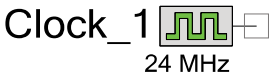
# Clock
1.50

## Features




- Quickly define new clocks
- Refer to system or design-wide clocks
- Configure the clock frequency tolerance

## General Description

The Clock component provides two key features: it provides the means to create local clocks, and it provides the means to connect designs to system and design-wide clocks. All clocks are shown in the Design-Wide Resources (DWR) Clock Editor. For more information, refer to the PSoC Creator Help, Clock Editor section.

Clocks may be defined in a variety of ways, for example:

- as a frequency with an automatically selected source clock
- as a frequency with a user-selected source clock
- as a divider and user-selected source clock

If a desired frequency is specified, PSoC Creator will automatically select a divider that yields the most accurate resulting frequency. If allowed, PSoC Creator will also examine all system and design-wide clocks and select a source and divider pair that yields the most accurate resulting frequency.

### Appearance

The color of the Clock component waveform symbol will change based on the clock's Domain (as shown in the DWR Clock Editor), as follows:

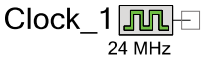
- Digital – the waveform color is the same as a digital wire, with a black outline
- Analog – the waveform color is the same as an analog wire, with a black outline
- Indeterminate – the waveform color is white, with no outline

# Input/Output Connections

This section describes the various input and output connections for the Clock. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

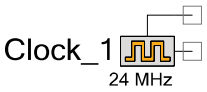
## clock – output

Clocks have a standard output terminal that provides access to the clock signal.




## digital domain – output *

If enabled, this optional output provides access to the digital domain output from an analog clock. Enable this output via the option on the **Advanced** tab of the Configure dialog.




# Component Parameters

Drag a Clock onto your design and double-click it to open the Configure dialog.

**Note** For any local clock you add to your design, the DWR Clock Editor contains a "Start on Reset" option, which is enabled by default. In some cases, such as to reduce power consumption, you may wish to control the clock programmatically. In such cases, deselect the "Start on Reset" option, and insert the Clock_Start() function in your code. Refer to the API section, as well as the Clock Editor section of the PSoC Creator Help, for more details.

## Configure Clock Tab

The **Configure Clock** tab contains the **Clock Type** and **Source** parameters. Based on your selections, this tab will contain various other parameters as shown in the following figures:

**Figure 1 Clock Type: New / Source: <Auto>**

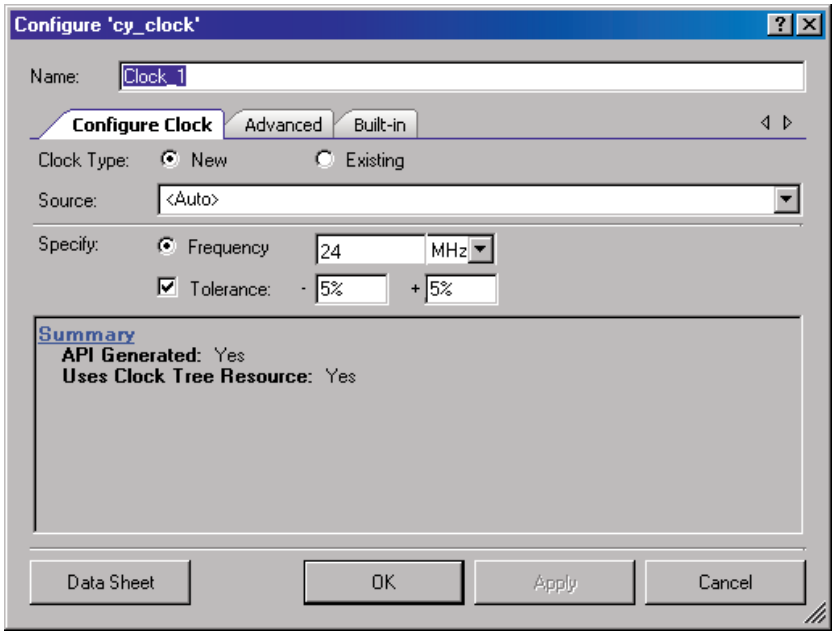



**Figure 2 Clock Type: New / Source: Specific Clock**

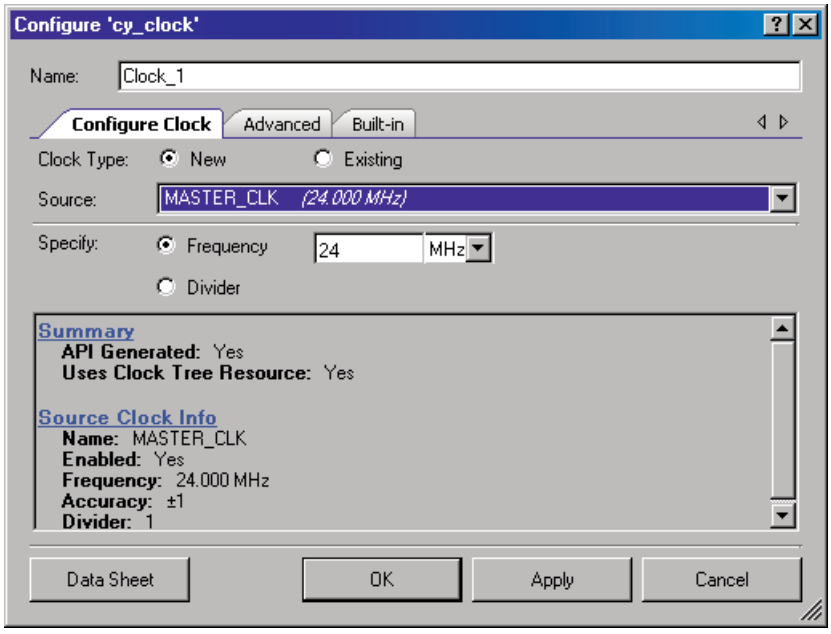
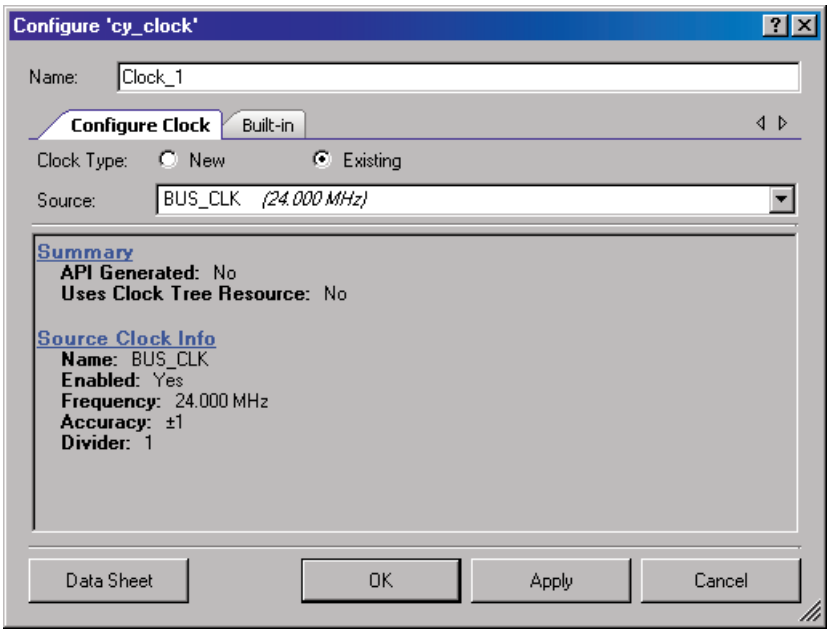


**Figure 3 Clock Type: Existing**

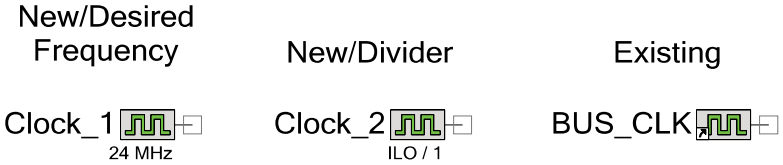



The following sections describe the Clock component parameters:

## Clock Type

There are two clock types: New and Existing. For new clocks, you can specify a clock **Source** to use or allow PSoC Creator to choose by selecting <Auto>. If you select <Auto>, you can also enter a specific **Frequency** and optional **Tolerance**. If you specify a Source, you can either specify a **Frequency** or choose a **Divider**. For existing clocks, you can only select the clock **Source**.

For different configurations, the clock symbol displays differently on the schematic, as shown in the following examples.




Clock components configured as "New" consume clock resources in the device and have APIs generated for them. Clock components configured as "Existing" to a system or design-wide clock do not consume any physical resources on the device and no APIs are generated for them. Instead, they use the selected system or design-wide clock.

## Source

Select <Auto> (default) if PSoC Creator should automatically locate an available source clock that, when divided down, provides the most accurate resulting frequency. Clocks with a source of <Auto> may only enter a desired frequency. A tolerance may also optionally be provided.

Select a system or design-wide clock from the list provided to force PSoC Creator to use that clock as the source.

### Frequency

Enter the desired frequency and units (default = 24 MHz). PSoC Creator will then calculate the divider that will create a clock signal that is as close as possible to the desired frequency.

### Tolerance

If <Auto> is selected as the clock source, you can enter the desired tolerance values for the clock (default is +/- 5%). PSoC Creator will ensure that the accuracy of the resulting clock falls within the given tolerance range or produce a warning if the desired clock is not achievable. Clock tolerances are specified as a percentage. (**Note** Entering ppm will cause the value entered to be converted to the corresponding percent value.) If there is no desired tolerance range, then the check box next to the tolerance can be unchecked and no warning will be generated for this clock.

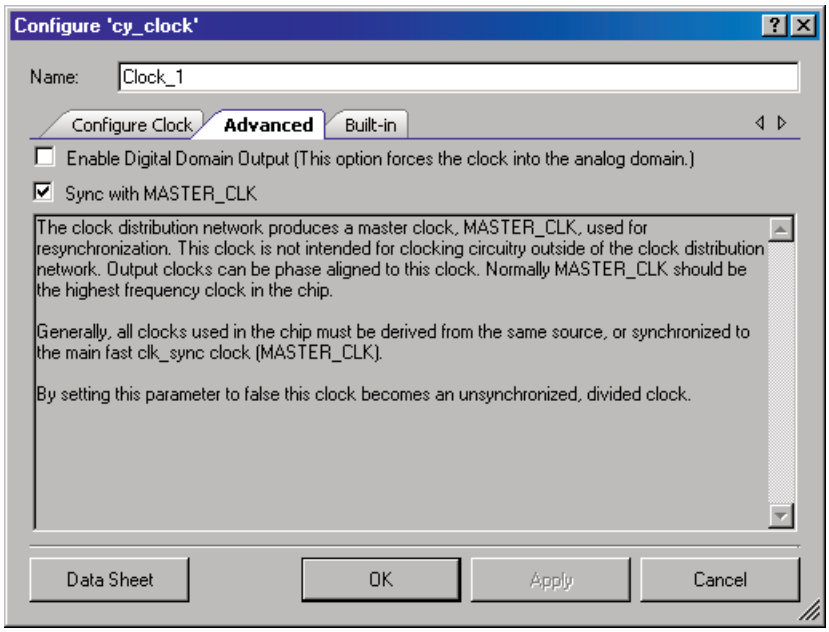
### Divider

If you choose a specific **Source Clock**, you can enter an explicit value for the **Divider**. Otherwise, if you leave the **Source Clock** set to <Auto>, the **Divider** option is not available (default).

If you do select the **Divider** option, then the **Frequency** option is not available.

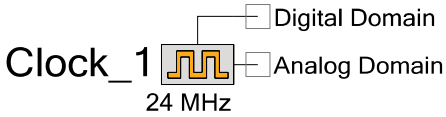
## Advanced Tab

The Advanced tab contains two parameters.



### Enable Digital Domain Output

If checked (default = unchecked), this option adds a terminal for the version of the analog clock that uses the main digital sync clock as the resync clock. If used, this clock is forced into the analog domain; however, the newly added terminal is in the digital domain.




### Sync with MASTER_CLK

If checked (default = checked) the clock is synchronized with the MASTER clock; otherwise, the clock is unsynchronized.

# Placement and Resources

Resource usage varies based on configuration and connectivity.

- Clock components configured as "Existing" do not consume any resource on the chip.
- Clock components configured as "New" consume a single clock resource. PSoC Creator automatically discovers whether the clock connects to digital or analog peripherals and consumes a digital clock or analog clock resource as necessary.

| | Digital Blocks | | | | | API Memory (Bytes) | | |
|---|---|---|---|---|---|---|---|---|
| Analog Block | Datapaths | Macro cells | Status Registers | Control Registers | Counter7 | Flash | RAM | Pins (per External I/O) |
| N/A | N/A | N/A | N/A | N/A | N/A | 698 | 0 | N/A |

# Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "Clock_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "Clock".

**Note** Local clocks configured with **Clock Type** "Existing" on the Configure dialog will not have any APIs generated.

| Function | Description |
|---|---|
| Clock_Start | Enables the clock. |
| Clock_Stop | Disables the clock. |
| Clock_StopBlock | Disables the clock and waits until the clock is disabled. |
| Clock_StandbyPower | Selects the power for standby (Alternate Active) operation mode. |
| Clock_SetDivider | Sets the divider of the clock and restarts the clock divider immediately. |
| Clock_SetDividerRegister | Sets the divider of the clock and optionally restarts the clock divider immediately. |
| Clock_SetDividerValue | Sets the divider of the clock and restarts the clock divider immediately. |
| Clock_GetDividerRegister | Gets the clock divider register value. |
| Clock_SetMode | Sets flags that control the operating mode of the clock. |
| Clock_SetModeRegister | Sets flags that control the operating mode of the clock. |
| Clock_GetModeRegister | Gets the clock mode register value. |
| Clock_ClearModeRegister | Clears flags that control the operating mode of the clock. |
| Clock_SetSource | Sets the source of the clock. |
| Clock_SetSourceRegister | Sets the source of the clock. |
| Clock_GetSourceRegister | Gets the source of the clock. |
| Clock_SetPhase | Sets the phase delay of the analog clock (only generated for analog clocks). |
| Clock_SetPhaseRegister | Sets the phase delay of the analog clock (only generated for analog clocks). |
| Clock_SetPhaseValue | Sets the phase delay of the analog clock (only generated for analog clocks). |
| Clock_GetPhaseRegister | Gets the phase delay of the analog clock (only generated for analog clocks). |

## void Clock_Start(void)

**Description:** Starts the clock.

**Note** On startup, clocks may already be running if the “Start on Reset” option is enabled in the DWR Clock Editor.

**Parameters:** void.

**Return Value:** void.

**Side Effects:** The clock is enabled.

## void Clock_Stop(void)

**Description:** Stops the clock and returns immediately. This API does not require the source clock to be running but may return before the hardware is actually disabled. If the settings of the clock are changed after calling this function, the clock may glitch when it is started. To avoid the clock glitch, use the Clock_StopBlock() function.

**Parameters:** void.

**Return Value:** void.

**Side Effects:** The clock is disabled. The output will be logic 0.

## void Clock_StopBlock(void)

**Description:** Stops the clock and waits for the hardware to actually be disabled before returning. This ensures that the clock is never truncated (high part of the cycle will terminate before the clock is disabled and the API returns). Note that the source clock must be running or this API will never return as a stopped clock cannot be disabled.

**Parameters:** void.

**Return Value:** void.

**Side Effects:** The clock is disabled. The output will be logic 0.

**Note** The Clock_StopBlock() API is not supported on PSoC3 ES2 and PSoC5 ES1, and will not be generated.

## void Clock_StandbyPower(uint8 state)

**Description:** Selects the power for standby (Alternate Active) operation mode.

**Parameters:** uint8 state: 0 to disable clock during Alternate Active mode, nonzero to enable.

**Return Value:** void.

**Side Effects:** None

## void Clock_SetDivider(uint16 clkDivider)

**Description:** Modifies the clock divider, and thus, the frequency. When the clock divider register is set to zero or changed from zero, the clock will be temporarily disabled in order to change a mode bit. If the clock is enabled when Clock_SetDivider() is called, then the source clock must be running. The current clock cycle will be truncated and the new divide value will take effect immediately.

**Parameters:** uint16 clkDivider: Divider register value (0-65,535). This value is NOT the divider; the clock hardware divides by clkDivider plus one. For example, to divide the clock by 2, this parameter should be set to 1.

**Return Value:** void.

**Side Effects:** None

## void Clock_SetDividerRegister(uint16 clkDivider, uint8 reset)

**Description:** Modifies the clock divider, and thus, the frequency. When the clock divider register is set to zero or changed from zero, the clock will be temporarily disabled in order to change a mode bit. If the clock is enabled when Clock_SetDivider() is called, then the source clock must be running.

**Parameters:** uint16 clkDivider: Divider register value (0-65,535). This value is NOT the divider; the clock hardware divides by clkDivider plus one. For example, to divide the clock by 2, this parameter should be set to 1.

uint8 reset: If nonzero, restarts the clock divider; the current clock cycle will be truncated and the new divide value will take effect immediately. If zero, the new divide value will take effect at the end of the current clock cycle.

**Return Value:** void.

**Side Effects:** None

## void Clock_SetDividerValue(uint16 clkDivider)

**Description:** Modifies the clock divider, and thus, the frequency. When the clock divider register is set to zero or changed from zero, the clock will be temporarily disabled in order to change the SSS mode bit. If the clock is enabled when Clock_SetDivider() is called, then the source clock must be running. The current clock cycle will be truncated and the new divide value will take effect immediately.

**Parameters:** uint16 clkDivider: Divide value (1-65535) or zero. If clkDivider is zero, the clock will be divided by 65,536.
The difference between this and Clock_SetDivider() is you do not have to consider the +1 factor.

**Return Value:** void.

**Side Effects:** None

## uint16 Clock_GetDividerRegister(void)

**Description:** Gets the clock divider register value.

**Parameters:** void.

**Return Value:** Divide value of the clock minus 1. For example, if the clock is set to divide by 2, the return value will be 1.

**Side Effects:** None

## void Clock_SetMode(uint8 clkMode)

**Description:** Sets flags that control the operating mode of the clock. This function only changes flags from 0 to 1; flags that are already 1 will remain unchanged. To clear flags, use the Clock_ClearModeRegister() function. The clock must be disabled before changing the mode.

**Parameters:** uint8 clkMode: Bit mask containing the bits to set. For PSoC 3 and PSoC 5, clkMode should be a set of the following optional bits or'ed together:

- CYCLK_EARLY: Enable early phase mode. Rising edge of output clock will occur when the divider counter reaches half of the divide value.
- CYCLK_DUTY: Enable 50% duty cycle output. When enabled, the output clock is asserted for approximately half of its period. When disabled, the output clock is asserted for one period of the source clock.
- CYCLK_SYNC: Enable output synchronization to master clock. This should be enabled for all synchronous clocks.

See the Technical Reference Manual for details about setting the mode of the clock. Specifically, see the CLKDIST.DCFG.CFG2 register.

**Return Value:** void.

**Side Effects:** None

## void Clock_SetModeRegister(uint8 clkMode)

**Description:** Same as Clock_SetMode(). Sets flags that control the operating mode of the clock. This function only changes flags from 0 to 1; flags that are already 1 will remain unchanged. To clear flags, use the Clock_ClearModeRegister() function. The clock must be disabled before changing the mode.

**Parameters:** uint8 clkMode: Bit mask containing the bits to set. It should be a set of the following optional bits ORed together:

- CYCLK_EARLY: Enable early phase mode. Rising edge of output clock will occur when the divider counter reaches half of the divide value.
- CYCLK_DUTY: Enable 50% duty cycle output. When enabled, the output clock is asserted for approximately half of its period. When disabled, the output clock is asserted for one period of the source clock.
- CYCLK_SYNC: Enable output synchronization to master clock. This should be enabled for all synchronous clocks.

See the Technical Reference Manual for details about setting the mode of the clock. Specifically, see the CLKDIST.DCFG.CFG2 register.

**Return Value:** void.

**Side Effects:** None

## uint8 Clock_GetModeRegister(void)

**Description:** Gets the clock mode register value.

**Parameters:** Void.

**Return Value:** Bit mask representing the enabled mode bits. See the Clock_SetModeRegister() and Clock_ClearMode() Register descriptions for details about the mode bits..

**Side Effects:** None

## void Clock_ClearModeRegister(uint8 clkMode)

**Description:** Clears flags that control the operating mode of the clock. This function only changes flags from 1 to 0; flags that are already 0 will remain unchanged. To clear flags, use the Clock_ClearModeRegister() function. The clock must be disabled before changing the mode.

**Parameters:** uint8 clkMode: Bit mask containing the bits to clear. It should be a set of the following optional bits ORed together:

- CYCLK_EARLY: Enable early phase mode. Rising edge of output clock will occur when the divider counter reaches half of the divide value.
- CYCLK_DUTY: Enable 50% duty cycle output. When enabled, the output clock is asserted for approximately half of its period. When disabled, the output clock is asserted for one period of the source clock.
- CYCLK_SYNC: Enable output synchronization to master clock. This should be enabled for all synchronous clocks.

See the Technical Reference Manual for details about setting the mode of the clock. Specifically, see the CLKDIST.DCFG.CFG2 register.

**Return Value:** void.

**Side Effects:** None

## void Clock_SetSource(uint8 clkSource)

**Description:** Sets the input source of the clock. The clock must be disabled before changing the source. The old and new clock sources must be running.

**Parameters:** uint8 clkSource: It should be one of the following input sources:

- CYCLK_SRC_SEL_SYNC_DIG: Phase-delayed master clock.
- CYCLK_SRC_SEL_IMO: Internal main oscillator.
- CYCLK_SRC_SEL_XTALM: 4-33 MHz external crystal oscillator.
- CYCLK_SRC_SEL_ILO: Internal low-speed oscillator.
- CYCLK_SRC_SEL_PLL: Phase locked loop output.
- CYCLK_SRC_SEL_XTALK: 32.768 kHz external crystal oscillator.
- CYCLK_SRC_SEL_DSI_G: DSI global input signal.
- CYCLK_SRC_SEL_DSI_D: DSI digital input signal.
- CYCLK_SRC_SEL_DSI_A: DSI analog input signal.

See the Technical Reference Manual for details on clock sources.

**Return Value:** void.

**Side Effects:** None

## void Clock_SetSourceRegister(uint8 clkSource)

**Description:** Same as Clock_SetSource(). Sets the input source of the clock. The clock must be disabled before changing the source. The old and new clock sources must be running.

**Parameters:** uint8 clkSource: It should be one of the following input sources:

- CYCLK_SRC_SEL_SYNC_DIG: Phase-delayed master clock.
- CYCLK_SRC_SEL_IMO: Internal main oscillator.
- CYCLK_SRC_SEL_XTALM: 4-33 MHz external crystal oscillator.
- CYCLK_SRC_SEL_ILO: Internal low-speed oscillator.
- CYCLK_SRC_SEL_PLL: Phase locked loop output.
- CYCLK_SRC_SEL_XTALK: 32.768 kHz external crystal oscillator.
- CYCLK_SRC_SEL_DSI_G: DSI global input signal.
- CYCLK_SRC_SEL_DSI_D/CYCLK_SRC_SEL_DSI_A: DSI input signal.

See the Technical Reference Manual for details on clock sources.

**Return Value:** void.

**Side Effects:** None

## uint8 Clock_GetSource(void)

**Description:** Gets the input source of the clock.

**Parameters:** void.

**Return Value:** The input source of the clock. See Clock_SetSourceRegister() for details.

**Side Effects:** None

## void Clock_SetPhase(uint8 clkPhase)

**Description:** Sets phase delay of the analog clock. This function is only available for analog clocks. The clock must be disabled before changing the phase delay to avoid glitches.

**Parameters:** uint8 clkPhase: Amount to delay the phase of the clock, in 1.0 ns increments. clkPhase must be from 1 to 11 inclusive. Other values, including 0, disable the clock.

| clkPhase value | PSoC 3 ES2 and earlier | PSoC 3 ES3 and later, PSoC 5 |
|---|---|---|
| 0 | Clock disabled | Clock disabled |
| 1 | 2.5 ns | 0.0 ns |
| 2 | 3.5 ns | 1.0 ns |
| 3 | 4.5 ns | 2.0 ns |
| 4 | 5.5 ns | 3.0 ns |
| 5 | 6.5 ns | 4.0 ns |
| 6 | 7.5 ns | 5.0 ns |
| 7 | 8.5 ns | 6.0 ns |
| 8 | 9.5 ns | 7.0 ns |
| 9 | 10.5 ns | 8.0 ns |
| 10 | 11.5 ns | 9.0 ns |
| 11 | 12.5 ns | 10.0 ns |
| 12-15 | Clock disabled | Clock disabled |

**Return Value:** void.

**Side Effects:** None

## void Clock_SetPhaseRegister(uint8 clkPhase)

**Description:** Same as Clock_SetPhase(). Sets phase delay of the analog clock. This function is only available for analog clocks. The clock must be disabled before changing the phase delay to avoid glitches.

**Parameters:** uint8 clkPhase: Amount to delay the phase of the clock, in 1.0 ns increments. clkPhase must be from 1 to 11 inclusive. Other values, including 0, disable the clock.

| clkPhase value | PSoC 3 ES2 and earlier | PSoC 3 ES3 and later, PSoC 5 |
|---|---|---|
| 0 | Clock disabled | Clock disabled |
| 1 | 2.5 ns | 0.0 ns |
| 2 | 3.5 ns | 1.0 ns |
| 3 | 4.5 ns | 2.0 ns |
| 4 | 5.5 ns | 3.0 ns |
| 5 | 6.5 ns | 4.0 ns |
| 6 | 7.5 ns | 5.0 ns |
| 7 | 8.5 ns | 6.0 ns |
| 8 | 9.5 ns | 7.0 ns |
| 9 | 10.5 ns | 8.0 ns |
| 10 | 11.5 ns | 9.0 ns |
| 11 | 12.5 ns | 10.0 ns |
| 12-15 | Clock disabled | Clock disabled |

**Return Value:** void.

**Side Effects:** None

### void Clock_SetPhaseValue(uint8 clkPhase)

**Description:** Sets phase delay of the analog clock. This function is only available for analog clocks. The clock must be disabled before changing the phase delay to avoid glitches. Same as Clock_SetPhase(), except Clock_SetPhaseValue() adds one to the value and then calls Clock_SetPhaseRegister() with it.

**Parameters:** uint8 clkPhase: Amount to delay the phase of the clock, in 1.0 ns increments. clkPhase must be from 0 to 10 inclusive. Other values disable the clock.

| clkPhase value | PSoC 3 ES2 and earlier | PSoC 3 ES3 and later, PSoC 5 |
|---|---|---|
| 0 | 2.5 ns | 0.0 ns |
| 1 | 3.5 ns | 1.0 ns |
| 2 | 4.5 ns | 2.0 ns |
| 3 | 5.5 ns | 3.0 ns |
| 4 | 6.5 ns | 4.0 ns |
| 5 | 7.5 ns | 5.0 ns |
| 6 | 8.5 ns | 6.0 ns |
| 7 | 9.5 ns | 7.0 ns |
| 8 | 10.5 ns | 8.0 ns |
| 9 | 11.5 ns | 9.0 ns |
| 10 | 12.5 ns | 10.0 ns |
| 11-15 | Clock disabled | Clock disabled |

**Return Value:** void.

**Side Effects:** None

### uint8 Clock_GetPhaseRegister(void)

**Description:** Gets the phase delay of the analog clock. This function is only available for analog clocks.

**Parameters:** void.

**Return Value:** Phase of the analog clock in nanoseconds. See Clock_SetPhaseRegister() for details.

**Side Effects:** None

## Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

# Known Problems

The cy_clock v1.50 component for PSoC 5 should not be used with the Clock_SetDividerRegister() function. This function has a known problem when used with this silicon. If this function is needed, the component should be updated to the latest component version.

# Component Changes

This section lists the major changes in the component from the previous version.

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 1.50.d | Minor datasheet edits. | |
| 1.50.c | Minor datasheet edits. | |
| 1.50.b | Added Known Problems section to datasheet | |
| 1.50.a | Added note to Clock_StopBlock() in datasheet to note lack of silicon support | |
| | Minor datasheet edits and updates | |
| 1.50 | Added Clock_StopBlock() API | This function stops the clock and waits for it to be disabled. This is necessary to prevent glitches when changing settings and restarting a clock. |
| | Added Clock_GetPhaseRegister() API (analog only) | Allows the firmware to read the current phase value. |
| | Added Clock_SetPhaseValue() API (analog only) | This macro wraps Clock_SetPhaseRegister() and automatically adds 1 to the phase value to provide a more intuitive interface. |
| | Renamed Clock_SetPhase() to Clock_SetPhaseRegister() (analog only) | For consistency with other names. For compatibility, SetPhase is provided as a macro and has the same effect as Clock_SetPhaseRegister(). |
| | Added Clock_GetSourceRegister() API | Allows the firmware to read the current clock source. |
| | Renamed Clock_SetSource() to Clock_SetSourceRegister() | For consistency with other names. For compatibility, SetSource is provided as a macro and has the same effect as Clock_SetSourceRegister(). |
| | Added Clock_GetModeRegister() API | Allows the firmware to read the current mode flags. |
| | Added Clock_SetModeRegister() API | This function replaces Clock_SetMode(). For compatibility, SetMode is provided as a macro and has the same effect as Clock_SetModeRegister(). Clock_SetModeRegister() only changes mode flags from 0 to 1. This prevents unintended clearing of other mode bits such as SYNC. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Added Clock_ClearModeRegister() API | This function is similar to Clock_SetModeRegister(), but only changes mode flags from 1 to 0. |
| | Added Clock_GetDividerRegister() API | Allows the firmware to read the current divider value. |
| | Added Clock_SetDividerRegister() API | The Clock_SetDivider() API unconditionally resets the clock divider. Clock_SetDividerRegister() allows the firmware author to control whether the divider is reset. |
| | Added Clock_SetDividerValue() API | This macro wraps Clock_SetDividerRegister() and automatically subtracts 1 from the divider to provide a more intuitive interface. |
| | Set SSS in Clock_SetDividerRegister() | When dividing by 1 (divide value of 0), the SSS bit must be set to bypass the divider. The Clock_SetDividerRegister() function will automatically set/clear SSS, temporarily disabling the clock if necessary. |
| | Changed register definitions | Updated to match component coding guidelines. |
| | Corrected Clock_SetDivider() API documentation | The Clock_SetDivider() API documentation stated that the clkDivider parameter should be the divide value + 1. This should have been the divide value - 1. The documentation incorrectly stated that 0 was an invalid value for clkDivider. |
| | Changed "Synch with Bus" to "Sync with Master" and associated tooltip on the Configure dialog. | Updated to match how the device works. This was just a cosmetic change. |
| | Added parameter to enable the digital domain output from the analog clock. | A signal is available from analog clocks in the hardware that was not previously exposed on the component. |
| | Added `=ReentrantKeil($INSTANCE_NAME . "_...")` to the following functions:<br>void Clock_Start()<br>void Clock_Stop()<br>void Clock_StopBlock()<br>void Clock_StandbyPower()<br>void Clock_SetDividerRegister()<br>uint16 Clock_GetDividerRegister()<br>void Clock_SetModeRegister()<br>void Clock_ClearModeRegister()<br>uint8 Clock_GetModeRegister()<br>void Clock_SetSourceRegister()<br>uint8 Clock_GetSourceRegister()<br>void Clock_SetPhaseRegister()<br>uint8 Clock_GetPhaseRegister() | Allows users to make these APIs reentrant if reentrancy is desired. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 1.0a | Move CYCLK_ constants to *cydevice.h*/*cydevice_trm.h*. | The CYCLK_ constants for the mode and source are now generated from the selected device's register map. This allows the clock component to be independent of device-specific register values. The *cydevice.h* file is already included from the clock header, so no user code changes should be necessary. |
| | Add description of CYCLK_ constants in the data sheet. | The parameter descriptions for the Clock_SetMode() and Clock_SetSource() APIs now contain a description of each value. |