将暂时禁用该时钟，以便更改模式位。如果调用 Clock_SetDivider() 时启用时钟，则必须运行源时钟。

参数： uint16 clkDivider：分频器寄存器值 (0-65,535)。该值不是分频器；时钟硬件由 clkDivider 分频，然后加 1。例如，要按 2 对时钟进行分频，则此参数应设置为 1。

uint8 复位：如果为非零值，重启时钟分频器；当前时钟周期将被截断，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

副作用： 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或被更改为除零以外的值时，将暂时禁用该时钟，以便更改模式位。如果调用 Clock_SetDivider() 时启用时钟，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 (1-65535) 或零。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

副作用： 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，时钟设置为 2 频时，返回值为 1。

副作用： 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 和 PSoC 5 而言，clkMode 应是以下可选位集合或组合：

- CYCLK_EARLY：启用初期相位模式。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：启用 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：启用输出同步化到主控时钟。这应针对所有同步化时钟而启用。

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

副作用： 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。它应是下列可选位设置或组合：

- CYCLK_EARLY：启用初期相位模式。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：启用 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：启用输出同步化到主控时钟。这应针对所有同步化时钟而启用。

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

副作用： 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。

副作用： 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。它应是下列可选位设置或组合：

- CYCLK_EARLY：启用初期相位模式。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：启用 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：启用输出同步化到主控时钟。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

副作用： 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：相位延迟主控时钟
- CYCLK_SRC_SEL_IMO：内部主振荡器
- CYCLK_SRC_SEL_XTALM：4-33 MHz 外部晶振
- 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 外部晶振
- 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

副作用： 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：它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：内部主振荡器
- CYCLK_SRC_SEL_XTALM：4-33 MHz 外部晶振
- 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 外部晶振
- 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（两个常量映射到同一值）。

有关时钟源的详细信息，请参见《PSoC 技术参考手册》。

返回值： void

副作用： 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()。

副作用： 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：总计达到时钟相位延迟，以 1.0-ns 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
12-15	禁用时钟

返回值： void

副作用： 无

void Clock_SetPhaseRegister(uint8 clkPhase)

说明： 与 Clock_SetPhase() 相同。设置模拟时钟相位延迟。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：总计达到时钟相位延迟，以 1.0-ns 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
12-15	禁用时钟

返回值： void

副作用： 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：总计达到时钟相位延迟，以 1.0-ns 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
11-15	禁用时钟

返回值： void

副作用： 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()。

副作用： 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 (0-65,535)。该值不是分频器；时钟硬件由 clkDivider 分频，然后加 1。例如，要按 2 对时钟进行分频，则此参数应设置为 1。

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

返回值： void

副作用： 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。

副作用： 无

MISRA 合规性

本节介绍了本组件与 MISRA-C:2004 的合规和偏差情况。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 - 仅适用于此组件的偏差

本节提供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

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的例子项目。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**（筛选选项）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。

资源

资源使用情况因配置和连通性的不同而异。

- 配置为 **Existing**（现有）的时钟组件不消耗芯片上的任何资源。
- 配置为 **New**（新建）的时钟组件消耗单个时钟源。PSoC Creator 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 存储器使用

根据编译器、组件、所用 API 数量和组件配置的不同，组件内存使用会出现较大变化。下表提供指定组件配置中可用的 API 的存储器使用。

已利用释放模式中配置的相关编译器进行了测量，大小采用了优化设定。有关特定的设计，可分析编译器生成的映射文件以确定内存使用情况。

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	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。	此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/停止处于活动和备用活动模式的时钟。这与其他 PSoC Creator 组件一致。查看 Clock_Standby API 说明，了解更多信息。
1.70	添加了 PSoC 5LP 支持	
	对数据表进行了少量编辑和更新	
1.60	更新了 Clock_SetDivider() 和 Clock_SetDividerRegister() APIs	确定 API 以便与 PSoC 5 一起正确运行
	更改“数字域 – 输出”的措辞	
	在数据表中补充了 Clock_Stop() 注解	
1.50.a	在数据表中补充了 Clock_StopBlock() 注解，以说明硅片支持的欠缺。	
	对数据表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。当更改设置和重新启动时钟时，必须要防止时钟跳变。
	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 用作宏，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。

版本	更改说明	更改/影响原因
	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。
	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	The 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 SSS in Clock_SetDividerRegister()	按 1 分频（分频值为 0），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	更新以符合组件编码准则。
	更正了 Clock_SetDivider() API 文档	Clock_SetDivider() API 文档声明 clkDivider 参数分频值应为 + 1。此分频值应为 -1。文档错误地声明 0 对于 clkDivider 而言是无效值。
	将“与总线同步”改为“与主控同步”，并与 Configure（配置）对话框的工具提示相关联。	更新以符合组件的工作方式。这仅是外观更改。
	补充了参数以实现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。

版本	更改说明	更改/影响原因
	将 `=ReentrantKeil($INSTANCE_NAME . "_...")` 添加到下列函数中： 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*。	此时从所选组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组件特定的寄存器值。包含时钟头的 *cydevice.h* 文件，因此不必更改用户代码。
	在数据表中补充 CYCLK_ constants 描述。	Clock_SetMode() 和 Clock_SetSource() API 当前包含每个值的描述。