PSoC 4 Low-Frequency Clock (cy_lfclk)

1.10

Features

- APIs to select Low-Frequency Clock (LFCLK) and Watch Crystal Oscillator (WCO) clock sources
- APIs to control Internal Low-Frequency Oscillator (ILO), WCO, Deep Sleep Timers, and Watchdog Timers (WDT)
- APIs to compensate and trim ILO frequency

General Description

The PSoC 4 Low-Frequency Clock (cy_lfclk) component is a design-wide component present in all PSoC 4 projects by default. It provides the application interface to configure various low-frequency clocks available in PSoC 4. These functions are not part of any component libraries, but the functions may be used by them. The cy_lfclk component also provides functions to configure WDTs and Deep Sleep Timers present in the device.

WDTs are available for PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M / PSoC 4000S devices. Deep Sleep Timers are available for PSoC 4100S and PSoC Analog Coprocessor devices. The main functional difference between WDTs and Deep Sleep Timers is that Deep Sleep Timers cannot generate device reset but WDTs can.

The cy_lfclk is not visible in the Component Catalog, but the API library is available all the time.

When to Use cy_lfclk

This component provides an interface to configure low-frequency clocks and watchdog timers. Use this interface to configure these resources as needed. PSoC Creator uses the interface to initialize the resources as configured in the Design-Wide Resources (*<project>.cydwr*) file.

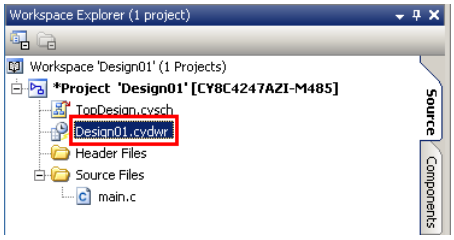
Input/Output Connections

The cy_lfclk component does not have input or output connections.

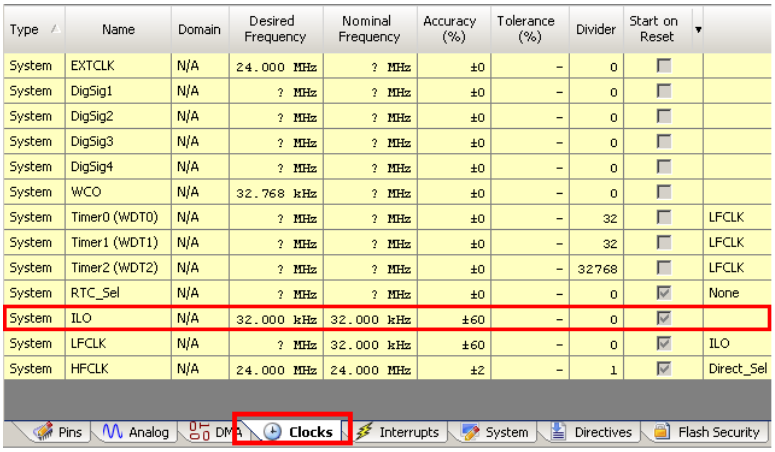
Component Parameters

In the PSoC Creator Workspace Explorer, double-click the *<project>.cydwr* file to open it. Then, click the **Clocks** tab to open the Clock Editor, and double-click any LFCLK clock source to open the Configure System Clock dialog.

Workspace Explorer



DWR Clock Editor

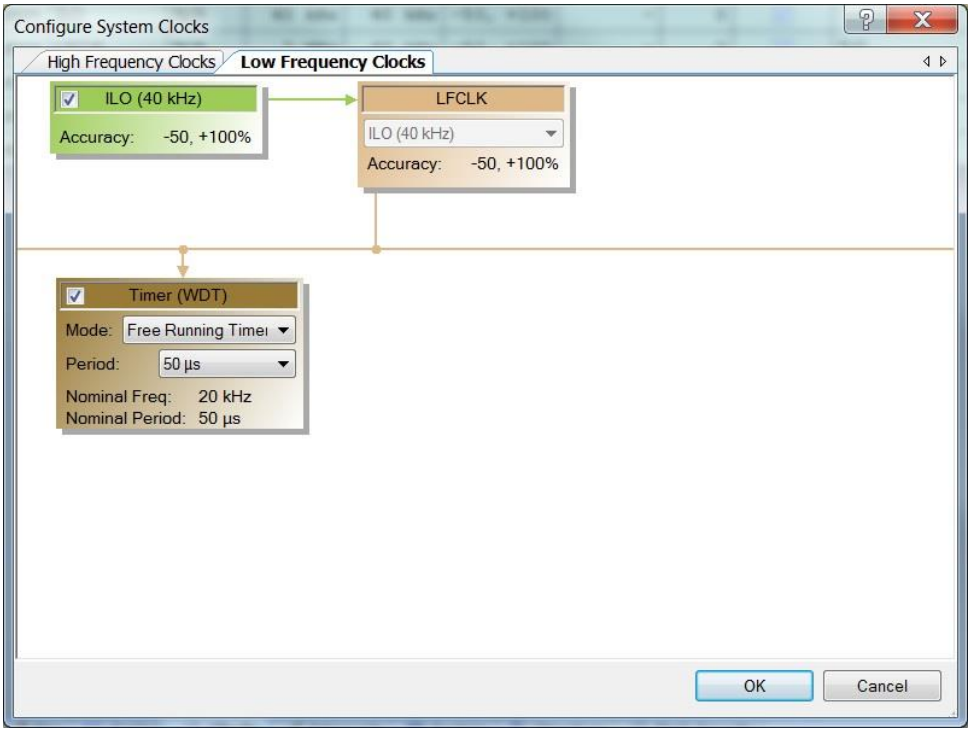


Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	
System	EXTCLK	N/A	24.000 MHz	? MHz	±0	-	0	☐	
System	DigSig1	N/A	? MHz	? MHz	±0	-	0	☐	
System	DigSig2	N/A	? MHz	? MHz	±0	-	0	☐	
System	DigSig3	N/A	? MHz	? MHz	±0	-	0	☐	
System	DigSig4	N/A	? MHz	? MHz	±0	-	0	☐	
System	WCO	N/A	32.768 kHz	? MHz	±0	-	0	☐	
System	Timer0 (WDT0)	N/A	? MHz	? MHz	±0	-	32	☐	LFCLK
System	Timer1 (WDT1)	N/A	? MHz	? MHz	±0	-	32	☐	LFCLK
System	Timer2 (WDT2)	N/A	? MHz	? MHz	±0	-	32768	☐	LFCLK
System	RTC_Sel	N/A	? MHz	? MHz	±0	-	0	☑	None
System	ILO	N/A	32.000 kHz	32.000 kHz	±60	-	0	☑	
System	LFCLK	N/A	? MHz	32.000 kHz	±60	-	0	☑	ILO
System	HFCLK	N/A	24.000 MHz	24.000 MHz	±2	-	1	☑	Direct_Sel

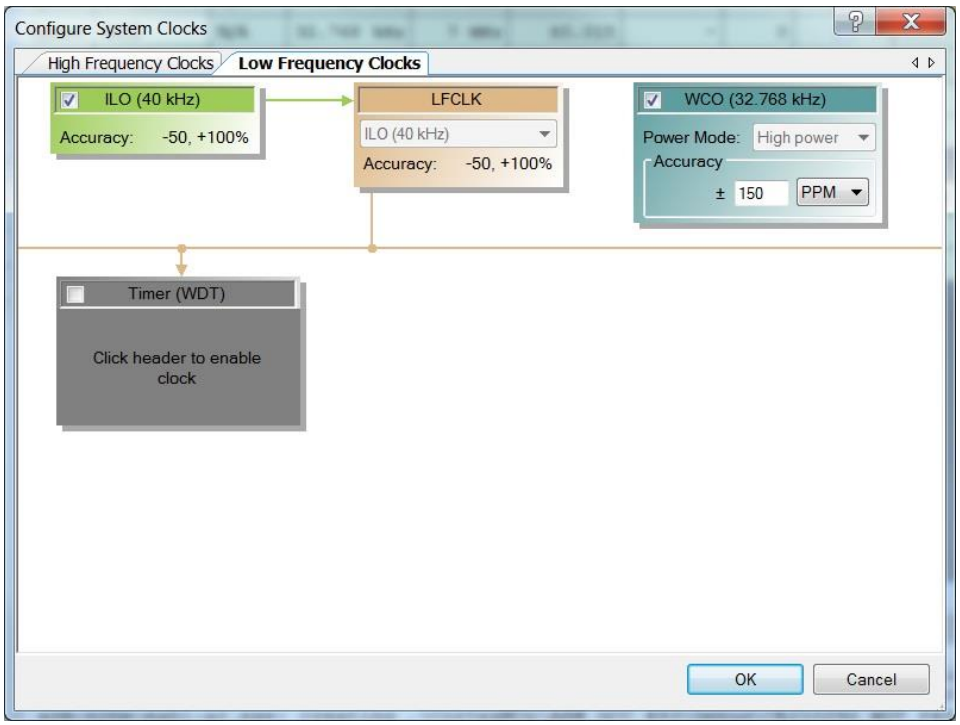
Pins | Analog | DMA | Clocks | Interrupts | System | Directives | Flash Security

The Configure dialog has different options based on the device selected for the design.

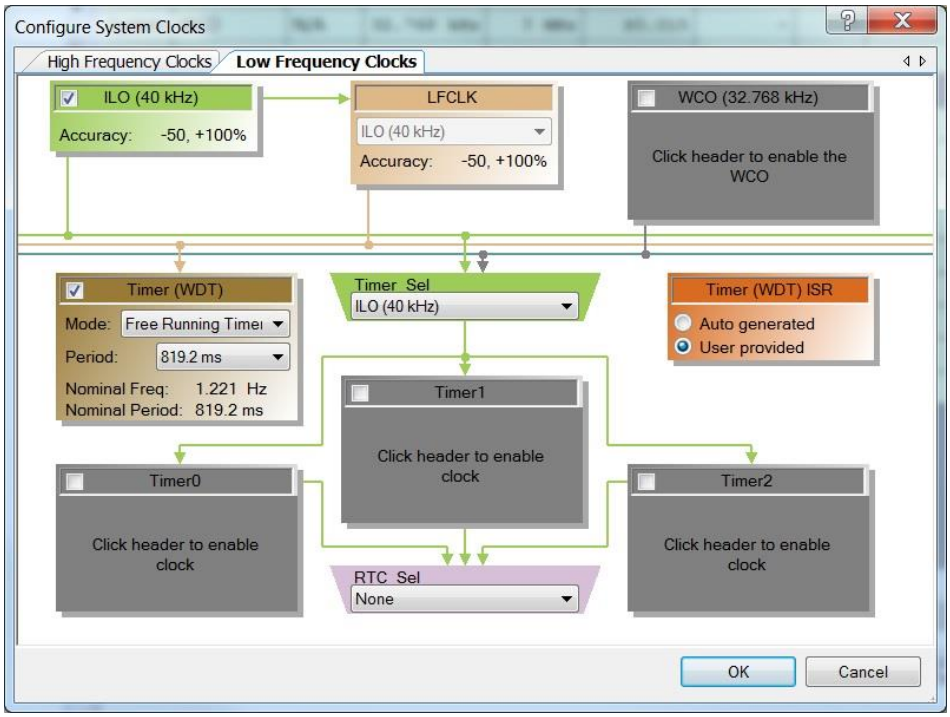
PSoC 4000 Configure Dialog



PSoC 4000S Configure Dialog

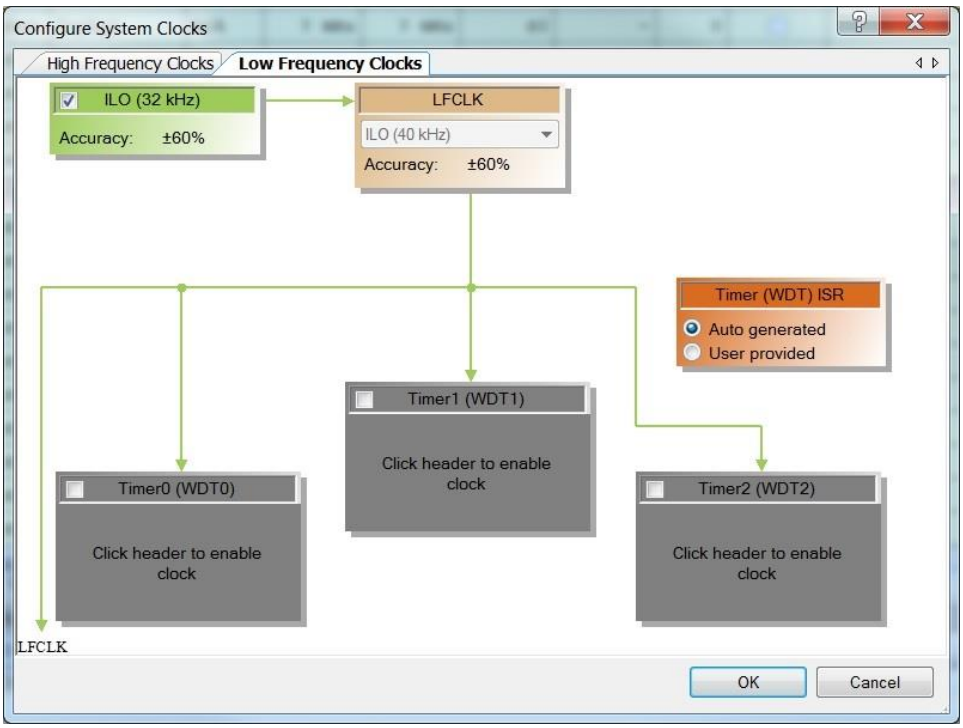


PSoC 4100S and PSoC Analog Coprocessor Configure Dialog

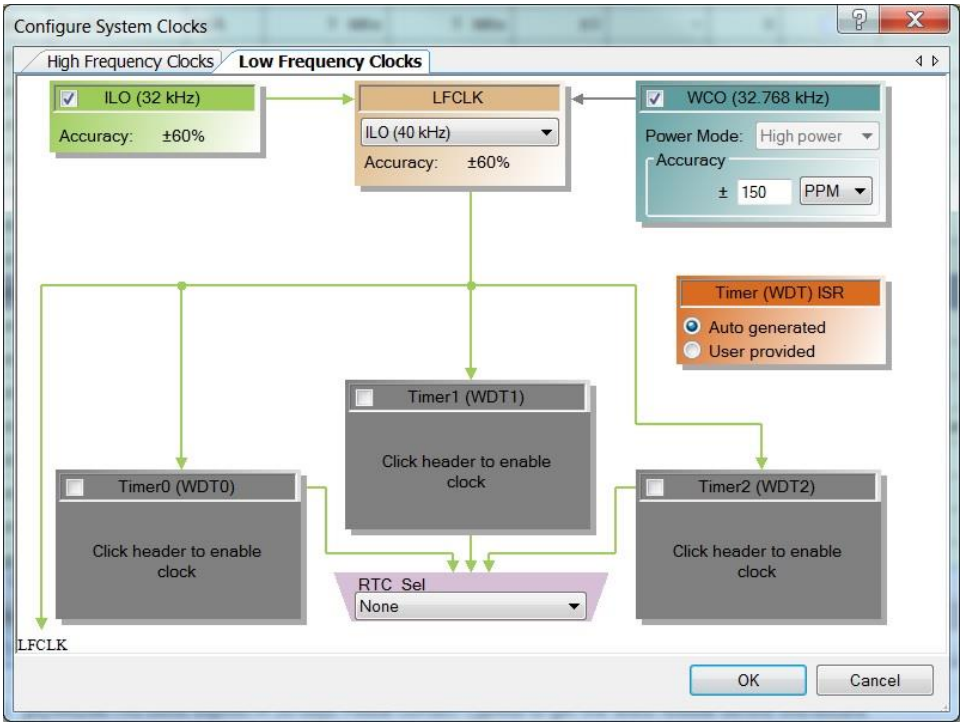


Note For the PSoC 4100S and PSoC Analog Coprocessor families, the Timer (WDT) ISR panel is used to configure the ISR for the Deep Sleep Timers.

PSoC 4100 / PSoC 4200 Configure Dialog



PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE /PSoC 4200L / PSoC 4100M / PSoC 4200M Configure Dialog



LFCLK Configuration Panels

The following table lists and describes the various panels that can be included in this dialog for various devices.

Panel	Description
ILO	This panel is used to configure the Internal Low-Frequency Oscillator. The ILO panel is available for all PSoC4 family devices.
LFCLK	This panel is used to select the LFCLK clock source. There are two possible clock sources: • ILO (32.000 kHz) • WCO (32.768 kHz) The option of selecting the LFCLK clock source WCO/ILO is applicable for PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4100L / PSoC 4200L / PSoC 4100M / PSoC 4200M. For all other devices, this option is greyed out. **Warning** LFCLK clock source can be changed while program executing. It is important to switch off all peripherals that are driven by the LFCLK while clock source switching. At least the interrupts should be masked if it is impossible to switch off peripherals. If you do not switch off the peripherals driven by the LFCLK or do not mask the interrupts, it can prevent the program from halting. For PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor devices, LFCLK can only be sourced from ILO even though WCO is available. The LFCLK panel is available for all PSoC 4 family devices.
WCO	This panel is used to configure the Watch Crystal Oscillator. It provides the interface to the following settings: • WCO Power mode • WCO Accuracy The **Power mode** option is available only for PSoC 4 BLE devices. For all other PSoC 4 devices, this field is fixed to high power and is grayed out. The WCO panel is available for PSoC 4000S / PSoC 4100S / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4100L / PSoC 4200L / PSoC 4100M / PSoC 4200M / PSoC Analog Coprocessor family devices. This panel is not available for the PSoC 4000 / PSoC 4100 / PSoC 4200 devices.

Panel	Description
RTC_Sel	This panel is used to select which WDT or Deep Sleep Timers to use for RTC. It also provides a **None** option when not using WDT or Deep Sleep Timers to clock the RTC. There are four possible RTC clock sources: • None • Timer0 (WDT0) or Timer0 (Deep Sleep Timer0) • Timer1 (WDT1) or Timer1 (Deep Sleep Timer1) • Timer2 (WDT2) or Timer2 (Deep Sleep Timer2) The RTC_Sel mux is available for devices that have WDTs or Deep Sleep Timers. This panel is not available for the PSoC 4000 / PSoC 4100 / PSoC 4200 devices. WDTs are available for PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M / PSoC 4000S devices. Deep Sleep Timers are available for PSoC 4100S and PSoC Analog Coprocessor devices.
Timer_sel	This mux selects the clock source for the Timers (Timer0/1/2) – ILO or WCO. For RTC support, it is recommended to enable WCO in the design and use that to source the Timers. **Warning** Timer_sel clock source can be changed while program executing. It is important to switch off all peripherals that are driven by Timer_sel while clock source switching. At least the interrupts should be masked if it is impossible to switch off the peripherals. If you do not switch off the peripherals driven by Timer_sel or do not mask the interrupts, it can prevent the program from halting. The Timer_sel mux is available only for PSoC 4100S and PSoC Analog Coprocessor devices.
WDT	This panel provides the option to enable and configure the Watchdog timers. The panel provides an interface to the following settings: Mode – WDT operation mode: • Free Running Timer – Does not generate an interrupt or reset. You can read the counter and set an interrupt in the firmware to generate occasional timing loops. • Periodic Timer – Generates an interrupt on a match event but no reset. The timer wraps at the set divider value. • Watchdog – Generates a reset on a match event (counter should be cleared before reaching a match event to prevent a reset). • Watchdog (w/interrupts) – Generates an interrupt on a match event and generates a reset on a 3rd unserviced interrupt. Period – This control provides the option to configure the WDT period. **Note** The WDT cascade options are not configurable using these panels but the APIs can be used to perform cascading of WDTs. WDT panel is available for all PSoC4 family devices.

Panel	Description
Timer0/Timer1/Timer2	This panel provides the option to enable and configure the Deep Sleep Timers. The panel provides an interface to the following settings: Mode – Timers operation mode: • Free Running Timer – Does not generate an interrupt or reset. You can read the counter and set an interrupt in the firmware to generate occasional timing loops. (Applicable only for Timer 0 and Timer 1) • Periodic Timer – Generates an interrupt on a match event but no reset. The timer wraps at the set divider value. The Deep Sleep Timer cascade options are not configurable using these panels but the APIs can be used to perform cascading of Timers. The Deep Sleep Timers panel is available only for PSoC 4100S and PSoC Analog Coprocessor devices.
Timer (WDT) ISR	This panel provides options for how the WDT or Deep Sleep Timer (depends on the device) interrupt handler should be implemented. For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M / PSoC 4000S family devices, this panel is used to configure the WDT interrupt. • If you select the auto-generated option, then the CySysWdtIsr() function is registered as the WDT interrupt handler. In this case, you can use the CySysWdtGetInterruptCallback()/ CySysWdtSetInterruptCallback() functions to Get/Set callbacks for each particular counter and use the CySysWdtEnableCounterIsr()/CySysWdtDisableCounterIsr()functions to Enable/Disable service of the registered callbacks for each particular counter. • If you select the User-provided option, then the CySysWdtIsr() function is not registered as the WDT interrupt handler. In this case, you can register either a custom handler or the CyWdtIsr() function by using the CyIntSetVector() API. For PSoC 4100S and PSoC Analog Coprocessor family devices, this panel is used to configure the Deep Sleep Timer interrupt. • If you select the auto-generated option, then the CySysTimerIsr() function is registered as the Deep Sleep Timer interrupt handler. In this case, you can use the CySysTimerGetInterruptCallback()/ CySysTimerSetInterruptCallback() functions to Get/Set callbacks for each particular counter and use the CySysTimerEnableIsr()/CySysTimertDisableIsr()functions to Enable/Disable service of the registered callbacks for each particular counter. • If you select the User-provided option, then the CySysTimerIsr() function is not registered as the Deep Sleep Timer interrupt handler. In this case, you can register either some custom handler or the CyTimerIsr() function by using the CyIntSetVector() API. For PSoC 4000 family devices, this panel is hidden by default. The WDT interrupt handler is implemented as user-provided.

Application Programming Interface

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

Modules

- Low Frequency APIs
- Compensating / Trimming
- WCO configuration
- Watchdog Timers configuration
- Single Watchdog Timer configuration
- DeepSleep Timer configuration

Low Frequency APIs

Description

Functions

- void CySysClkIloStart(void)
  Enables ILO.
- void CySysClkIloStop(void)
  Disables the ILO.
- void CySysClkSetLfclkSource(uint32 source)
  Sets the clock source for the LFCLK clock.

Function Documentation

void CySysClkIloStart (void)

Enables ILO.

Refer to the device datasheet for the ILO startup time.

void CySysClkIloStop (void)

Disables the ILO.

This function has no effect if WDT is locked (CySysWdtLock() is called). Call CySysWdtUnlock() to unlock WDT and stop ILO.

PSoC 4100 / PSoC 4200: Note that ILO is required for WDT's operation.

PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M: Stopping ILO affects the peripheral clocked by LFCLK, if LFCLK is configured to be sourced by ILO.

If the ILO is disabled, all blocks run by ILO will stop functioning.

void CySysClkSetLfclkSource (uint32 *source*)

Sets the clock source for the LFCLK clock.

The switch between LFCLK sources must be done between the positive edges of LFCLK, because the glitch risk is around the LFCLK positive edge. To ensure that the switch can be done safely, the WDT counter value is read until it changes.

That means that the positive edge just finished and the switch is performed. The enabled WDT counter is used for that purpose. If no counters are enabled, counter 0 is enabled. And after the LFCLK source is switched, counter 0 configuration is restored.

The function is applicable only for devices with more than one source for LFCLK - PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4100M / PSoC 4200M / PSoC 4200L.

Note:

For PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor devices LFCLK can only be sourced from ILO even though WCO is available.

Parameters:

source	CY_SYS_CLK_LFCLK_SRC_ILO - Internal Low Frequency (32 kHz) Oscillator (ILO). CY_SYS_CLK_LFCLK_SRC_WCO - Low Frequency Watch Crystal Oscillator (WCO).

This function has no effect if WDT is locked (CySysWdtLock() is called). Call CySysWdtUnlock() to unlock WDT.

Both the current source and the new source must be running and stable before calling this function.

Changing the LFCLK clock source may change the LFCLK clock frequency and affect the functionality that uses this clock. For example, watchdog timer "uses this clock" or "this clock uses" (WDT) is clocked by LFCLK.

Compensating / Trimming

Description

Functions

- cystatus CySysClkIloCompensate(uint32 desiredDelay, uint32 *compensatedCycles)
  This API measures the current ILO accuracy.
- void CySysClkIloStartMeasurement(void)
  Starts the ILO accuracy measurement.
- void CySysClkIloStopMeasurement(void)
  Stops the ILO accuracy measurement.
- cystatus CySysClkIloTrim(uint32 mode, int32 *iloAccuracyInPPT)
  The API trims the ILO frequency to +/- 10% accuracy range using accurate SysClk.
- cystatus CySysClkIloRestoreFactoryTrim(void)
  Restores the ILO Trim Register to factory value.

Function Documentation

cystatus CySysClkIloCompensate (uint32 *desiredDelay*, uint32 * *compensatedCycles*)

This API measures the current ILO accuracy.

Basing on the measured frequency the required number of ILO cycles for a given delay (in microseconds) is obtained. The desired delay that needs to be compensated is passed through the desiredDelay parameter. The compensated cycle count is returned through the compesatedCycles pointer. The compensated ILO cycles can

then be used to define the WDT period value, effectively compensating for the ILO inaccuracy and allowing a more accurate WDT interrupt generation.

CySysClkIloStartMeasurement() API should be called prior to calling this API.

Note:

SysClk should be sourced by IMO. Otherwise CySysClkIloTrim() and CySysClkIloCompensate() API can give incorrect results.
If the System clock frequency is changed in runtime, the CyDelayFreq() with the appropriate parameter (Frequency of bus clock in Hertz) should be called before calling a next CySysClkIloCompensate().

Warning:

Do not enter deep sleep mode until the function returns CYRET_SUCCESS.

Parameters:

desiredDelay	Required delay in microseconds.
**compensate dCycles*	The pointer to the variable in which the required number of ILO cycles for the given delay will be returned.

The value returned in *compensatedCycles pointer is not valid until the function returns CYRET_SUCCESS.

The desiredDelay parameter value should be in next range:

From 100 to 2 000 000 microseconds for PSoC 4000 / PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor devices.

From 100 to 4 000 000 000 microseconds for PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M /PSoC 4200M devices.

Returns:

CYRET_SUCCESS - The compensation process is complete and the compensated cycles value is returned in the compensatedCycles pointer.
CYRET_STARTED - Indicates measurement is in progress. It is strongly recommended to do not make pauses between API calling. The function should be called repeatedly until the API returns CYRET_SUCCESS.
CYRET_INVALID_STATE - Indicates that measurement not started. The user should call CySysClkIloStartMeasurement() API before calling this API.

Note:

For a correct WDT or DeepSleep Timers functioning with ILO compensating the CySysClkIloCompensate() should be called before WDT or DeepSleep Timers enabling.

void CySysClkIloStartMeasurement (void)

Starts the ILO accuracy measurement.

This function is non-blocking and needs to be called before using the CySysClkIloTrim() and CySysClkIloCompensate() API.

This API configures measurement counters to be sourced by SysClk (Counter 1) and ILO (Counter 2).

Note:

SysClk should be sourced by IMO. Otherwise CySysClkIloTrim() and CySysClkIloCompensate() API can give incorrect results.

In addition, this API stores the factory ILO trim settings on the first call after reset. This stored factory setting is used by the CySysClkIloRestoreFactoryTrim() API to restore the ILO factory trim. Hence, it is important to call this API before restoring the ILO factory trim settings.

void CySysClkIloStopMeasurement (void)

Stops the ILO accuracy measurement.

Calling this function immediately stops the the ILO frequency measurement. This function should be called before placing the device to deepsleep, if CySysClkIloStartMeasurement() API was called before.

cystatus CySysClkIloTrim (uint32 *mode*, int32 * *iloAccuracyInPPT*)

The API trims the ILO frequency to +/- 10% accuracy range using accurate SysClk.

The API can be blocking or non-blocking depending on the value of the mode parameter passed. The accuracy of ILO after trimming in parts per thousand is returned through the iloAccuracyInPPT pointer. A positive number indicates that the ILO is running fast and a negative number indicates that the ILO is running slowly. This error is relative to the error in the reference clock (SysClk), so the absolute error will be higher and depends on the accuracy of the reference.

The CySysClkIloStartMeasurement() API should be called prior to calling this API. Otherwise it will return CYRET_INVALID_STATE as the measurement was not started.

Note:

SysClk should be sourced by IMO. Otherwise CySysClkIloTrim() and CySysClkIloCompensate() API can give incorrect results.
If System clock frequency is changed in runtime, the CyDelayFreq() with the appropriate parameter (Frequency of bus clock in Hertz) should be called before next CySysClkIloCompensate() usage.

Warning:

Do not enter deep sleep mode until the function returns CYRET_SUCCESS or CYRET_TIMEOUT.

Available for all PSoC 4 devices with ILO trim capability. This excludes PSoC 4000 / PSoC 4100 / PSoC 4200 / PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor devices.

Parameters:

mode	CY_SYS_CLK_BLOCKING - The function does not return until the ILO is within +/-10% accuracy range or time out has occurred. CY_SYS_CLK_NON_BLOCKING - The function returns immediately after performing a single iteration of the trim process. The function should be called repeatedly until the trimming is completed successfully.
**iloAccuracyInPPT*	Pointer to an integer in which the trimmed ILO accuracy will be returned.

The value returned in *iloAccuracyInPPT pointer is not valid until the function returns CYRET_SUCCESS. ILO accuracy in PPT is given by:

IloAccuracyInPPT = ((MeasuredIloFreq - DesiredIloFreq) * CY_SYS_CLK_PERTHOUSAND) / DesiredIloFreq);

DesiredIloFreq = 32000, CY_SYS_CLK_PERTHOUSAND = 1000;

Returns:

CYRET_SUCCESS - Indicates trimming is complete. This value indicates trimming is successful and iloAccuracyInPPT is within +/- 10%.
CYRET_STARTED - Indicates measurement is in progress. This is applicable only for non-blocking mode.
CYRET_INVALID_STATE - Indicates trimming was unsuccessful. You should call CySysClkIloStartMeasurement() before calling this API.
CYRET_TIMEOUT - Indicates trimming was unsuccessful. This is applicable only for blocking mode. Timeout means the trimming was tried 5 times without success (i.e. ILO accuracy > +/- 10%). The user can call the API again for another try or wait for some time before calling it again (to let the system to settle to another operating point change in temperature etc.) and continue using the previous trim value till the next call.

cystatus CySysClkIloRestoreFactoryTrim (void)

Restores the ILO Trim Register to factory value.

The CySysClkIloStartMeasurement() API should be called prior to calling this API. Otherwise CYRET_UNKNOWN will be returned.

Available for all PSoC 4 devices except for PSoC 4000 / PSoC 4100 / PSoC 4200 / PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor devices.

Returns:

CYRET_SUCCESS - Operation was successful.
CYRET_UNKNOWN - CySysClkIloStartMeasurement() was not called before this API. Hence the trim value cannot be updated.

WCO configuration

Description

APIs are available for PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M / PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor

Functions

- void CySysClkWcoStart(void)
  Enables Watch Crystal Oscillator (WCO).
- void CySysClkWcoStop(void)
  Disables the 32 KHz Crystal Oscillator.
- uint32 CySysClkWcoSetPowerMode(uint32 mode)
  Sets the power mode for the 32 KHz WCO.
- void CySysClkWcoClockOutSelect(uint32 clockSel)
  Selects the WCO block output source.

Function Documentation

void CySysClkWcoStart (void)

Enables Watch Crystal Oscillator (WCO).

This API enables WCO which is used as a source for LFCLK. Similar to ILO, WCO is also available in all modes except Hibernate and Stop modes.

Note:

In PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor devices WCO cannot be a source for the LFCLK.

WCO is always enabled in High Power Mode (HPM). Refer to the device datasheet for the WCO startup time. Once WCO becomes stable it can be switched to Low Power Mode (LPM). Note that oscillator can be unstable during a switch and hence its output should not be used at that moment.

The CySysClkWcoSetPowerMode() function configures the WCO power mode.

void CySysClkWcoStop (void)

Disables the 32 KHz Crystal Oscillator.

API switch of WCO.

Note:

PSoC 4100S / PSoC Analog Coprocessor: WCO is required for DeepSleep Timer's operation.

uint32 CySysClkWcoSetPowerMode (uint32 *mode*)

Sets the power mode for the 32 KHz WCO.

By default (if this function is not called), the WCO is in High power mode during Active and device's low power modes

Parameters:

mode	CY_SYS_CLK_WCO_HPM - The High Power mode. CY_SYS_CLK_WCO_LPM - The Low Power mode.

Returns:

A previous power mode. The same as the parameters.

Note:

The WCO Low Power mode is applicable for PSoC 4100 BLE / PSoC 4200 BLE devices.

void CySysClkWcoClockOutSelect (uint32 *clockSel*)

Selects the WCO block output source.

In addition to generating 32.768 kHz clock from external crystals, WCO can be sourced by external clock source using wco_out pin. The API help to lets you select between the sources: External crystal or external pin.

If you want to use external pin to drive WCO the next procedure is required:

1) Disable the WCO.

2) Drive the wco_out pin to an external signal source.

3) Call CySysClkWcoClockOutSelect(CY_SYS_CLK_WCO_SEL_PIN).

4) Enable the WCO and wait for 15 us before clocking the XO pad at the high potential. Let's assume you are using the 1.6v clock amplitude, then the sequence would start at 1.6v, then 0v, then 1.6v etc at a chosen frequency.

If you want to use WCO after using an external pin source:

1) Disable the WCO.

2) Drive off wco_out pin with external signal source.

3) Call CySysClkWcoClockOutSelect(CY_SYS_CLK_WCO_SEL_CRYSTAL).

4) Enable the WCO.

Warning:

Do not use the oscillator output clock prior to a 15uS delay in your system. There are no limitations on the external clock frequency.
When external clock source was selected to drive WCO block the IMO can be trimmed only when external clock source period is equal to WCO external crystal period. Also external clock source accuracy should be higher or equal to WCO external crystal accuracy.

Parameters:

clockSel	CY_SYS_CLK_WCO_SEL_CRYSTAL - Selects External crystal as clock source of WCO. CY_SYS_CLK_WCO_SEL_PIN - Selects External clock input on wco_in pin as clock source of WCO.

Watchdog Timers configuration

Description

APIs are available for PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE /PSoC 4200L / PSoC 4100M / PSoC 4200M

Functions

- void CySysWdtLock(void)
  Locks out configuration changes to the Watchdog timer registers and ILO configuration register.

- void CySysWdtUnlock(void)
  Unlocks the Watchdog Timer configuration register.
- void CySysWdtSetMode(uint32 counterNum, uint32 mode)
  Writes the mode of one of the three WDT counters.
- uint32 CySysWdtGetMode(uint32 counterNum)
  Reads the mode of one of the three WDT counters.
- uint32 CySysWdtGetEnabledStatus(uint32 counterNum)
  Reads the enabled status of one of the three WDT counters.
- void CySysWdtSetClearOnMatch(uint32 counterNum, uint32 enable)
  Configures the WDT counter "clear on match" setting.
- uint32 CySysWdtGetClearOnMatch(uint32 counterNum)
  Reads the "clear on match" setting for the specified counter.
- void CySysWdtEnable(uint32 counterMask)
  Enables the specified WDT counters.
- void CySysWdtDisable(uint32 counterMask)
  Disables the specified WDT counters. All the counters specified in the mask are disabled. The function waits for the changes to come into effect.
- void CySysWdtSetCascade(uint32 cascadeMask)
  Writes the two WDT cascade values based on the combination of mask values specified.
- uint32 CySysWdtGetCascade(void)
  Reads the two WDT cascade values returning a mask of the bits set.
- void CySysWdtSetMatch(uint32 counterNum, uint32 match)
  Configures the WDT counter match comparison value.
- void CySysWdtSetToggleBit(uint32 bits)
  Configures which bit in WDT counter 2 to monitor for a toggle.
- uint32 CySysWdtGetToggleBit(void)
  Reads which bit in WDT counter 2 is monitored for a toggle.
- uint32 CySysWdtGetMatch(uint32 counterNum)
  Reads the WDT counter match comparison value.
- uint32 CySysWdtGetCount(uint32 counterNum)
  Reads the current WDT counter value.
- uint32 CySysWdtGetInterruptSource(void)
  Reads a mask containing all the WDT counters interrupts that are currently set by the hardware, if a corresponding mode is selected.
- void CySysWdtClearInterrupt(uint32 counterMask)
  Clears all the WDT counter interrupts set in the mask.
- void CySysWdtResetCounters(uint32 countersMask)
  Resets all the WDT counters set in the mask.
- cyWdtCallback CySysWdtSetInterruptCallback(uint32 counterNum, cyWdtCallback function)
  Sets the ISR callback function for the particular WDT counter. These functions are called on the WDT interrupt.
- cyWdtCallback CySysWdtGetInterruptCallback(uint32 counterNum)
  Gets the ISR callback function for the particular WDT counter.
- void CySysTimerDelay(uint32 counterNum, cy_sys_timer_delaytype_enum delayType, uint32 delay)
  The function implements the delay specified in the LFCLK clock ticks.
- void CySysTimerDelayUntilMatch(uint32 counterNum, cy_sys_timer_delaytype_enum delayType, uint32 match)

The function implements the delay specified as the number of WDT or DeepSleep Timer clock source ticks between WDT or DeepSleep Timer current value and match" value.

- void CySysWatchdogFeed(uint32 counterNum)
  Feeds the corresponded Watchdog Counter before it causes the device reset.
- void CySysWdtEnableCounterIsr(uint32 counterNum)
  Enables the ISR callback servicing for the particular WDT counter.
- void CySysWdtDisableCounterIsr(uint32 counterNum)
  Disables the ISR callback servicing for the particular WDT counter.
- void CySysWdtIsr(void)
  This is the handler of the WDT interrupt in CPU NVIC.

Function Documentation

void CySysWdtLock (void)

Locks out configuration changes to the Watchdog timer registers and ILO configuration register.

After this function is called, ILO clock can't be disabled until CySysWdtUnlock() is called.

void CySysWdtSetMode (uint32 *counterNum*, uint32 *mode*)

Writes the mode of one of the three WDT counters.

Parameters:

counterNum	Valid range [0-2]. The number of the WDT counter.
mode	CY_SYS_WDT_MODE_NONE - Free running. CY_SYS_WDT_MODE_INT - The interrupt generated on match for counter 0 and 1, and on bit toggle for counter 2. CY_SYS_WDT_MODE_RESET - Reset on match (valid for counter 0 and 1 only). CY_SYS_WDT_MODE_INT_RESET - Generate an interrupt. Generate a reset on the 3rd non-handled interrupt (valid for counter 0 and counter 1 only).

WDT counter counterNum should be disabled to set a mode. Otherwise, this function call has no effect. If the specified counter is enabled, call the CySysWdtDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop.

uint32 CySysWdtGetMode (uint32 *counterNum*)

Reads the mode of one of the three WDT counters.

Parameters:

counterNum	Valid range [0-2]. The number of the WDT counter.

Returns:

The mode of the counter. The same enumerated values as the mode parameter used in CySysWdtSetMode().

uint32 CySysWdtGetEnabledStatus (uint32 *counterNum*)

Reads the enabled status of one of the three WDT counters.

Parameters:

counterNum	Valid range [0-2]. The number of the WDT counter.

Returns:

The status of the WDT counter:
0 - If the counter is disabled.
1 - If the counter is enabled.

This function returns an actual WDT counter status from the status register. It may take up to 3 LFCLK cycles for the WDT status register to contain actual data after the WDT counter is enabled.

void CySysWdtSetClearOnMatch (uint32 *counterNum*, uint32 *enable*)

Configures the WDT counter "clear on match" setting.

If configured to "clear on match", the counter counts from 0 to MatchValue giving it a period of (MatchValue + 1).

Parameters:

counterNum	Valid range [0-1]. The number of the WDT counter. The match values are not supported by counter 2.
enable	0 to disable appropriate counter 1 to enable appropriate counter

WDT counter counterNum should be disabled. Otherwise this function call has no effect. If the specified counter is enabled, call the CySysWdtDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop. This may take up to three LFCLK cycles.

uint32 CySysWdtGetClearOnMatch (uint32 *counterNum*)

Reads the "clear on match" setting for the specified counter.

Parameters:

counterNum	Valid range [0-1]. The number of the WDT counter. The match values are not supported by counter 2.

Returns:

The "clear on match" status:
1 if enabled
0 if disabled

void CySysWdtEnable (uint32 *counterMask*)

Enables the specified WDT counters.

All the counters specified in the mask are enabled.

Parameters:

counterMask	CY_SYS_WDT_COUNTER0_MASK - The mask for counter 0 to enable. CY_SYS_WDT_COUNTER1_MASK - The mask for counter 1 to enable. CY_SYS_WDT_COUNTER2_MASK - The mask for counter 2 to enable.

Enabling or disabling WDT requires 3 LFCLK cycles to come into effect. Therefore, the WDT enable state must not be changed more than once in that period.

After WDT is enabled, it is illegal to write WDT configuration (WDT_CONFIG) and control (WDT_CONTROL) registers. This means that all WDT functions that contain 'write' in the name (with the exception of CySysWdtSetMatch() function) are illegal to call if WDT is enabled.

PSoC 4100 / PSoC 4200: This function enables ILO.

PSoC 4100 BLE / PSoC 4200 BLE / PSoC4200L / PSoC 4100M / PSoC 4200M: LFLCK should be configured before calling this function. The desired source should be enabled and configured to be the source for LFCLK.

void CySysWdtDisable (uint32 *counterMask*)

Disables the specified WDT counters. All the counters specified in the mask are disabled. The function waits for the changes to come into effect.

Parameters:

counterMask	CY_SYS_WDT_COUNTER0_MASK - The mask for counter 0 to disable. CY_SYS_WDT_COUNTER1_MASK - The mask for counter 1 to disable. CY_SYS_WDT_COUNTER2_MASK - The mask for counter 2 to disable.

void CySysWdtSetCascade (uint32 *cascadeMask*)

Writes the two WDT cascade values based on the combination of mask values specified.

Parameters:

cascadeMask	The mask value used to set or clear the cascade values: CY_SYS_WDT_CASCADE_NONE - Neither CY_SYS_WDT_CASCADE_01 - Cascade 01 CY_SYS_WDT_CASCADE_12 - Cascade 12

If only one cascade mask is specified, the second cascade is disabled. To set both cascade modes, two defines should be ORed: (CY_SYS_TIMER_CASCADE_01 | CY_SYS_TIMER_CASCADE_12).

Note:

If CySysWdtSetCascade() was called with ORed defines it is necessary to call CySysWdtSetClearOnMatch(1,1). It is needed to make sure that Counter 2 will be updated in the expected way.

WDT counters that are part of the specified cascade should be disabled. Otherwise this function call has no effect. If the specified counter is enabled, call CySysWdtDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop. This may take up to 3 LFCLK cycles.

uint32 CySysWdtGetCascade (void)

Reads the two WDT cascade values returning a mask of the bits set.

Returns:

The mask of the cascade values set.
CY_SYS_WDT_CASCADE_NONE - Neither
CY_SYS_WDT_CASCADE_01 - Cascade 01
CY_SYS_WDT_CASCADE_12 - Cascade 12

void CySysWdtSetMatch (uint32 *counterNum*, uint32 *match*)

Configures the WDT counter match comparison value.

Parameters:

counterNum	Valid range [0-1]. The number of the WDT counter. The match values are not supported by counter 2.

match	Valid range [0-65535]. The value to be used to match against the counter.

void CySysWdtSetToggleBit (uint32 *bits*)

Configures which bit in WDT counter 2 to monitor for a toggle.

When that bit toggles, an interrupt is generated if the mode for counter 2 has enabled interrupts.

Parameters:

bits	Valid range [0-31]. Counter 2 bit to monitor for a toggle.

WDT Counter 2 should be disabled. Otherwise this function call has no effect.

If the specified counter is enabled, call the CySysWdtDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop. This may take up to 3 LFCLK cycles.

uint32 CySysWdtGetToggleBit (void)

Reads which bit in WDT counter 2 is monitored for a toggle.

Returns:

The bit that is monitored (range of 0 to 31)

uint32 CySysWdtGetMatch (uint32 *counterNum*)

Reads the WDT counter match comparison value.

Parameters:

counterNum	Valid range [0-1]. The number of the WDT counter. The match values are not supported by counter 2.

Returns:

A 16-bit match value.

uint32 CySysWdtGetCount (uint32 *counterNum*)

Reads the current WDT counter value.

Parameters:

counterNum	Valid range [0-2]. The number of the WDT counter.

Returns:

A live counter value. Counter 0 and Counter 1 are 16 bit counters and counter 2 is a 32 bit counter.

uint32 CySysWdtGetInterruptSource (void)

Reads a mask containing all the WDT counters interrupts that are currently set by the hardware, if a corresponding mode is selected.

Returns:

The mask of interrupts set
CY_SYS_WDT_COUNTER0_INT - Counter 0
CY_SYS_WDT_COUNTER1_INT - Counter 1
CY_SYS_WDT_COUNTER2_INT - Counter 2

void CySysWdtClearInterrupt (uint32 *counterMask*)

Clears all the WDT counter interrupts set in the mask.

Calling this function also prevents from Reset when the counter mode is set to generate 3 interrupts and then the device resets.

All the WDT interrupts are to be cleared by the firmware, otherwise interrupts are generated continuously.

Parameters:

counterMask	CY_SYS_WDT_COUNTER0_INT - Clears counter 0 interrupts CY_SYS_WDT_COUNTER1_INT - Clears counter 1 interrupts CY_SYS_WDT_COUNTER2_INT - Clears counter 2 interrupts

This function temporary removes the watchdog lock, if it was set, and restores the lock state after cleaning the WDT interrupts that are set in a mask.

void CySysWdtResetCounters (uint32 *countersMask*)

Resets all the WDT counters set in the mask.

Parameters:

countersMask	CY_SYS_WDT_COUNTER0_RESET - Reset counter 0 CY_SYS_WDT_COUNTER1_RESET - Reset counter 1 CY_SYS_WDT_COUNTER2_RESET - Reset counter 2

This function does not reset counter values if the Watchdog is locked. This function waits while corresponding counters will be reset. This may take up to 3 LFCLK cycles. The LFCLK source must be enabled. Otherwise, the function will never exit.

cyWdtCallback CySysWdtSetInterruptCallback (uint32 *counterNum*, cyWdtCallback *function*)

Sets the ISR callback function for the particular WDT counter. These functions are called on the WDT interrupt.

Parameters:

counterNum	The number of the WDT counter.
function	The pointer to the callback function.

Returns:

The pointer to the previous callback function.
NULL is returned if the specified address is not set.

cyWdtCallback CySysWdtGetInterruptCallback (uint32 *counterNum*)

Gets the ISR callback function for the particular WDT counter.

Parameters:

counterNum	The number of the WDT counter.

Returns:

The pointer to the callback function registered for a particular WDT by a particular address that are passed through arguments.

void CySysTimerDelay (uint32 *counterNum*, cy_sys_timer_delaytype_enum *delayType*, uint32 *delay*)

The function implements the delay specified in the LFCLK clock ticks.

This API is applicable for PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices to use WDT. Also this API is available to use for PSoC4100S and / PSoC Analog Coprocessor devices to use DeepSleep Timers.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices: The specified WDT counter should be configured as described below and started.

For PSoC 4100S / PSoC Analog Coprocessor devices: The specified DeepSleep Timer counter should be configured as described below and started.

This function can operate in two modes: the "WAIT" and "INTERRUPT" modes. In the "WAIT" mode, the function waits for the specified number of ticks. In the "INTERRUPT" mode, the interrupt is generated after the specified number of ticks.

For the correct function operation, the "Clear On Match" option should be disabled for the specified WDT or DeepSleep Timer counter. Use CySysWdtSetClearOnMatch() for WDT or CySysTimerSetClearOnMatch() for DeepSleep Timer function with the "enable" parameter equal to zero for the used WDT counter or DeepSleep Timer counter.

The corresponding WDT counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" / "Watchdog (w/Interrupt)" for the "INTERRUPT" mode.

Or the corresponding DeepSleep Timer counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" for the "INTERRUPT" mode.

This can be configured in two ways:

- Through the DWR page. Open the "Clocks" tab, click the "Edit Clocks..." button, in the "Configure System Clocks" window click on the "Low Frequency Clocks" tab and choose the appropriate option for the used WDT or DeepSleep Timer counter.
- Through the CySysWdtSetMode() for WDT or CySysTimerSetMode() for DeepSleep Timer function. Call it with the appropriate "mode" parameter for the used WDT or DeepSleep Timer counter.

For the "INTERRUPT" mode, the recommended sequence is the following:

- Call the CySysWdtDisableCounterIsr() for WDT or CySysTimerDisableIsr() for DeepSleep Timer function to disable servicing interrupts of the specified WDT or DeepSleep Timer counter.
- Call the CySysWdtSetInterruptCallback() for WDT or CySysTimerSetIsrCallback() for DeepSleep Timer function to register the callback function for the corresponding WDT or DeepSleep Timer counter.
- Call the CySysTimerDelay() function.

Parameters:

counterNum	Valid range [0-1]. The number of the counter (Timer0 or Timer1).
delayType	CY_SYS_TIMER_WAIT - "WAIT" mode. CY_SYS_TIMER_INTERRUPT - "INTERRUPT" mode.
delay	The delay value in the LFCLK ticks (allowable range - 16-bit value).

In the "INTERRUPT" mode, this function enables ISR callback servicing from the corresponding WDT or DeepSleep Timer counter. Servicing of this ISR callback will be disabled after the expiration of the delay time.

void CySysTimerDelayUntilMatch (uint32 *counterNum*, cy_sys_timer_delaytype_enum *delayType*, uint32 *match*)

The function implements the delay specified as the number of WDT or DeepSleep Timer clock source ticks between WDT or DeepSleep Timer current value and match" value.

This API is applicable for PSoC 4100 / PSoC 4200 / PRoC BLE / PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices to use WDT. Also this API is available to use for PSoC4100S / Analog Coprocessor devices to use DeepSleep Timers.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices: The function implements the delay specified as the number of LFCLK ticks between the specified WDT counter's current value and the "match" passed as the parameter to this function. The current WDT counter value can be obtained using the CySysWdtGetCount() function.

For PSoC4100 S and Analog Coprocessor devices: The function implements the delay specified as the number of DeepSleep Timer input clock ticks for Timer0/Timer1 counter's current value and the "match" passed as the parameter to this function. The current DeepSleep Timer counter value can be obtained using the CySysWdtGetCount() function.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices: The specified WDT counter should be configured as described below and started.

For PSoC PSoC 4100S / PSoC Analog Coprocessor devices: The specified DeepSleep Timer counter should be configured as described below and started.

This function can operate in two modes: the "WAIT" and "INTERRUPT" modes. In the "WAIT" mode, the function waits for the specified number of ticks. In the "INTERRUPT" mode, the interrupt is generated after the specified number of ticks.

For the correct function operation, the "Clear On Match" option should be disabled for the specified WDT or DeepSleep Timer counter. Use CySysWdtSetClearOnMatch() for WDT or CySysTimerSetClearOnMatch() for DeepSleep Timer function with the "enable" parameter equal to zero for the used WDT or DeepSleep Timer counter.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M/PSoC 4200M devices: The corresponding WDT counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" / "Watchdog (w/Interrupt)" for the "INTERRUPT" mode.

For PSoC 4100S / PSoC Analog Coprocessor devices: Corresponding DeepSleep Timer counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" for the "INTERRUPT" mode.

This can be configured in two ways:

- Through the DWR page. Open the "Clocks" tab, click the "Edit Clocks..." button, in the "Configure System Clocks" window click on the "Low Frequency Clocks" tab and choose the appropriate option for the used WDT or DeepSleep Timer counter.
- Through the CySysWdtSetMode() for WDT or CySysTimerSetMode() for DeepSleep Timer function. Call it with the appropriate "mode" parameter for the used WDT or DeepSleep Timer counter.

For the "INTERRUPT" mode, the recommended sequence is the following:

- Call the CySysWdtDisableCounterIsr() for WDT or CySysTimerDisableIsr() for DeepSleep Timer function to disable servicing interrupts of the specified WDT or DeepSleep Timer counter.
- Call the CySysWdtSetInterruptCallback() for WDT or CySysTimerSetInterruptCallback() for DeepSleep Timer function to register the callback function for the corresponding WDT or DeepSleep Timer counter.
- Call the CySysTimerDelay() function.

Parameters:

counterNum	Valid range [0-1]. The number of the WDT or DeepSleep Timer. counter (Timer0 or Timer1).
delayType	CY_SYS_TIMER_WAIT - "WAIT" mode. CY_SYS_TIMER_INTERRUPT - "INTERRUPT" mode.
delay	The delay value in the LFCLK ticks (allowable range - 16-bit value).

In the "INTERRUPT" mode, this function enables ISR callback servicing from the corresponding WDT counter. Servicing of this ISR callback will be disabled after the expiration of the delay time.

void CySysWatchdogFeed (uint32 *counterNum*)

Feeds the corresponded Watchdog Counter before it causes the device reset.

Supported only for first WDT0 and second WDT1 counters in the "Watchdog" or "Watchdog w/ Interrupts" modes.

Parameters:

counterNum	CY_SYS_WDT_COUNTER0 - Feeds the Counter 0 CY_SYS_WDT_COUNTER1 - Feeds the Counter 1

Value of counterNum corresponds to appropriate counter. For example value 1 corresponds to second WDT1 Counter.

Clears the WDT counter in the "Watchdog" mode or clears the WDT interrupt in "Watchdog w/ Interrupts" mode. Does nothing in other modes.

void CySysWdtEnableCounterIsr (uint32 *counterNum*)

Enables the ISR callback servicing for the particular WDT counter.

Parameters:

counterNum	Valid range [0-2]. The number of the WDT counter.

Value corresponds to appropriate WDT counter. For example value 1 corresponds to second WDT counter.

void CySysWdtDisableCounterIsr (uint32 *counterNum*)

Disables the ISR callback servicing for the particular WDT counter.

Parameters:

counterNum	Valid range [0-2]. The number of the WDT counter.

void CySysWdtIsr (void)

This is the handler of the WDT interrupt in CPU NVIC.

The handler checks which WDT triggered in the interrupt and calls the respective callback functions configured by the user by using CySysWdtSetIsrCallback() API.

The order of the callback execution is incremental. Callback-0 is run as the first one and callback-2 is called as the last one.

This function clears the WDT interrupt every time when it is called. Reset after the 3rd interrupt does not happen if this function is registered as the interrupt handler even if the "Watchdog with Interrupt" mode is selected on the "Low Frequency Clocks" tab.

The handler calls the respective callback functions configured by the user by using CySysWdtSetIsrCallback() API.

This function clears the WDT interrupt every time when it is called. Reset after the 3rd interrupt does not happen if this function is registered as the interrupt handler even if the "Watchdog with Interrupt" mode is selected on the "Low Frequency Clocks" tab.

Single Watchdog Timer configuration

Description

APIs are available for PSoC 4000 / PSoC 4000S / PSoC4100S / PSoC Analog Coprocessor

Functions

- uint32 CySysWdtGetEnabledStatus(void)
  Reads the enabled status of the WDT counter.
- void CySysWdtEnable(void)
  Enables watchdog timer reset generation.
- void CySysWdtDisable(void)
  Disables the WDT reset generation.
- void CySysWdtSetMatch(uint32 match)
  Configures the WDT counter match comparison value.
- uint32 CySysWdtGetMatch(void)
  Reads the WDT counter match comparison value.

- uint32 CySysWdtGetCount(void)
  Reads the current WDT counter value.
- void CySysWdtSetIgnoreBits(uint32 bitsNum)
  Configures the number of the MSB bits of the watchdog timer that are not checked against the match.
- uint32 CySysWdtGetIgnoreBits(void)
  Reads the number of the MSB bits of the watchdog timer that are not checked against the match.
- void CySysWdtClearInterrupt(void)
  Feeds the watchdog. Cleans the WDT match flag which is set every time the WDT counter reaches a WDT match value. Two unserviced interrupts lead to a system reset (i.e. at the third match).
- void CySysWdtMaskInterrupt(void)
  After masking interrupts from WDT, they are not passed to CPU. This function does not disable WDT reset generation.
- void CySysWdtUnmaskInterrupt(void)
  After unmasking interrupts from WDT, they are passed to CPU. This function does not impact the reset generation.
- cyWdtCallback CySysWdtSetInterruptCallback(cyWdtCallback function)
  Sets the ISR callback function for the WDT counter.
- cyWdtCallback CySysWdtGetInterruptCallback(void)
  Gets the ISR callback function for the WDT counter.
- void CySysWdtIsr(void)
  This is the handler of the WDT interrupt in CPU NVIC.

Function Documentation

uint32 CySysWdtGetEnabledStatus (void)

Reads the enabled status of the WDT counter.

Returns:

The status of the WDT counter:
0 - Counter is disabled
1 - Counter is enabled

void CySysWdtEnable (void)

Enables watchdog timer reset generation.

CySysWdtClearInterrupt() feeds the watchdog. Two unserviced interrupts lead to a system reset (i.e. at the third match).

ILO is enabled by the hardware once WDT is started.

void CySysWdtDisable (void)

Disables the WDT reset generation.

This function unlocks the ENABLE bit in the CLK_ILO_CONFIG registers and enables the user to disable ILO.

void CySysWdtSetMatch (uint32 *match*)

Configures the WDT counter match comparison value.

Parameters:

match	Valid range [0-65535]. The value to be used to match against the counter.

uint32 CySysWdtGetMatch (void)

Reads the WDT counter match comparison value.

Returns:

The counter match value.

uint32 CySysWdtGetCount (void)

Reads the current WDT counter value.

Returns:

A live counter value.

void CySysWdtSetIgnoreBits (uint32 *bitsNum*)

Configures the number of the MSB bits of the watchdog timer that are not checked against the match.

Parameters:

bitsNum	Valid range [0-15]. The number of the MSB bits.

The value of bitsNum controls the time-to-reset of the watchdog (which happens after 3 successive matches).

uint32 CySysWdtGetIgnoreBits (void)

Reads the number of the MSB bits of the watchdog timer that are not checked against the match.

Returns:

The number of the MSB bits.

cyWdtCallback CySysWdtSetInterruptCallback (cyWdtCallback *function*)

Sets the ISR callback function for the WDT counter.

Parameters:

function	The pointer to the callback function.

Returns:

The pointer to a previous callback function.

cyWdtCallback CySysWdtGetInterruptCallback (void)

Gets the ISR callback function for the WDT counter.

Returns:

The pointer to the callback function registered for WDT.

void CySysWdtIsr (void)

This is the handler of the WDT interrupt in CPU NVIC.

The handler checks which WDT triggered in the interrupt and calls the respective callback functions configured by the user by using CySysWdtSetIsrCallback() API.

The order of the callback execution is incremental. Callback-0 is run as the first one and callback-2 is called as the last one.

This function clears the WDT interrupt every time when it is called. Reset after the 3rd interrupt does not happen if this function is registered as the interrupt handler even if the "Watchdog with Interrupt" mode is selected on the "Low Frequency Clocks" tab.

The handler calls the respective callback functions configured by the user by using CySysWdtSetIsrCallback() API.

This function clears the WDT interrupt every time when it is called. Reset after the 3rd interrupt does not happen if this function is registered as the interrupt handler even if the "Watchdog with Interrupt" mode is selected on the "Low Frequency Clocks" tab.

DeepSleep Timer configuration

Description

APIs are available for PSoC 4100S / PSoC Analog Coprocessor

Functions

- void CySysClkSetTimerSource(uint32 source)
  Sets the clock source for the DeepSleep Timers.
- void CySysTimerSetMode(uint32 counterNum, uint32 mode)
  Writes the mode of one of the three DeepSleep Timer counters.
- uint32 CySysTimerGetMode(uint32 counterNum)
  Reads the mode of one of the three DeepSleep Timer counters.
- uint32 CySysTimerGetEnabledStatus(uint32 counterNum)
  Reads the enabled status of one of the three DeepSleep Timer counters.
- void CySysTimerSetClearOnMatch(uint32 counterNum, uint32 enable)
  Configures the DeepSleep Timer counter "clear on match" setting.
- uint32 CySysTimerGetClearOnMatch(uint32 counterNum)
  Reads the "clear on match" setting for the specified DeepSleep Timer counter.
- void CySysTimerEnable(uint32 counterMask)
  Enables the specified DeepSleep Timer counters. All the counters specified in the mask are enabled.
- void CySysTimerDisable(uint32 counterMask)
  Disables the specified DeepSleep Timer counters.
- void CySysTimerSetCascade(uint32 cascadeMask)
  Writes the two DeepSleep Timers cascade values based on the combination of mask values specified.
- uint32 CySysTimerGetCascade(void)
  Reads the two DeepSleep Timer cascade values returning a mask of the bits set.
- void CySysTimerSetMatch(uint32 counterNum, uint32 match)
  Configures the Timer counter match comparison value.
- void CySysTimerSetToggleBit(uint32 bits)
  Configures which bit in Timer counter 2 to monitor for a toggle.
- uint32 CySysTimerGetToggleBit(void)
  Reads which bit in Timer counter 2 is monitored for a toggle.
- uint32 CySysTimerGetMatch(uint32 counterNum)
  Reads the Timer counter match comparison value.
- uint32 CySysTimerGetCount(uint32 counterNum)

Reads the current DeepSleep Timer counter value.

- uint32 CySysTimerGetInterruptSource(void)
  Reads a mask containing all the DeepSleep Timer counters interrupts that are currently set by the hardware, if a corresponding mode is selected.
- void CySysTimerClearInterrupt(uint32 counterMask)
  Clears all the DeepSleep Timer counter interrupts set in the mask.
- cyTimerCallback CySysTimerSetInterruptCallback(uint32 counterNum, cyTimerCallback function)
  Sets the ISR callback function for the particular DeepSleep Timer counter.
- cyTimerCallback CySysTimerGetInterruptCallback(uint32 counterNum)
  Gets the ISR callback function for the particular DeepSleep Timer counter.
- void CySysTimerResetCounters(uint32 countersMask)
  Resets all the Timer counters set in the mask.
- void CySysTimerEnableIsr(uint32 counterNum)
  Enables the ISR callback servicing for the particular Timer counter.
- void CySysTimerDisableIsr(uint32 counterNum)
  Disables the ISR callback servicing for the particular Timer counter.
- void CySysTimerIsr(void)
  This is the handler of the DeepSleep Timer interrupt in CPU NVIC.
- void CySysTimerDelay(uint32 counterNum, cy_sys_timer_delaytype_enum delayType, uint32 delay)
  The function implements the delay specified in the LFCLK clock ticks.
- void CySysTimerDelayUntilMatch(uint32 counterNum, cy_sys_timer_delaytype_enum delayType, uint32 match)
  The function implements the delay specified as the number of WDT or DeepSleep Timer clock source ticks between WDT or DeepSleep Timer current value and match" value.

Function Documentation

void CySysClkSetTimerSource (uint32 *source*)

Sets the clock source for the DeepSleep Timers.

The function is applicable only for PSoC 4100S / PSoC Analog Coprocessor devices.

Parameters:

source	CY_SYS_CLK_TIMER_SRC_ILO - Internal Low Frequency (32 kHz) Oscillator (ILO). CY_SYS_CLK_TIMER_SRC_WCO - Low Frequency Watch Crystal Oscillator (WCO).

Both the current source and the new source must be running and stable before calling this function.

Warning:

DeepSleep Timer reset is required if Timer source was switched while DeepSleep Timers were running. Call CySysTimerResetCounters() API after Timer source switching. It is highly recommended to disable DeepSleep Timers before Timer source switching. Changing the Timer source may change the functionality that uses this Timers as clock source.

void CySysTimerSetMode (uint32 *counterNum*, uint32 *mode*)

Writes the mode of one of the three DeepSleep Timer counters.

Parameters:

counterNum	Valid range [0-2]. The number of the DeepSleep Timer counter.
mode	CY_SYS_TIMER_MODE_NONE - Free running.

	CY_SYS_TIMER_MODE_INT - The interrupt generated on match for counter 0 and 1, and on bit toggle for counter 2.

DeepSleep Timer counter counterNum should be disabled to set a mode. Otherwise, this function call has no effect. If the specified counter is enabled, call the CySysTimerDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop.

uint32 CySysTimerGetMode (uint32 *counterNum*)

Reads the mode of one of the three DeepSleep Timer counters.

Parameters:

counterNum	Valid range [0-2]. The number of the Timer counter.

Returns:

The mode of the counter. The same enumerated values as the mode parameter used in CySysTimerSetMode().

uint32 CySysTimerGetEnabledStatus (uint32 *counterNum*)

Reads the enabled status of one of the three DeepSleep Timer counters.

Parameters:

counterNum	Valid range [0-2]. The number of the DeepSleep Timer counter.

Returns:

The status of the Timers counter:
0 - If the Counter is disabled.
1 - If the Counter is enabled.

This function returns an actual DeepSleep Timer counter status from the status register. It may take up to 3 LFCLK cycles for the Timer status register to contain actual data after the Timer counter is enabled.

void CySysTimerSetClearOnMatch (uint32 *counterNum*, uint32 *enable*)

Configures the DeepSleep Timer counter "clear on match" setting.

If configured to "clear on match", the counter counts from 0 to MatchValue giving it a period of (MatchValue + 1).

Parameters:

counterNum	Valid range [0-1]. The number of the Timer counter. The match values are not supported by counter 2.
enable	0 to disable appropriate counter 1 to enable appropriate counter

Timer counter counterNum should be disabled. Otherwise this function call has no effect. If the specified counter is enabled, call the CySysTimerDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop. This may take up to three Timer source-cycles.

uint32 CySysTimerGetClearOnMatch (uint32 *counterNum*)

Reads the "clear on match" setting for the specified DeepSleep Timer counter.

Parameters:

counterNum	Valid range [0-1]. The number of the Timer counter. The match values are not supported by counter 2.

Returns:

The "clear on match" status:
1 if enabled

0 if disabled

void CySysTimerEnable (uint32 *counterMask*)

Enables the specified DeepSleep Timer counters. All the counters specified in the mask are enabled.

Parameters:

counterMask	CY_SYS_TIMER0_MASK - The mask for counter 0 to enable. CY_SYS_TIMER1_MASK - The mask for counter 1 to enable. CY_SYS_TIMER2_MASK - The mask for counter 2 to enable.

Enabling or disabling Timer requires 3 Timer source-cycles to come into effect. Therefore, the Timer enable state must not be changed more than once in that period.

After Timer is enabled, it is illegal to write Timer configuration (WCO_WDT_CONFIG) and control (WCO_WDT_CONTROL) registers. This means that all Timer functions that contain 'write' in the name (with the exception of CySysTimerSetMatch() function) are illegal to call once Timer enabled.

Timer current source must be running and stable before calling this function.

void CySysTimerDisable (uint32 *counterMask*)

Disables the specified DeepSleep Timer counters.

All the counters specified in the mask are disabled. The function waits for the changes to come into effect.

Parameters:

counterMask	CY_SYS_TIMER0_MASK - The mask for Counter 0 to disable. CY_SYS_TIMER1_MASK - The mask for Counter 1 to disable. CY_SYS_TIMER2_MASK - The mask for Counter 2 to disable.

void CySysTimerSetCascade (uint32 *cascadeMask*)

Writes the two DeepSleep Timers cascade values based on the combination of mask values specified.

Parameters:

cascadeMask	The mask value used to set or clear the cascade values: CY_SYS_TIMER_CASCADE_NONE - Neither CY_SYS_TIMER_CASCADE_01 - Cascade 01 CY_SYS_TIMER_CASCADE_12 - Cascade 12

If only one cascade mask is specified, the second cascade is disabled. To set both cascade modes, two defines should be ORed: (CY_SYS_TIMER_CASCADE_01 | CY_SYS_TIMER_CASCADE_12).

Note:

If CySysTimerSetCascade() was called with ORed defines it is necessary to call CySysTimeSetClearOnMatch(1,1). It is needed to make sure that Counter 2 will be updated in the expected way.

Timer counters that are part of the specified cascade should be disabled. Otherwise this function call has no effect. If the specified counter is enabled, call CySysTimerDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop. This may take up to 3 Timers source-cycles.

uint32 CySysTimerGetCascade (void)

Reads the two DeepSleep Timer cascade values returning a mask of the bits set.

Returns:

The mask of the cascade values set.
CY_SYS_TIMER_CASCADE_NONE - Neither

CY_SYS_TIMER_CASCADE_01 - Cascade 01
CY_SYS_TIMER_CASCADE_12 - Cascade 12

void CySysTimerSetMatch (uint32 *counterNum*, uint32 *match*)

Configures the Timer counter match comparison value.

Parameters:

counterNum	Valid range [0-1]. The number of the Timer counter. The match values are not supported by counter 2.
match	Valid range [0-65535]. The value to be used to match against the counter.

void CySysTimerSetToggleBit (uint32 *bits*)

Configures which bit in Timer counter 2 to monitor for a toggle.

When that bit toggles, an interrupt is generated if mode for counter 2 has enabled interrupts.

Parameters:

bits	Valid range [0-31]. Counter 2 bit to monitor for a toggle.

Timer counter 2 should be disabled. Otherwise this function call has no effect.

If the specified counter is enabled, call the CySysTimerDisable() function with the corresponding parameter to disable the specified counter and wait for it to stop. This may take up to three Timer source-cycles.

uint32 CySysTimerGetToggleBit (void)

Reads which bit in Timer counter 2 is monitored for a toggle.

Returns:

The bit that is monitored (range of 0 to 31)

uint32 CySysTimerGetMatch (uint32 *counterNum*)

Reads the Timer counter match comparison value.

Parameters:

counterNum	Valid range [0-1]. The number of the DeepSleep Timer counter. The match values are not supported by counter 2.

Returns:

A 16-bit match value.

uint32 CySysTimerGetCount (uint32 *counterNum*)

Reads the current DeepSleep Timer counter value.

Parameters:

counterNum	Valid range [0-2]. The number of the Timer counter.

Returns:

A live counter value. Counter 0 and Counter 1 are 16 bit counters and counter 2 is a 32 bit counter.

uint32 CySysTimerGetInterruptSource (void)

Reads a mask containing all the DeepSleep Timer counters interrupts that are currently set by the hardware, if a corresponding mode is selected.

Returns:

The mask of interrupts set
CY_SYS_TIMER0_INT - Set interrupt for Counter 0
CY_SYS_TIMER1_INT - Set interrupt for Counter 1
CY_SYS_TIMER2_INT - Set interrupt for Counter 2

void CySysTimerClearInterrupt (uint32 *counterMask*)

Clears all the DeepSleep Timer counter interrupts set in the mask.

All the Timer interrupts are to be cleared by the firmware, otherwise interrupts are generated continuously.

Parameters:

counterMask	CY_SYS_TIMER0_INT - Clear counter 0 CY_SYS_TIMER1_INT - Clear counter 1 CY_SYS_TIMER2_INT - Clear counter 2

cyTimerCallback CySysTimerSetInterruptCallback (uint32 *counterNum*, cyTimerCallback *function*)

Sets the ISR callback function for the particular DeepSleep Timer counter.

These functions are called on the Timer interrupt.

Parameters:

counterNum	The number of the Timer counter.
function	The pointer to the callback function.

Returns:

The pointer to the previous callback function.
NULL is returned if the specified address is not set.

cyTimerCallback CySysTimerGetInterruptCallback (uint32 *counterNum*)

Gets the ISR callback function for the particular DeepSleep Timer counter.

Parameters:

counterNum	The number of the Timer counter.

Returns:

The pointer to the callback function registered for a particular Timer by a particular address that are passed through arguments.

void CySysTimerResetCounters (uint32 *countersMask*)

Resets all the Timer counters set in the mask.

Parameters:

countersMask	CY_SYS_TIMER0_RESET - Reset the Counter 0 CY_SYS_TIMER1_RESET - Reset the Counter 1 CY_SYS_TIMER2_RESET - Reset the Counter 2

This function waits while corresponding counters will be reset. This may take up to 3 DeepSleep Timer source-cycles. DeepSleep Timer source must be enabled. Otherwise, the function will never exit.

void CySysTimerEnableIsr (uint32 *counterNum*)

Enables the ISR callback servicing for the particular Timer counter.

Parameters:

counterNum	Valid range [0-2]. The number of the Timer counter.

Value corresponds to appropriate Timer counter. For example value 1 corresponds to second Timer counter.

void CySysTimerDisableIsr (uint32 *counterNum*)

Disables the ISR callback servicing for the particular Timer counter.

Parameters:

counterNum	Valid range [0-2]. The number of the Timer counter.

void CySysTimerIsr (void)

This is the handler of the DeepSleep Timer interrupt in CPU NVIC.

The handler checks which Timer triggered in the interrupt and calls the respective callback functions configured by the user by using CySysTimerSetIsrCallback() API.

The order of the callback execution is incremental. Callback-0 is run as the first one and callback-2 is called as the last one.

This function clears the DeepSleep Timer interrupt every time when it is called.

void CySysTimerDelay (uint32 *counterNum*, cy_sys_timer_delaytype_enum *delayType*, uint32 *delay*)

The function implements the delay specified in the LFCLK clock ticks.

This API is applicable for PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices to use WDT. Also this API is available to use for PSoC4100S and / PSoC Analog Coprocessor devices to use DeepSleep Timers.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices: The specified WDT counter should be configured as described below and started.

For PSoC 4100S / PSoC Analog Coprocessor devices: The specified DeepSleep Timer counter should be configured as described below and started.

This function can operate in two modes: the "WAIT" and "INTERRUPT" modes. In the "WAIT" mode, the function waits for the specified number of ticks. In the "INTERRUPT" mode, the interrupt is generated after the specified number of ticks.

For the correct function operation, the "Clear On Match" option should be disabled for the specified WDT or DeepSleep Timer counter. Use CySysWdtSetClearOnMatch() for WDT or CySysTimerSetClearOnMatch() for DeepSleep Timer function with the "enable" parameter equal to zero for the used WDT counter or DeepSleep Timer counter.

The corresponding WDT counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" / "Watchdog (w/Interrupt)" for the "INTERRUPT" mode.

Or the corresponding DeepSleep Timer counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" for the "INTERRUPT" mode.

This can be configured in two ways:

- Through the DWR page. Open the "Clocks" tab, click the "Edit Clocks..." button, in the "Configure System Clocks" window click on the "Low Frequency Clocks" tab and choose the appropriate option for the used WDT or DeepSleep Timer counter.
- Through the CySysWdtSetMode() for WDT or CySysTimerSetMode() for DeepSleep Timer function. Call it with the appropriate "mode" parameter for the used WDT or DeepSleep Timer counter.

For the "INTERRUPT" mode, the recommended sequence is the following:

- Call the CySysWdtDisableCounterIsr() for WDT or CySysTimerDisableIsr() for DeepSleep Timer function to disable servicing interrupts of the specified WDT or DeepSleep Timer counter.

- Call the CySysWdtSetInterruptCallback() for WDT or CySysTimerSetIsrCallback() for DeepSleep Timer function to register the callback function for the corresponding WDT or DeepSleep Timer counter.
- Call the CySysTimerDelay() function.

Parameters:

counterNum	Valid range [0-1]. The number of the counter (Timer0 or Timer1).
delayType	CY_SYS_TIMER_WAIT - "WAIT" mode. CY_SYS_TIMER_INTERRUPT - "INTERRUPT" mode.
delay	The delay value in the LFCLK ticks (allowable range - 16-bit value).

In the "INTERRUPT" mode, this function enables ISR callback servicing from the corresponding WDT or DeepSleep Timer counter. Servicing of this ISR callback will be disabled after the expiration of the delay time.

void CySysTimerDelayUntilMatch (uint32 *counterNum*, cy_sys_timer_delaytype_enum *delayType*, uint32 *match*)

The function implements the delay specified as the number of WDT or DeepSleep Timer clock source ticks between WDT or DeepSleep Timer current value and match" value.

This API is applicable for PSoC 4100 / PSoC 4200 / PRoC BLE / PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices to use WDT. Also this API is available to use for PSoC4100S / Analog Coprocessor devices to use DeepSleep Timers.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices: The function implements the delay specified as the number of LFCLK ticks between the specified WDT counter's current value and the "match" passed as the parameter to this function. The current WDT counter value can be obtained using the CySysWdtGetCount() function.

For PSoC4100 S and Analog Coprocessor devices: The function implements the delay specified as the number of DeepSleep Timer input clock ticks for Timer0/Timer1 counter's current value and the "match" passed as the parameter to this function. The current DeepSleep Timer counter value can be obtained using the CySysWdtGetCount() function.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M devices: The specified WDT counter should be configured as described below and started.

For PSoC PSoC 4100S / PSoC Analog Coprocessor devices: The specified DeepSleep Timer counter should be configured as described below and started.

This function can operate in two modes: the "WAIT" and "INTERRUPT" modes. In the "WAIT" mode, the function waits for the specified number of ticks. In the "INTERRUPT" mode, the interrupt is generated after the specified number of ticks.

For the correct function operation, the "Clear On Match" option should be disabled for the specified WDT or DeepSleep Timer counter. Use CySysWdtSetClearOnMatch() for WDT or CySysTimerSetClearOnMatch() for DeepSleep Timer function with the "enable" parameter equal to zero for the used WDT or DeepSleep Timer counter.

For PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M/PSoC 4200M devices: The corresponding WDT counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" / "Watchdog (w/Interrupt)" for the "INTERRUPT" mode.

For PSoC 4100S / PSoC Analog Coprocessor devices: Corresponding DeepSleep Timer counter should be configured to match the selected mode: "Free running Timer" for the "WAIT" mode, and "Periodic Timer" for the "INTERRUPT" mode.

This can be configured in two ways:

- Through the DWR page. Open the "Clocks" tab, click the "Edit Clocks..." button, in the "Configure System Clocks" window click on the "Low Frequency Clocks" tab and choose the appropriate option for the used WDT or DeepSleep Timer counter.
- Through the CySysWdtSetMode() for WDT or CySysTimerSetMode() for DeepSleep Timer function. Call it with the appropriate "mode" parameter for the used WDT or DeepSleep Timer counter.

For the "INTERRUPT" mode, the recommended sequence is the following:

- Call the CySysWdtDisableCounterIsr() for WDT or CySysTimerDisableIsr() for DeepSleep Timer function to disable servicing interrupts of the specified WDT or DeepSleep Timer counter.
- Call the CySysWdtSetInterruptCallback() for WDT or CySysTimerSetInterruptCallback() for DeepSleep Timer function to register the callback function for the corresponding WDT or DeepSleep Timer counter.
- Call the CySysTimerDelay() function.

Parameters:

counterNum	Valid range [0-1]. The number of the WDT or DeepSleep Timer. counter (Timer0 or Timer1).
delayType	CY_SYS_TIMER_WAIT - "WAIT" mode. CY_SYS_TIMER_INTERRUPT - "INTERRUPT" mode.
delay	The delay value in the LFCLK ticks (allowable range - 16-bit value).

In the "INTERRUPT" mode, this function enables ISR callback servicing from the corresponding WDT counter. Servicing of this ISR callback will be disabled after the expiration of the delay time.

MISRA 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.

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
12.4	R	Right hand operand of '&&' or '||' is an expression with possible side effects.	The reason of this violation that the one of operands is the value of register.

API Memory Usage

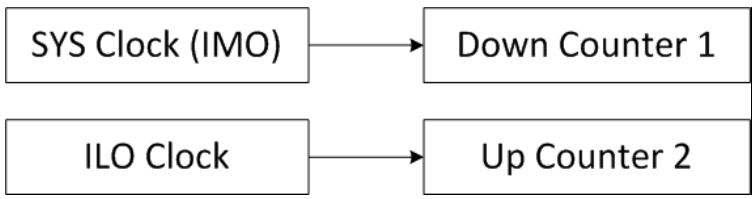
Refer to the PSoC 4 Family *System Reference Guide* (cy_boot).

ILO compensating / Trimming

Compensating / trimming processes use two-system counters controlled by two registers. Counter 1 is a down-counter clocked by the SysClk which is sourced by the accurate IMO. Counter 2 is an up-counter clocked by the ILO. The CySysClkIloStartMeasurement API configures these counters to be clocked respectively by the SysClk (Counter 1) and ILO (Counter 2).

Note SysClk should be sourced by IMO. If SysClk is sourced by another source (ECO or external source) the ILO compensating / trimming APIs can return unexpected results.

When a known 16-bit value (SysClk counts) is loaded into Counter 1, Counter 2 starts a clock from the ILO source. Counter 1 starts to count down on the SysClk, while Counter 2 starts to count up on the ILO source. Both counters stop when Counter 1 reaches 0, and the corresponding count value can be read out of Counter 2.



Using the ILO counts counted in Counter 2, the ILO current frequency can be calculated:

$$ILO_{CurrentFreq} = (ILO_Counts * SysClk_{Freq}) / SysClk_Counts$$

Basing on the obtained ILO frequency, the required number of the ILO cycles for a given delay can be calculated:

$$ILO_{accurate\ clocks} = (ILO_{CurrentFreq} * desiredDelay_clocks) / ILO_{NomFreq}$$

Also, the relative ILO accuracy can be calculated:

$$\Delta_{ILO\ accuracy} = (ILO_{CurrentFreq} - ILO_{NomFreq}) / ILO_{NomFreq}$$

Note The compensating API is applicable for all PSoC4 devices. The trimming API is applicable only for PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M. For more details, see the API descriptions in the Application Programming Interface.

Warning The compensating/trimming functionality is based on the measured ILO frequency. The ILO frequency can be measured only in active-power mode. When switching the device into low-power modes (Sleep, DeepSleep), the ILO frequency might change and there is no possibility to measure the ILO frequency in low-power modes. Therefore, compensating/trimming functionality cannot guarantee +/-10% ILO accuracy in low-power modes.

Selecting WCO Output Source

For devices with WCO (PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M / PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor) a possibility is created to externally drive the WCO. If you want to use an external pin to drive the WCO, the following procedure is required:

1. Disable the WCO.
2. Drive the wco_out pin to an external signal source.
3. Call CySysClkWcoClockOutSelect().
4. Enable the WCO and wait for 15 us before clocking the wco_out pin pad at a high potential [1].

Note Do not use the oscillator output clock prior to a 15-microsecond delay in your system.

If you want to use the WCO after using an external pin source:

---

[1] Assuming you are using the1.6 V clock amplitude, then the sequence would start at 1.6 V, then 0 V, then 1.6 V, etc. at a chosen frequency.

1. Disable the WCO.
2. Drive the wco_out pin with external signal.
3. Call CySysClkWcoClockOutSelect().
4. Enable the WCO.

For stable WCO operation with an external clock source, the VDDA/VDDD external supply should be in the range from 1.65 V to 5 V. For stable WCO operation, the amplitude of the clock source driving the wco_out pin should be from a minimum of 1.0 V to a maximum of 1.6 V with respect to Vgnd because low-voltage devices depend on this external clock source. Therefore, the requirement is to provide an external clock source that toggles from 0 V to a minimum of 1.0 V or 0 V to the maximum of 1.6 V.

The specification for the WCO clock duty-cycle is 20% to 80%, so the external clock source should adhere to this same specification.

There are no limitations on the external clock frequency.

As mentioned in the procedure above, the wco_out pin should be used to drive an external clock source. Even if driving the wco_in pin would work, you get better performance using the wco_out pin because we can bypass the feedback resistor because this resistor causes an RC delay and causes duty-cycle variation.

In case when the WCO block is sourced by an external clock source it is possible to trim the IMO from the WCO only when the external clock source period is equal to the WCO period. Also, the external clock source accuracy should be higher or equal to the WCO accuracy.

See the CySysClkWcoClockOutSelect() explanation in the API section.

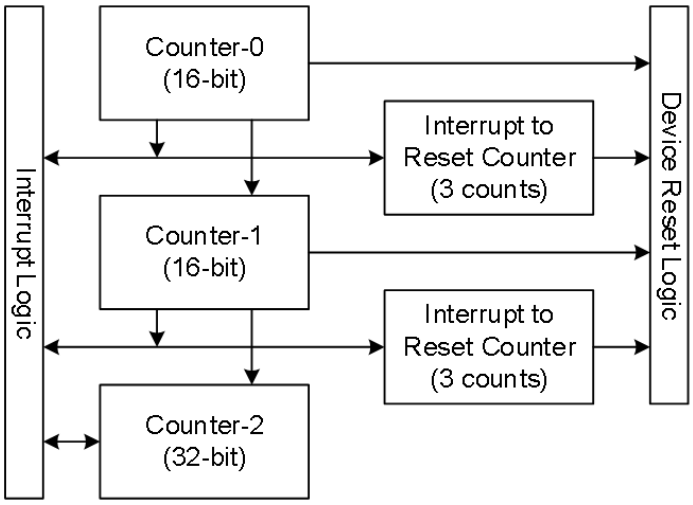
WDT Functional Description

PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200L / PSoC 4100M / PSoC 4200M

The WDT asserts an interrupt or a hardware reset to the device after a preprogrammed interval, unless it is periodically serviced in firmware. The WDT has two 16-bit counters (Counter-0 and Counter-1) and one 32-bit counter (Counter-2).

These counters can be configured to work independently or in cascade. The cascade configuration provides an option to increase the reset or interrupt interval.

If Counter-0 and Counter-1 are set in a cascade, it should be noted that Counter-0 is performing action on (match value + 1), but Counter-1 is performing action on (match value).



Counter-0 and Counter-1 generate an interrupt or a reset on reaching the specified terminal count for the first time. After three continuous unhandled interruptsCounter-0 and Counter-1 generate a reset, whereas Counter-2 only generates an interrupt. The interrupt must be cleared after entering the Interrupt Service Routine (ISR) in firmware by calling CySysWdtClearInterrupt() with the corresponding parameter.

Counter-0 and Counter-1 perform actions when the corresponding counter value equals the corresponding match value configured by calling CySysWdtWriteMode(). Counter-2 performs the action when the bit defined by calling CySysWdtWriteToggleBit() is toggled in Counter-2. For example, if the toggle bit is bit number 7 (configured by call to the CySysWdtWriteToggleBit(7) function), Counter-2 generates one interrupt per 2^7=128 WDT clocks.

Power Modes

In Active mode, an interrupt request from the WDT is sent to the CPU via IRQ 9. In Sleep or Deep Sleep power mode, the CPU subsystem is powered-down, so the interrupt request from the WDT is directly sent to the WakeUp Interrupt Controller (WIC), which will then wake up the

CPU. Then, the CPU acknowledges the interrupt request and executes the Interrupt Service Routine (ISR).

After waking from Deep Sleep, an internal timer value is set to zero until the ILO loads the register with the correct value. This led to an increase in Low-power mode current consumption. The work around is to wait for the first positive edge of the ILO clock before allowing the WDT_CTR_* registers to be read by CySysWdtReadCount() function.

Clock Source

The WDT is clocked by the LFCLK. The LFCLK can be sourced by a 32 kHz ILO or WCO. The WCO is available only for the PSoC 4100 BLE / PSoC4200 BLE, PSoC 4200L, and PSoC 4100M / PSoC 4200M devices. According to the device datasheet, the ILO accuracy is +/-60% over voltage and temperature. This means that the timeout period may vary by 60% from the configured value. Appropriate margins should be added while configuring WDT intervals to make sure that unwanted device resets do not occur on some devices.

According to the datasheet, the ILO accuracy can be obtained up to +/-10% by using a trimming API. Also, use a compensating API to obtain more accurate WDT functioning.

Refer to the device datasheet for more information on the oscillator accuracy.

Register Locking

Accidental corruption of the WDT configuration can be prevented by setting the bit-field WDT_LOCK of the CLK_SELECT register by calling the CySysWdtLock() function. When WDT is locked, any writing to the WDT_* and CLK_ILO* registers is ignored.

The CySysWdtUnlock() function should be called to allow WDT registers modification.

Clearing WDT

The LFCLK clock is asynchronous to the SYSCLK. So, generally, it takes 3 LFCLK cycles for the WDT register changes to come into effect. It is important to remember that a WDT should be cleared at least 4 cycles (3 + 1 for sure) before a timeout occurs, especially when small match values / low-toggle bit numbers are used.

The WDT counters should be cleared by calling the CySysWdtResetCounters() function with the parameter corresponding to the counters whose values are going to be cleared.

It is recommended to clear WDT counters from the portion of the code that is not directly associated with the WDT interrupt. It is possible that the main function of the firmware has crashed or is in an endless loop, but that the WDT interrupt vector is still intact and the WDT is getting serviced properly.

Reset Detection

The CySysGetResetReason() function can be used to detect if the watchdog has triggered a device reset.

Interrupt Configuration

The Global Signal Reference and Interrupt components can be used for the ISR configuration. If the WDT is configured to generate an interrupt, the pending interrupts must be cleared within ISR (otherwise, the interrupt will be generated continuously):

A pending interrupt to the interrupt controller must be cleared by the call to the WDTISR_ClearPending() function, where WDTISR is the instance name of the interrupt component.

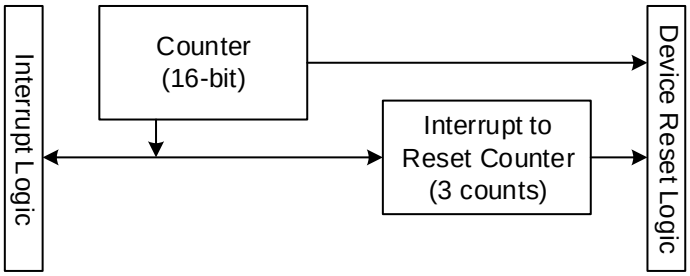
A pending interrupt to the WDT block must be cleared by the call to the CySysWdtClearInterrupt() function. The call to the function clears the unhandled WDT interrupt counter, if WDT is configured to be in "Generate interrupts and reset on 3rd unhandled interrupt" mode.

It is recommended to use the WDT ISR as a timer to trigger certain actions and to change the next WDT match value.

PSoC 4000 / PSoC 4000S / 4100S and PSoC Analog Coprocessor

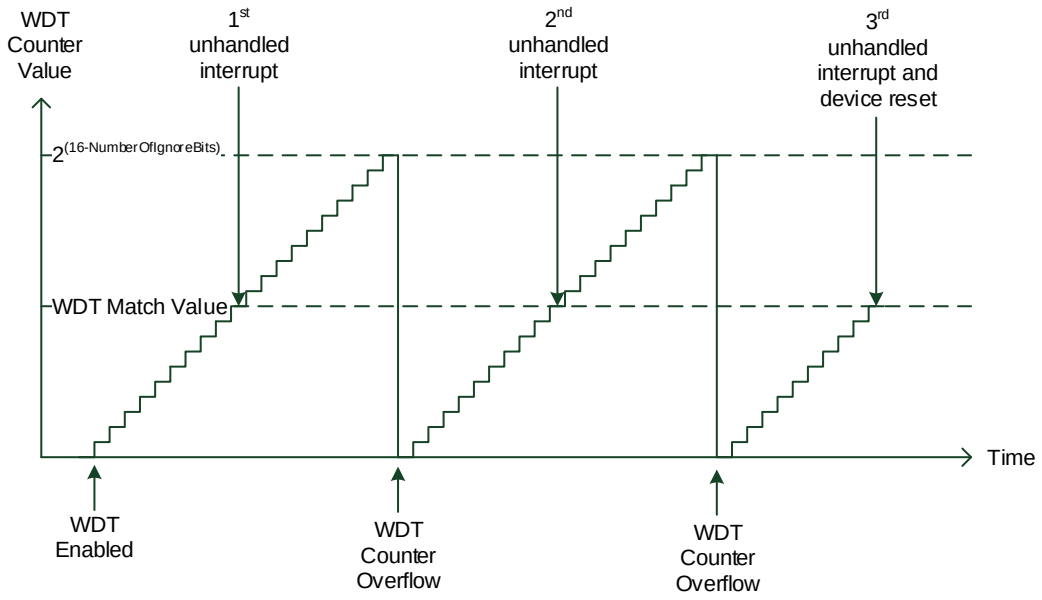
Note It is highly recommended to enable a WDT if the power supply can produce sudden brown-out events that may compromise the CPU functionality. This ensures that after a brown-out compromises the CPU functionality, the system always recovers.

The WDT asserts an interrupt or a hardware reset to the device after a preprogrammed interval, unless it is periodically serviced in firmware. The WDT is a 16- bit free-running up-counter.



The WDT generates an interrupt when the count value in the counter equals the configured match value.

It is important that the counter is not reset on a match. When the counter reaches a match value, it generates an interrupt and then keeps counting up till it overflows and rolls back to zero and reaches the match value again at which point another interrupt is generated.



To use a WDT for a periodic interrupt generation, the match value should be incremented in the ISR. As a result, the next WDT interrupt is generated when the counter reaches a new match value.

Also, some functionality is added to reduce the entire WDT counter period by specifying the number of most significant bits that are cut-off in the WDT counter. For example, if the CySysWdtWriteIgnoreBits() function is called with parameter 3, the WDT counter becomes a 13-bit free-running up-counter.

The WDT reset period can be calculated using the following equation:

$$WDT_{ResetTime} = 2 * (LFCLK_{Period} * (2^{(16 - NumberOfIgnoreBits)})) + (LFCLK_{Period} * WDT_{MatchValue})$$

Power Modes

In Active mode, the interrupt request from the WDT is sent to the CPU via IRQ 4. In the Sleep or Deep Sleep power mode, the CPU subsystem is powered down, so the interrupt request from the WDT is directly sent to the WakeUp Interrupt Controller (WIC). The WIC wakes up the CPU. Then, the CPU acknowledges the interrupt request and executes the Interrupt Service Routine (ISR).

Enabling or disabling a WDT requires three LFCLK cycles to come into effect. During that period the SYSCLK clock should be available. That means that the device should not be put into Deep Sleep mode during that period.

After waking from Deep Sleep, an internal timer value is set to zero until the ILO loads the register with the correct value. This leads to an increase in the Low-power mode current consumption. The workaround is to wait for the first positive edge of the ILO clock before allowing the WDT_CTR_* registers to be read by the CySysWdtReadCount() function.

Clock Source

The WDT is clocked by the LFCLK sourced by the 32 kHz ILO. The WDT reset must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO is ignored. Enabling the WDT reset automatically enables the ILO.

According to the device datasheet, the ILO accuracy is +/-60% over voltage and temperature. This means that the timeout period may vary by 60% from the configured value. An appropriate margin should be added while configuring WDT intervals to make sure that unwanted device resets do not occur on some devices.

Use a compensating API to obtain more accurate WDT functioning. Refer to the device datasheet for more information on the oscillator accuracy.

Register Locking

This feature is not available for the device.

Clearing WDT

The LFCLK clock is asynchronous to the SYSCLK. So, generally, it takes three LFCLK cycles for the WDT registers changes to come into effect.

Note A WDT should be cleared at least for four cycles (3 LFCLK cycles + 1 to be sure) before a timeout occurs, especially when small match values / low toggle bit number are used.

It is recommended to clear WDT counters from the portion of the code that is not directly associated with a WDT interrupt. It is possible that the main function of the firmware has crashed or is in an endless loop, but that the WDT interrupt vector is still intact and the WDT is getting serviced properly.

Reset Detection

The CySysGetResetReason() function can be used to detect if the watchdog has triggered a device reset.

Interrupt Configuration

The Global Signal Reference and Interrupt components can be used for the ISR configuration. If the WDT is configured to generate an interrupt, the pending interrupts must be cleared within ISR (otherwise, the interrupt will be generated continuously):

A pending interrupt to the interrupt controller must be cleared by the call to the WDTISR_ClearPending() function where WDTISR is the instance name of the interrupt component.

A pending interrupt to the WDT block must be cleared by the call to the CySysWdtClearInterrupt() function. The call to the function will clear the unhandled WDT interrupt counter, if the WDT is configured to be in “Generate interrupts and reset on 3rd unhandled interrupt” mode.

It is recommended to use the WDT ISR as a timer to trigger certain actions and to change a next WDT match value.

Note If the Watchdog is configured as “Watchdog (w/ interrupts),” the interrupts from WDT are not passed to the CPU to avoid unregistered interrupts. On the third unhandled interrupt, a continuous device reset occurs. To avoid a continuous device reset, call the CySysWdtUnmaskInterrupt() API. After that, call the APIs with WDT interrupts handling/clearing.

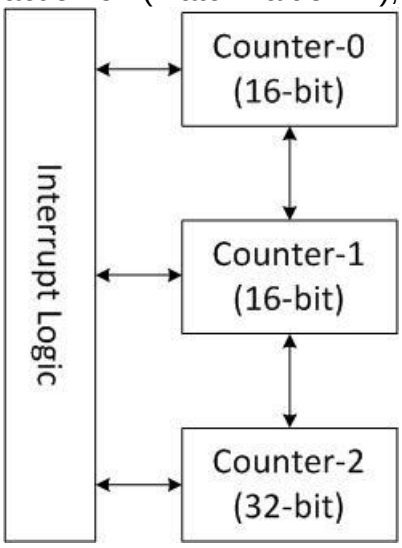
Deep Sleep Timers Functional Description

PSoC 4100S and PSoC Analog Coprocessor

The Deep Sleep Timer asserts an interrupt to the device after a preprogrammed interval, unless it is periodically serviced in firmware. The Timer has two 16-bit counters (Counter-0 and Counter-1) and one 32-bit counter (Counter-2).

These counters can be configured to work independently or in cascade. The cascade configuration provides an option to increase the reset or interrupt interval.

If Counter-0 and Counter-1 are set in a cascade, it should be noted that Counter-0 is performing action on (match value + 1), but Counter-1 is performing action on (match value).



Counter-0 and Counter-1 generate an interrupt on reaching the specified terminal count for the first time. All three counters can only generates an interrupt. The interrupt must be cleared after entering the Interrupt Service Routine (ISR) in firmware by calling CySysTimerClearInterrupt() with the corresponding parameter.

Counter-0 and Counter-1 perform actions when the corresponding counter value equals the corresponding match value configured by calling CySysTimerWriteMode(). Counter-2 performs the action when the bit defined by calling CySysTimerWriteToggleBit() is toggled in Counter-2. For example, if the toggle bit is bit number 7 (configured by call to the

CySysTimerWriteToggleBit(7) function), Counter-2 generates one interrupt per 2^7=128 Timer clocks.

Power Modes

In Active mode, an interrupt request from the Deep Sleep Timer is sent to the CPU. In Sleep or Deep Sleep power mode, the CPU subsystem is powered-down, so the interrupt request from the Deep Sleep Timer is directly sent to the WakeUp Interrupt Controller (WIC), which will then wake up the CPU. Then, the CPU acknowledges the interrupt request and executes the Interrupt Service Routine (ISR).

After waking from Deep Sleep, an internal timer value is set to zero until the ILO loads the register with the correct value. This led to increase in Low-power mode current consumption. The work around is to wait for the first positive edge of the ILO clock before allowing the WCO_WDT_CTR_* registers to be read by CySysTimerReadCount() function.

Clock Source

The Deep Sleep Timers are clocked by a 32 kHz ILO or WCO. According to the device datasheet, the ILO accuracy is +/-60% over voltage and temperature. This means that the timeout period may vary by 60% from the configured value if the Deep Sleep Timers are clocked by ILO. Appropriate margins should be added while configuring Timers intervals to make sure that unwanted interrupt was generated on some devices.

According to the datasheet, the ILO accuracy can be obtained up to +/-10% by using a trimming API. Also, use a compensating API to obtain more accurate Deep Sleep Timers functioning.

Refer to the device datasheet for more information on the oscillator accuracy.

Also to obtain accurate Deep Sleep Timers functioning use the WCO to drive Timers.

Clearing Deep Sleep Timers

The Deep Sleep Timer source (ILO or WCO) clock is asynchronous to the SYSCLK. So, generally, it takes 3 Timer source-cycles for the Deep Sleep Timers register changes to come into effect. It is important to remember that a Deep Sleep Timers should be cleared at least 4 cycles (3 + 1 for sure) before a timeout occurs, especially when small match values / low-toggle bit numbers are used.

The Deep Sleep Timer counters should be cleared by calling the CySysTimerResetCounters() function with the parameter corresponding to the counters whose values are going to be cleared.

It is recommended to clear Deep Sleep Timer counters from the portion of the code that is not directly associated with the Timer interrupt. It is possible that the main function of the firmware has crashed or is in an endless loop, but that the Timer interrupt vector is still intact and the Deep Sleep Timer is getting serviced properly.

Interrupt Configuration

The Global Signal Reference and Interrupt components can be used for the ISR configuration. If the Deep Sleep Timer is configured to generate an interrupt, the pending interrupts must be cleared within ISR (otherwise, the interrupt will be generated continuously):

A pending interrupt to the interrupt controller must be cleared by the call to the TimerISR_ClearPending() function, where TimerISR is the instance name of the interrupt component.

A pending interrupt to the Deep Sleep Timer block must be cleared by the call to the CySysTimerClearInterrupt() function.

It is recommended to use the Deep Sleep Timer ISR as a timer to trigger certain actions and to change the next Timer match value.

AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

Note Final characterization data for PSoC 4000S, PSoC 4100S and PSoC Analog Coprocessor devices is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.

Symbol	Description	Conditions	Min	Typ	Max	Units
$F_{ILOTRIM1}$	ILO frequency for PSoC 4100 / PSoC 4200 / PSoC 4100 BLE / PSoC 4200 BLE / PRoC BLE / PSoC 4200M / PSoC 4200L	Trimmed ILO frequency (+/ 60%)	15	32	50	kHz
	ILO frequency for PSoC 4000 / PSoC 4000S / 4100S / PSoC Analog Coprocessor		20	40	80	kHz
$F_{ILOTRIM2}$	ILO Frequency after trimming	ILO accuracy after successful completion of ILO trim	28.8	32	35.2	kHz
$T_{ILOTRIMDUR}$	Time taken to successfully complete ILO trim	IMO running at 24 MHz and using the APIs provided as part of cy_lfclk	0.5	-	30	ms
$T_{STARTILO}$	ILO Startup time		-	-	2	ms
F_{WCOTOL}	WCO Frequency tolerance	With 20 ppm crystal	-	50	250	ppm
C_{WCOL}	WCO Crystal load capacitance		6	-	12.5	pF
C_{WCO0}	WCO Crystal shunt capacitance		-	1.35	-	pF
R_{WCOESR}	WCO equivalent series resistance		-	50	-	kΩ

Symbol	Description	Conditions	Min	Typ	Max	Units
$T_{STARTWCO}$	WCO startup/settling time	WCO settling delay during System boot	-	-	500	ms
F_{WCO}	WCO frequency		-	32.768	-	kHz

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.10.b	Edited datasheet.	Provided additional information about difference between WDTs and Deep Sleep Timers in General Description. Added information about compensating/trimming ILO frequency API.
1.10.a	Updated LFCLK Configuration Panels Added additional information about Deep Sleep Timers Changed term “WCO Timers” on “Deep Sleep Timers” instead	Provide more clear information
	Edited datasheet.	Final characterization data for PSoC 4000S, PSoC 4100S and PSoC Analog Coprocessor devices is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.
1.10	Added APIs to compensate and trim ILO frequency. Added API for externally driving the WCO. Added support for PSoC 4000S / PSoC 4100S / Analog Coprocessor Added WCO Timers APIs.	Increase ILO accuracy. To drive the WCO by an external source. Support new PSoC 4 device families. Support for WCO Timers.
1.0.b	Edited the datasheet.	Removed the Errata section. Fixed the PSoC 4000 Configure Dialog. The “Timer (WDT) ISR” panel and WDT interrupt generation is disabled when the Timer (WDT) panel is disabled.
		Added a note to Enable the WDT if the power supply might cause a brown-out event.
1.0.a	Added Component Errata section.	Document an issue and workaround with the PSOC 4000 WDT.
1.0	Initial component version.	

© Cypress Semiconductor Corporation, 2016. 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, 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.