


Clock (SysClk_PDL)

1.0

Features

- Generates programmable clock dividers for use with other Components that require clocks
- 8 bit, 16 bit, 16.5 bit and 24.5 bit dividers available
- Configure clock by specifying frequency, or divider value
- Specify frequency tolerance for clock
- Phase align with another programmable clock divider

Clock_1  12 MHz

General Description

The SysClk_PDL Component provides an interface to the programmable peripheral clock dividers. It allows you to configure the dividers by specifying a frequency with tolerance, or by specifying a divider.

All SysClk_PDL Components are sourced from clk_peri, thus you specify a divider from the frequency of clk_peri. For details, see [Figure 1](#), [Figure 2](#), and [Figure 3](#) in [Functional Description](#). To configure the frequency of clk_peri, use the Design-Wide Resources Clock Editor in your PSoC Creator project.

When to Use a Clock


This Component should be used anytime a clock input is needed for another Component such as a TCPWM, or SCB, or UDB based Component.

Input/Output Connections

This section describes the various input and output connections for the SysClk_PDL Component.

clock – output

The SysClk_PDL Component has a standard output terminal that provides access to the clock signal.

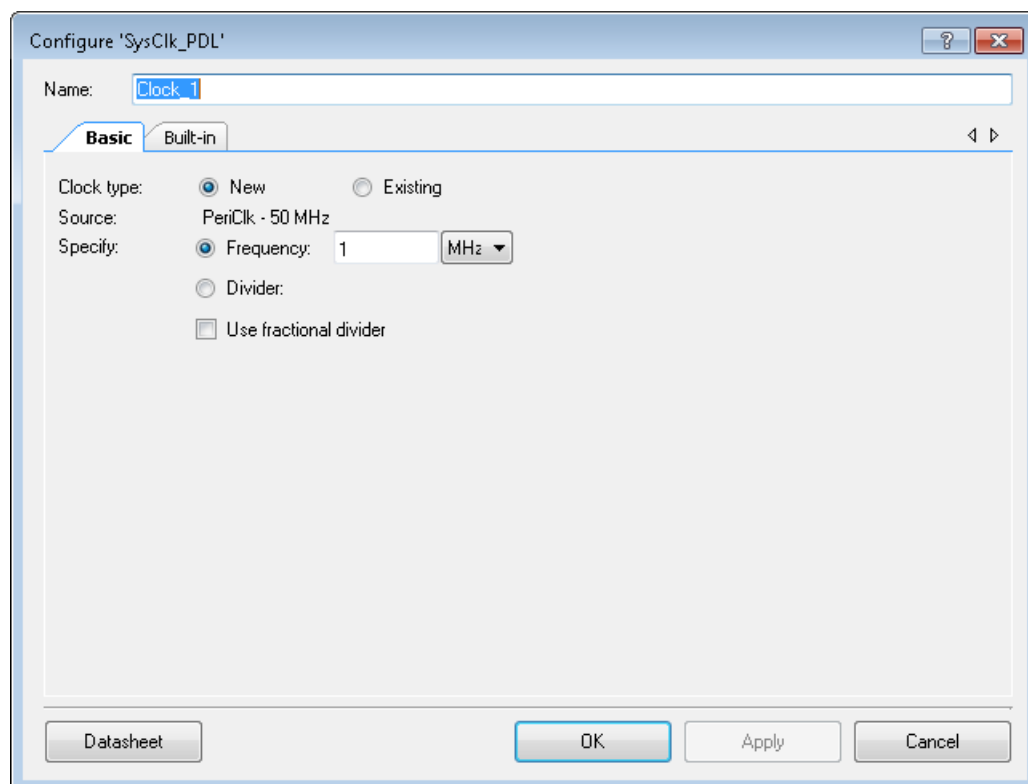
Clock_1  12 MHz

PRELIMINARY

Component Parameters

Drag a SysClk_PDL Component onto your design and double click it to open the Configure dialog. This dialog has the following tabs with different parameters.

Note For any local clock you add to your design, the Design-Wide Resources 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_Enable() function in your code. See the Application Programming Interface section of this datasheet and the Clock Editor section of the PSoC Creator Help, for more details.



Clock type

There are two clock types: **New** and **Existing**. A new clock uses a programmable divider. An existing clock uses a clock that already exists in the system. Note that the option to select **Existing** may not be available, depending on the Component being clocked. When **Existing** is chosen, the **Source** drop-down shows all existing clocks. Existing clocks are those routed from the clock tree to the DSI. This is device dependent.

PRELIMINARY



Source

The source is always the peripheral clock, clk_peri. The current peri_clk frequency setting is displayed.

Specify:

- **Frequency:** You can specify your clock by frequency or by divider value. For frequency, select the **Frequency** option and enter the desired frequency value and units (Hz, kHz, MHz).
- **Divider:** To specify your clock by divider value, select the **Divider** option and enter the desired divider value.
 - **Use fractional divider:** If **Divider** is selected, you can optionally select the **Use fractional divider** box and specify a fractional divider value, in units of 1/32, from 0 to 31. For more information about fractional division, refer to the device *Technical Reference Manual*.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the Component using software.

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.

This Component uses the sysclk driver module from the PDL. The driver is copied into the “pd\drivers\peripheral\sysclk\” directory of the application project after a successful build.

Refer to the PDL documentation for a detailed description of the complete API. To access this document, right-click on the Component symbol on the schematic and choose the “**Open PDL Documentation...**” option in the drop-down menu.

Preprocessor Macros

The SysClk_PDL Component generates the following preprocessor macro(s). Note that each macro is prefixed with the instance name of the Component (e.g., “Clock_1”).

```
#define Clock_1_DIV_NUM ((uint32_t)Clock_1__DIV_NUM)
```

The peripheral clock divider number

```
#define Clock_1_DIV_TYPE ((cy_en_divider_types_t)Clock_1__DIV_TYPE)
```

The peripheral clock divider type



PRELIMINARY

Component Functions

This Component also includes a set of Component-specific wrapper functions that provide simplified access to the basic sysclk operation. These functions are generated during the build process and are all prefixed with the name of the Component instance.

__STATIC_INLINE void Clock_1_Enable (void)

Enables the programmable clock divider assigned to this Component.

__STATIC_INLINE void Clock_1_Disable (void)

Disables the programmable clock divider assigned to this Component.

__STATIC_INLINE void Clock_1_SetDivider (uint32_t dividerValue)

Sets the value of a programmable clock divider assigned to this Component. This is only used for integer dividers. Use [Clock_1_SetFracDivider\(\)](#) for setting fractional dividers.

Parameters:

<i>dividerValue</i>	The divider value. The source of the divider is peri_clk which is a divided version of hf_clk[0]. The divider value causes integer division of (divider value + 1), or division by 1 to 256 (8-bit divider) or 1 to 65536 (16-bit divider).
---------------------	---

Returns:

None

__STATIC_INLINE uint32_t Clock_1_GetDivider (void)

Returns the integer divider value for the programmable clock divider assigned to this Component. This is only used for integer dividers. Use [Clock_1_GetFracDivider\(\)](#) with a fractional divider.

Returns:

The divider value. The source of the divider is peri_clk which is a divided version of hf_clk[0]. The integer division done is by (divider value + 1), or division by 1 to 256 (8-bit divider) or 1 to 65536 (16-bit divider).

__STATIC_INLINE void Clock_1_SetFracDivider (uint32_t dividerIntValue, uint32_t dividerFracValue)

Sets the values of a programmable clock divider assigned to this Component. This is only used for fractional dividers. Use [Clock_1_SetDivider\(\)](#) for setting integer dividers.

Parameters:

<i>dividerIntValue</i>	The integer divider value. The source of the divider is peri_clk which is a divided version of hf_clk[0]. The divider value causes integer division of (divider value + 1), or division by 1 to 65536 (16-bit divider) or 1 to 16777216 (24-bit divider).
<i>dividerFracValue</i>	This is the fraction piece of the divider. The fractional divider can be 0 - 31; it divides the clock by 1/32 for each count. To divide the clock by 11/32nds, set this value to 11.

Returns:

None

PRELIMINARY



__STATIC_INLINE void Clock_1_GetFracDivider (uint32_t *dividerIntValue, uint32_t *dividerFracValue)

Returns the divider values for the programmable clock divider assigned to this Component. This is only used for fractional dividers. Use [Clock_1_GetDivider\(\)](#) with an integer divider.

Parameters:

*dividerIntValue	pointer to return integer divider value
*dividerFracValue	pointer to return fractional divider value

Returns:

None. Loads pointed-to variables.

Code Examples and Application Notes

Code Examples

PSoC Creator provides access to code examples in the Code Example 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 Code Example" topic in the PSoC Creator Help for more information.

There are also numerous code examples that include schematics and example code available online at the Cypress Code Examples web page. Examples that use this Component include:

- CE2##### - Clock Configuration

API Memory Usage

The Component memory usage varies significantly depending on the compiler, device, number of APIs used and Component configuration. The following table provides the memory usage for all APIs available in the given Component configuration.

The measurements have been done with an associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

PSoC 6 (GCC)

Configuration		Flash Bytes	SRAM Bytes
Includes PDL driver functions		6K	0

Functional Description

Describes the functionality of the Component and expands on complex topics.

Definitions

Include this section if relevant; otherwise, delete. It is considered as relevant if there are five or more not common industry abbreviations and acronyms. Abbreviations and acronyms should be spelled out on the first usage if they are not common industry terms like IO or PC.

- IMO – Internal main oscillator (8 MHz)
- ECO – External crystal oscillator (MHz)
- EXT – External clock, typically MHz range
- DSI – Digital signal interconnect. The DSI is a switching fabric that routes signals within the Universal Digital Block (UDB) matrix and throughout the device. Signals can be routed from the DSI to be used as clock inputs.
- ALTHF – Alternate high frequency clock (future implementation)
- ILO – Internal low-speed oscillator (32.768 kHz nominal)
- PILO – Precision internal low-speed oscillator (32.768 kHz nominal). May also be referred to as ALTLF, alternate low-frequency clock
- WCO – Watch crystal oscillator (32.768 kHz nominal)
- FLL – Frequency lock loop. Used to generate higher-frequency clocks from an input clock. Only one FLL circuit exists in a device.
- PLL – Phased lock loop. Used to generate higher-frequency clocks from an input clock. One or more PLL circuits exist in a device, depending on device family.
- Clk_paths – Outputs of FLL and PLLs; inputs to HF_CLK circuits.
- HF_CLK – High frequency clock. There are as many HF_CLK instances as there are FLL and PLLs. HF_CLK[0] is a source for the CPU and peripheral clocks. HF_CLKs are only available in active modes.
- Clk_fast – Cortex-M4 CPU clock. Divided from HF_CLK[0].
- Clk_peri – Peripheral clock. Divided from HF_CLK[0]. Clock source for clk_slow and peripheral clock dividers.
- Clk_slow – Cortex-M0+ CPU clock. Divided from Clk_peri.

PRELIMINARY



- **LF_CLK** – Low frequency clock. Selection of one of the available 32.768-kHz clocks. Available in the low-power modes.
- **CSV** – Clock Supervisor. Circuit attached to certain clocks, that detects if the clock has stopped running or is out of frequency tolerance.

Block Diagram and Configuration

The following is a simplified diagram of the system clocks hardware:

Figure 1. Clock Generation Diagram

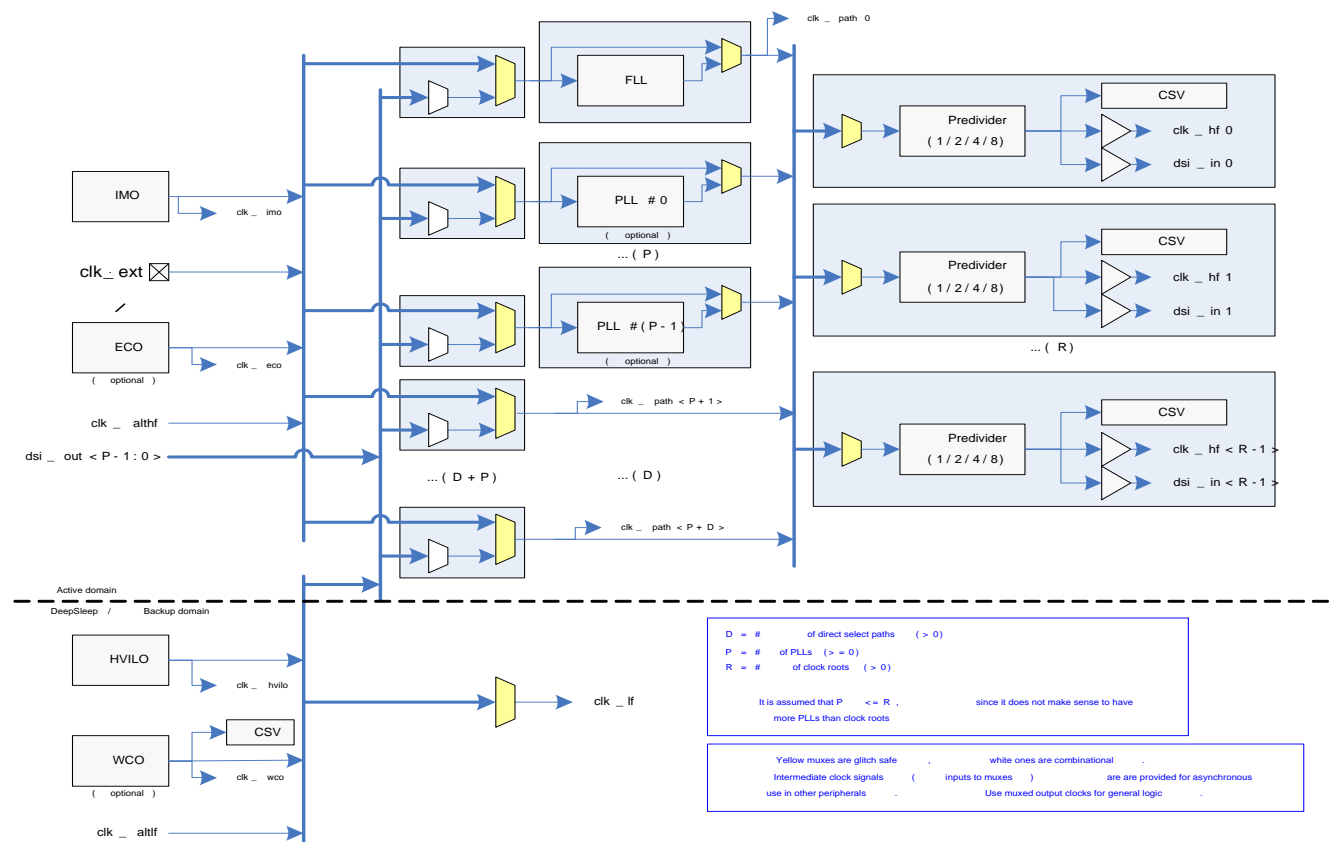
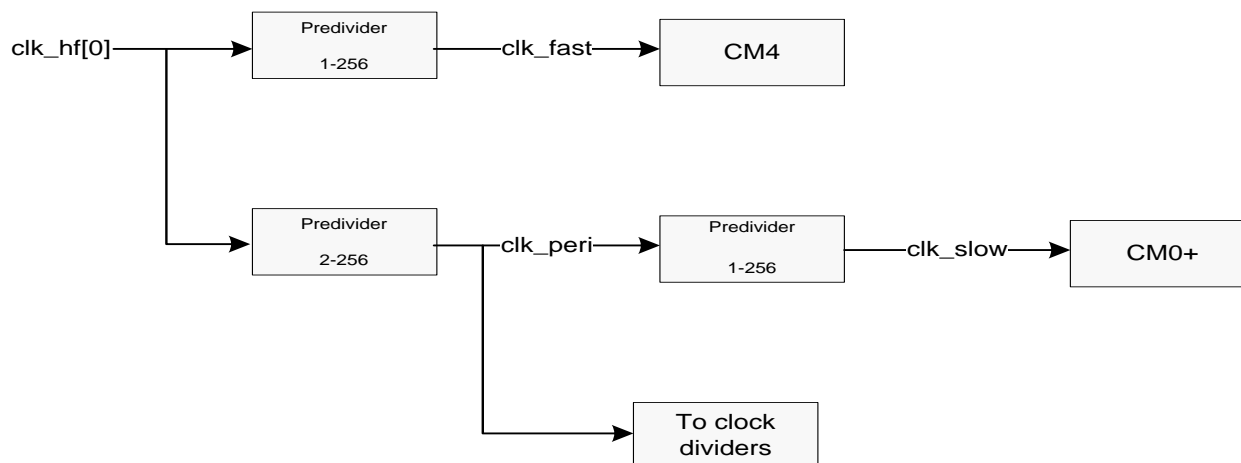
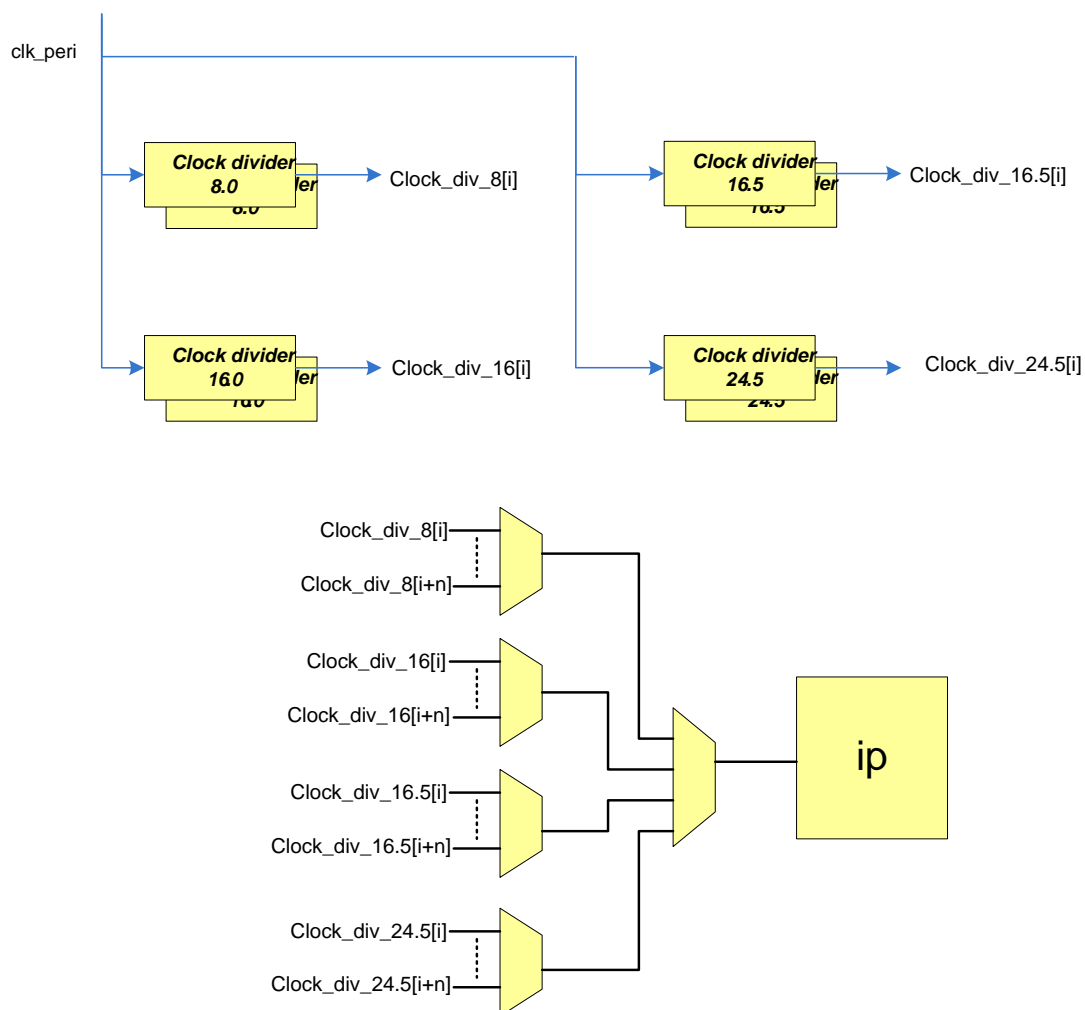


Figure 2. CPU Subsystem Clocks**Figure 3. Peripheral Clock Dividers****PRELIMINARY**

Placement

PSoC Creator selects an appropriate clock divider (see [Figure 3](#)) based on Component configuration and the Components to which the SysClk_PDL Component output is connected.

MISRA-C Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

This Component has the following specific deviations:

Rule	Rule Class (Required/Advisory)	Rule Description	Description of Deviation(s)
1.1	R	This rule states that code shall conform to C ISO/IEC 9899:1990 standard.	PDL v3.0.0 supports ISO:C99 standard.
1.2	R	No reliance on undefined behavior.	Calculation of absolute value in FLL and PLL configure.
5.6	R	No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.	The "mode" and "retval" are used as a structure/union member they are a label, tag or ordinary identifier.
10.1	R	The value of an expression of integer type shall not be implicitly converted to a different, underlying type if the expression is complex.	Using a Cypress defined macro to access memory mapped objects. Using a Cypress defined macro to divide with rounding.
10.2	R	The value of an expression of floating type shall not be implicitly converted to a different type if: (a) it is not a conversion to a wider floating type, or (b) the expression is complex, or (c) the expression is a function argument, or (d) the expression is a return expression.	The operands of this relational operator are expressions of different "essential type" categories (enum and unsigned).
10.3	R	A composite integer expression is being cast to a wider type.	Use of a Cypress defined macro to access memory-mapped objects. Calculating the clock parameters.
10.4	R	A composite floating point expression is being cast to double, or unsigned.	Use of the C library sqrt() function. Casting a floating-point calculation result to an integer.

Rule	Rule Class (Required/Advisory)	Rule Description	Description of Deviation(s)
12.2	A	The value of an expression must be the same under any order of evaluation that the standard permits.	The "rtnval" is modified more than once between the sequence points - the evaluation order is not specified.
12.4	A	The right hand operand of a logical && or operator shall not contain side effects.	No side effect in this case.
13.4	R	The controlling expression of a for statement shall not contain any objects of floating type.	Scanning through a list of floating point values.
14.3	R	Before preprocessing, a null statement only occurs on a line by itself; it may be followed by a comment if the first character following the null statement is a white-space character.	The null statement is located close to other code or comments.
14.8	R	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	The body of the control statement is on the same line and is not enclosed within braces.
14.9	R	An if (expression) construct is followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	The body of the control statement is on the same line and is not enclosed within braces.
18.4	R	Unions shall not be used.	Clock path in Cy_SysClk_PeriphGetFrequency() may use either FLL or PLL.
19.7	A	A function shall be used in preference to a function-like macro.	Function-like macros are used to achieve more efficient code.

Registers

Refer to the device *Technical Reference Manual* for more information about the system clock control registers.

Resources

A SysClk_PDL Component consumes a single clock divider resource.

DC and AC Electrical Characteristics

Specifications are valid for $-40\text{ }^{\circ}\text{C} \leq T_A \leq 85\text{ }^{\circ}\text{C}$ and 1.71 V to 3.6 V except where noted.

DC Specifications

Spec ID	Parameter	Description	Min	Typ	Max	Units	Details/Conditions
SID218	I _{IMO1}	IMO operating current	-	-	100	μA	8 MHz

PRELIMINARY



Spec ID	Parameter	Description	Min	Typ	Max	Units	Details/Conditions
SID316	IDD_MHz	MHz ECO operating current	-	1	-	mA	-
SID307P	PLL_IDD	PLL operating current	-	-	1.3	mA	-
SID231	I _{ILO2}	ILO operating current	-	0.3	1.05	μA	32 kHz
SID318	WCO_kHz	WCO operating current	-	0.25	1	μA	With 32-kHz crystal. After trimming, including aging and temperature drift.
SID321E	ESR32K	WCO crystal equivalent series resistance	-	50	-	kΩ	-
SID322E	PD32K	WCO drive level	-	-	1	μW	-

AC Specifications

Spec ID	Parameter	Description	Min	Typ	Max	Units	Details/Conditions
SID223	F _{IMOTOL1}	IMO frequency variation	-	-	±1	%	Centered on 8 MHz
SID226	T _{STARTIMO}	IMO startup time	-	-	12	μs	-
SID227	T _{JITRMSIMO1}	IMO RMS jitter at 8 MHz	-	156	-	ns	-
SID317	F_MHz	ECO crystal frequency range	4	-	33	MHz	-
SID305	EXTCLKFREQ	External clock input frequency	0	-	100	MHz	-
SID306	EXTCLKDUTY	External clock duty cycle	45	-	55	%	Measured at VDD/2
SID305P	PLL_LOCK	Time to achieve PLL lock	-	40	-	μs	-
SID306P	PLL_OUT	Output frequency from PLL	-	-	150	MHz	-
SID234	T _{STARTILO1}	ILO startup time	-	-	2	ms	-
SID236	T _{ILODUTY}	ILO duty cycle	40	50	60	%	-
SID237	F _{ILOTRIM1}	ILO 32-kHz trimmed frequency	15	32	50	kHz	-
SID319	F_kHz	WCO 32-kHz trimmed frequency	-	32.768	-	kHz	-
SID320	Ton_kHz	WCO startup time	-	-	500	ms	-
SID320E	FTOL32K	WCO frequency tolerance	-	50	250	ppm	-

References

TBD



PRELIMINARY

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.0.b	Updated datasheet.	Replaced Configure dialog screen capture. Updated the MISRA table.
1.0.a	Updated datasheet.	Fixed a typo. Regenerated API documentation.
1.0	New Component	

© Cypress Semiconductor Corporation, 2016-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.

PRELIMINARY

