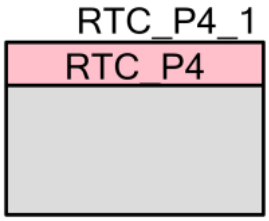
PSoC 4 Real-Time Clock (RTC_P4)

1.20



Features

- Multiple alarm options
- Configurable alarm functionality
- Daylight Savings Time (DST) functionality
- Automatic leap year compensation
- Unix/Epoch time

General Description

The PSoC 4 Real-time Clock (RTC_P4) Component provides an application interface for keeping track of time and date. The Component can have any of the Watchdog Timers (WDT) or DeepSleep Timers available in the device as its input clock source, or have a user-configured clock source such as SysTick, TCPWM, etc.

- If you choose to use a WDT or DeepSleep Timer as the input clock source in the "Low Frequency Clocks" Configure System Clock dialog, the Component derives the input clock period from the WDT or DeepSleep Timer frequency configured.
  - The WDTs are available for all PSoC 4 devices.
  - The DeepSleep Timers are available for PSoC 4100S and PSoC Analog Coprocessor devices.

  Note Not all WDTs can be driven by accurate WCO, even though a WCO is available. For more information about WDT(s), DeepSleep Timers, and WCO, refer to the to the cy_lfclk Component datasheet.

- If you choose to use other sources, then you need to set the period of the input clock source manually. For more details, refer to the Ticks counter updating behavior section.

  Note The accuracy of the RTC depends on the accuracy of the input clock source.

The time can be represented in either 12-hour format or 24-hour format. The date representation can be in "MM/DD/YYYY", "DD/MM/YYYY" or "YYYY/MM/DD" format. The RTC_P4 Component keeps track of second, minute, hour, day of the week, day of the month, month, and year. The day of the week is automatically calculated from the day, month, and year.

It automatically accounts for leap year changes. Leap year is identified as the year, which is a multiple of 4 or 400, but not 100. Note that the time is in GMT +00:00 hour zone as the time is derived from UNIX time, which is UTC.

DST may optionally be enabled and supports any start and end date. The start and end dates can be fixed date like 24 March or relative like the second or last Sunday in March.

The Component has an optional alarm feature, which provides match detection for a second, minute, hour, day of week, day of month, month, and year. A mask selects what combination of time and date information will be used to generate the alarm. The alarm flexibility supports periodic alarms such as every twenty-third minute after the hour, or a single alarm such as 4:52 a.m. on September 28, 2043.

The Component offers time and date as a single integer value, representing time and date in Unix/Epoch format. This is a single integer value storing number of seconds elapsed since 12:00:00 AM January 1, 1970, UTC.

When to Use an RTC_P4 Component

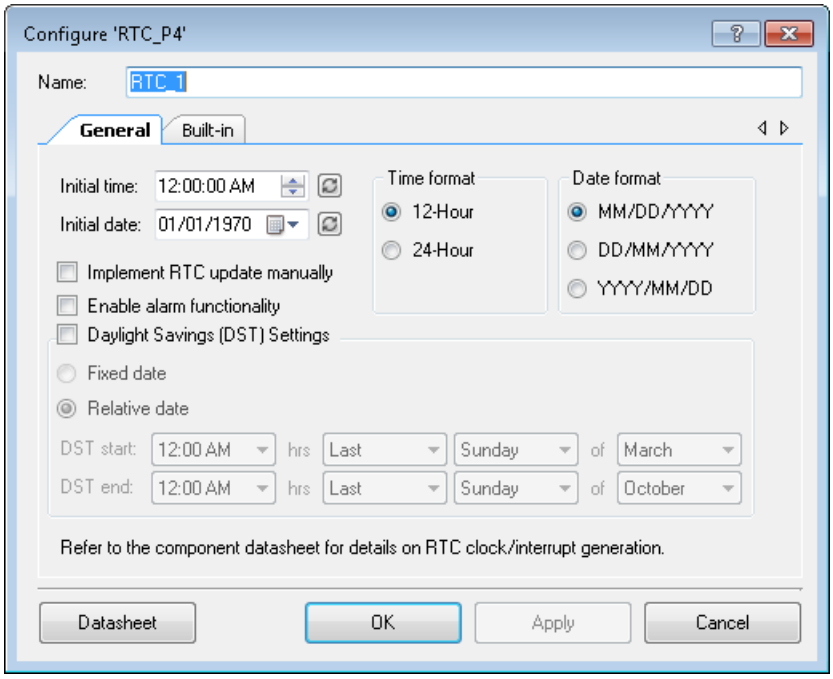
Use the RTC_P4 Component when the system requires the current time or date. You can also use the RTC_P4 when you do not need the current time and date but you need accurate timing of events with one-second resolution.

Input/Output Connections

The RTC_P4 Component does not have input or output connections.

Component Parameters

Drag an RTC_P4 Component onto your design and double-click it to open the **Configure** dialog.

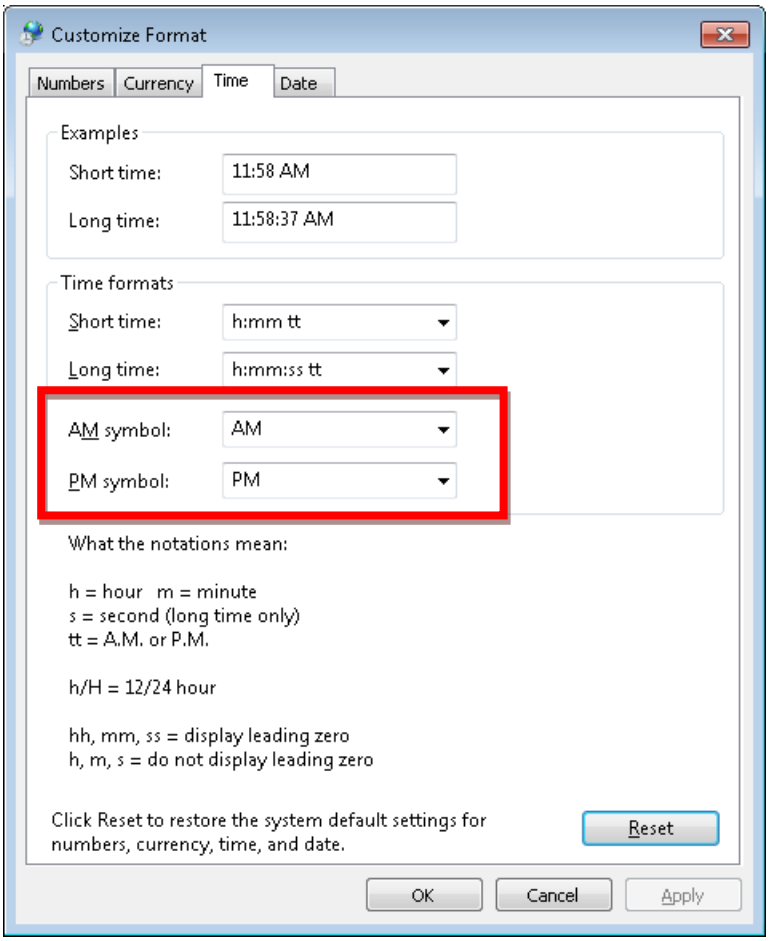


The RTC_P4 Component contains the following options:

Initial time

This parameter allows users to choose whether the daylight savings time functionality is enabled in the RTC_P4 Component. The default value is cleared (false). Sets the initial time for the RTC_P4 Component; Default time is "00:00:00" or "12:00:00 AM". The value can be set by clicking the hour, min or sec entry and using the up/down buttons or the value can be entered directly. The format in which time is displayed depends on the '**Time Format**' selected in the dialog.

Note For 12-hour format, the parameters "AM symbol" and "PM symbol" should be properly configured in the "Customize Format" dialog in Windows. See the following figure (open via the **Control Panel > Clock, Language, and Region > Change the date, time, or number format > Additional settings > Time** tab):



Initial date

Sets the initial date for the RTC_P4 Component; the default date is "01-01-1970."

The value can be set using the date selection dropdown. You can click on day, month, and year value and enter them manually.

Time format

This parameter selects how the time is represented/stored in the time variable. You can select the standard 12-hour format or 24-hour format.

Date format

This parameter selects how the date is represented/stored in the date variable. You can select from one of three standard formats:

- "MM/DD/YYYY" (Default)
- "DD/MM/YYYY"
- "YYYY/MM/DD"

Implement RTC update manually

This parameter is used to map the RTC time update API automatically during RTC start to one of the WDTs or DeepSleep Timers selected and configured for RTC in the LFCLK interface. If this parameter is checked, then the mapping of RTC update API needs to be resolved manually. This parameter has no effect (mapping is manual by default) if ‘None’ option is selected in ‘RTC_sel Mux’ in Clocks configuration window (Low frequency clocks tab) or ‘User provided’ option is selected in ‘Timer (WDT) ISR’ panel.

Note This parameter applies to the following device families:

- PSoC 4100 BLE/PSoC 4200 BLE
- PRoC BLE
- PSoC 4100M/PSoC 4200M
- PSoC 4200L
- PSoC 4000S/PSoC 4100S
- PSoC Analog Coprocessor

The ‘RTC_sel Mux’ and ‘Timer (WDT) ISR’ panel can be configured in the Low frequency clocks tab. To access the Low frequency clocks tab, open the Design-Wide Resources Clock Editor from the Workspace Explorer. Then, double-click any LFCLK clock source to open the Configure System Clock dialog. For more information, refer to the PSoC Creator Help and the cy_lfclk Component datasheet.

Enable Alarm Functionality

This parameter allows you to enable/disable the alarm functionality available in the RTC_P4 Component. Disabling the feature will remove the code related to alarm generation in the Component.

Daylight Savings (DST) Settings

This parameter allows you to choose whether the daylight savings time functionality is enabled in the RTC_P4 Component.

DST Settings

These settings are enabled only if the check box is checked. These settings provide two sets of parameters depending on the type of DST date. If the DST date is a ‘Relative date’, then you can set day of week, week of month, and month. If the DST date is a ‘Fixed date’, then you can set day of month and month. The Start/Stop hours setting is available in both the date modes.

- *DST settings with ‘Relative date’ option*

  This parameter selects "Relative date" format for storing the DST start/stop dates. A relative date can be "Last Sunday of March".

- *DST settings with ‘Fixed date’ option*

  This parameter selects "Fixed date" format for storing the DST start/stop dates. A fixed date can be "21 March".

Clock Selection

The Component provides an option to map the RTC time update API to any clock source that you want. The time update API is allowed to attach to any interrupts like WDT, DeepSleep Timer, SysTick, TCPWM, etc.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail. By default, PSoC Creator assigns the instance name "RTC_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "RTC." You should disable the component's interrupts while calling functions that read or modify global variables. Refer to the Registers section of this datasheet for more information, as needed.

General API

General API functions are used for run-time configuration of the component during active power mode.

These include, initializing, starting, stopping, reading from registers and writing to registers.

Functions

- void RTC_Start (void)
- void RTC_Stop (void)
- void RTC_Init (void)
- void RTC_SetUnixTime (uint64 unixTime)
- uint64 RTC_GetUnixTime (void)
- void RTC_SetPeriod (uint32 ticks, uint32 refOneSecTicks)
- uint32 RTC_GetPeriod (void)
- uint32 RTC_GetRefOneSec (void)
- void RTC_SetDateAndTime (uint32 inputTime, uint32 inputDate)
- void RTC_GetDateAndTime (RTC_DATE_TIME *dateTime)
- uint32 RTC_GetTime (void)
- uint32 RTC_GetDate (void)
- void RTC_SetAlarmDateAndTime (const RTC_DATE_TIME *alarmTime)
- void RTC_GetAlarmDateAndTime (RTC_DATE_TIME *alarmTimeDate)
- void RTC_SetAlarmMask (uint32 mask)
- uint32 RTC_GetAlarmMask (void)
- uint32 RTC_ReadStatus (void)
- uint32 RTC_GetAlarmStatus (void)
- void RTC_ClearAlarmStatus (void)
- void RTC_SetDSTStartTime (const RTC_DST_TIME *dstStartTime, RTC_DST_DATETYPE_ENUM type)
- void RTC_SetDSTStopTime (const RTC_DST_TIME *dstStopTime, RTC_DST_DATETYPE_ENUM type)
- uint32 RTC_ConvertBCDToDec (uint32 bcdNum)
- uint32 RTC_ConvertDecToBCD (uint32 decNum)
- void RTC_Update (void)
- void * RTC_SetAlarmHandler (void(*CallbackFunction)(void))
- static uint32 RTC_ConstructDate (uint32 month, uint32 day, uint32 year)
- static uint32 RTC_ConstructTime (uint32 timeFormat, uint32 stateAmPm, uint32 hour, uint32 min, uint32 sec)
- static uint32 RTC_LeapYear (uint32 year)
- static uint32 RTC_IsBitSet (uint32 var, uint32 mask)
- static uint32 RTC_GetSecond (uint32 inputTime)
- static uint32 RTC_GetMinutes (uint32 inputTime)
- static uint32 RTC_GetHours (uint32 inputTime)
- static uint32 RTC_GetAmPm (uint32 inputTime)
- static uint32 RTC_GetDay (uint32 date)
- static uint32 RTC_GetMonth (uint32 date)

- static uint32 RTC_GetYear (uint32 date)
- void RTC_UnixToDateTime (RTC_DATE_TIME *dateTime, uint64 unixTime, uint32 timeFormat)
- uint64 RTC_DateTimeToUnix (uint32 inputDate, uint32 inputTime)
- static void RTC_CySysRtcSetCallback (uint32 wdtNumber)
- static void RTC_CySysRtcResetCallback (uint32 wdtNumber)

Function Documentation

void RTC_Start (void)

Performs all the required calculations for the time and date registers and initializes the component along with the date and time selected in the customizer.

If "Implement RTC update manually" is disabled in the customizer and if WDT or DeepSleep timer is selected as a source in the clocks configuration window (low frequency clocks tab), attaches RTC_Update API to a corresponding ISR callback of WDT or DeepSleep Timer.

Note:

"Implement RTC update manually" checkbox is available for PSoC 4200L / PSoC 4100M / PSoC 4200M / PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4000S / PSoC 4100S and Analog Coprocessor.

void RTC_Stop (void)

Stops the time and date updates.

void RTC_Init (void)

Initializes or restores the component according to the customizer Configure dialogue settings.

It is not necessary to call RTC_Init() because RTC_Start() API calls this function and is the preferred method to begin component operation.

All registers are set to values according to the customizer Configure dialogue. The default date value, if not set by the user before this function call, is 12:00:00 AM January 1, 2000.

void RTC_SetUnixTime (uint64 *unixTime*)

Sets the time in the Unix/Epoch time format - the number of seconds elapsed from January 1, 1970 UTC 00:00 hrs.

Parameters:

time	The time value in the Unix time/Epoch time format.

uint64 RTC_GetUnixTime (void)

Returns the time in the Unix/Epoch time format - the number of seconds elapsed from January 1, 1970 UTC 00:00 hrs.

Returns:

Time The time value in the Unix time/Epoch time format.

void RTC_SetPeriod (uint32 *ticks*, uint32 *refOneSecTicks*)

Sets the RTC time update API period.

The user needs to pass the period as a number of ticks and also a reference number of ticks taken by the same clock source for one second. For instance, for a 32 kHz clock source and RTC period of 100 ms, the "ticks" value is 3200 and the "refOneSecTicks" value is 32000. This value is used to increment the time every time RTC_Update() API is called.

Parameters:

ticks	The clock period taken as a number of ticks.

refOneSecTicks	The reference number of ticks taken by the same clock source for one second (the input clock frequency in Hz).

uint32 RTC_GetPeriod (void)

Gets the RTC time update API period.

Returns:

Period The clock period taken as a number of ticks.

uint32 RTC_GetRefOneSec (void)

Gets the RTC time update API period.

Returns:

Period The reference number of ticks taken by the RTC clock source for one second.

void RTC_SetDateAndTime (uint32 *inputTime*, uint32 *inputDate*)

Sets the time and date values as the current time and date.

Parameters:

inputTime	The time value in the HH:MM:SS format. "HH"- The 2nd 8-bit MSB that denotes the hour value. (0-23 for the 24-hour format and 1-12 for the 12-hour format. The MSB bit of the value denotes AM/PM for the 12-hour format (0-AM and 1-PM). "MM" - The 3nd 8-bit MSB denotes the minutes value, the valid entries -> 0-59. "SS" - The 8-bit LSB denotes the seconds value, the valid entries -> 0-59. Each byte is in the BCD format. Invalid time entries retain the previously set values.
inputDate	The date value in the format selected in the customizer. For the MM/DD/YYYY format: "MM" - The 8-bit MSB denotes the month value in BCD, the valid entries -> 1-12 "DD" - The 2nd 8-bit MSB denotes a day of the month value in BCD, the valid entries -> 1-31. "YYYY" - The 16-bit LSB denotes a year in BCD, the valid entries -> 1900-2200. Each byte is in the BCD format. Invalid date entries retain the previously set values.

void RTC_GetDateAndTime (RTC_DATE_TIME **dateTime*)

Reads the current time and date.

Parameters:

dateTime	The pointer to the RTC_date_time structure to which time and date is returned.

uint32 RTC_GetTime (void)

Reads the current time.

Returns:

date The value of date in the user selected format. The date value is available in the BCD format.

Warning:

Using RTC_GetTime and RTC_GetDate API separately might result in errors when the time wraps around the end of the day. To avoid this, use RTC_GetDateAndTime API.

uint32 RTC_GetDate (void)

Reads the current time.

Returns:

time The time value in the format selected by the user (12/24 hr); The time value is available in the BCD format.

Note:

Using RTC_GetTime and RTC_GetDate API separately might result in errors when the time wraps around the end of the day. To avoid this, use RTC_GetDateAndTime API.

void RTC_SetAlarmDateAndTime (const RTC_DATE_TIME *alarmTime)

Writes the time and date values as the current alarm time and date.

Parameters:

alarmTime	The pointer to the RTC_date_time global structure where new values of the alarm time and date are stored, see RTC_DATE_TIME.

Note:

Invalid time entries are written with "00:00:00:00" for the 24-hour format and "AM 12:00:00:00" for the 12-hour format. Invalid date entries are written with a date equivalent to 01-JAN-2000.

void RTC_GetAlarmDateAndTime (RTC_DATE_TIME *alarmTimeDate)

Reads the current alarm time and date.

Parameters:

alarmTimeDate	The pointer to the RTC_date_time structure to which the alarm date and time are returned, see RTC_DATE_TIME.

void RTC_SetAlarmMask (uint32 *mask*)

Writes the Alarm Mask software register with one bit per time/date entry. The alarm is true when all masked time/date values match the Alarm values. Generated only if the alarm functionality is enabled.

Parameters:

mask	The Alarm Mask software register value. The values shown below can be OR'ed and passed as an argument as well, see Definitions for Alarm Mask software register.

uint32 RTC_GetAlarmMask (void)

Reads the Alarm Mask software register. Generated only if the alarm functionality is enabled, see Definitions for Alarm Mask software register.

uint32 RTC_ReadStatus (void)

Reads the Status software register, which has flags for DST (DST), Leap Year (LY), AM/PM (AM_PM).

Returns:

The values shown below are OR'ed and returned if more than one status bits are set, see Definitions of the RTC status values.

Note:

Reading the status without sync with the date and time read may cause an error due to a roll-over at AM/PM, the end of a year, the end of a day; RTC_GetDateAndTime() API is used to obtain the status and the status member of the returned structure can be checked with the masks.

uint32 RTC_GetAlarmStatus (void)

Returns the alarm status of RTC.

Returns:

The Alarm active status. This bit is high when the current time and date match the alarm time and date.
0 - The Alarm status is not active
1 - The Alarm status is active.

void RTC_ClearAlarmStatus (void)

Clears the alarm status of RTC.

Note:

The Alarm active (AA) flag clears after read. This bit will be set in the next alarm match event only. If Alarm is set on only minutes and the alarm minutes is 20 minutes - the alarm triggers once every 20th minute of every hour.

void RTC_SetDSTStartTime (const RTC_DST_TIME *dstStartTime, RTC_DST_DATETYPE_ENUM type)

Stores the DST Start time.

Only generated if DST is enabled. The date passed can be relative or fixed. For a relative date, the user needs to provide a valid day of a week, a week of a month and a month in the dstStartTime structure. For a fixed date, the user needs to enter a valid day of a month and a month in the dstStartTime structure. The hour value is optional and if invalid taken as 00 hrs. Invalid entries are not stored and the DST start date retains a previous value or no value at all.

Parameters:

dstStartTime	The DST Start time register value, see RTC_DST_TIME
type	Defines the DST operation mode, see RTC_DST_DATETYPE_ENUM.

void RTC_SetDSTStopTime (const RTC_DST_TIME *dstStopTime, RTC_DST_DATETYPE_ENUM type)

Stores the DST Stop time.

Only generated if DST is enabled. The date passed can be relative or fixed. For a relative date, the user needs to provide a valid day of a week, a week of a month and a month in the dstStopTime structure. For a fixed date, the user needs to enter a valid day of a month and a month in the dstSoptTime structure. The hour value is optional and if invalid taken as 00 hrs. Invalid entries are not stored and the DST start date retains a previous value or no value at all.

Parameters:

dstStopTime	DST Stop time register values, see RTC_DST_TIME
type	Defines the DST operation mode, see RTC_DST_DATETYPE_ENUM.

uint32 RTC_ConvertBCDToDec (uint32 bcdNum)

Converts a 4-byte BCD number into a 4-byte hexadecimal number. Each byte is converted individually and returned as an individual byte in the 32-bit variable.

Parameters:

bcdNum	A 4-byte BCD number. Each byte represents BCD. 0x11223344 -> 4 bytes 0x11, 0x22, 0x33 and 0x44 the in BCD format.

Returns:

decNum A 4-byte hexadecimal equivalent number of the BCD number. BCD number 0x11223344 -> returned hexadecimal number 0x0B16212C.

uint32 RTC_ConvertDecToBCD (uint32 decNum)

Converts a 4-byte hexadecimal number into a 4-byte BCD number. Each byte is converted individually and returned as an individual byte in the 32-bit variable.

Parameters:

decNum	A 4-byte hexadecimal number. Each byte is represented in hex. 0x11223344 -> 4 bytes 0x11, 0x22, 0x33 and 0x44 in the hex format.

Returns:

bcdNum - A 4-byte BCD equivalent of the passed hexadecimal number. Hexadecimal number 0x11223344 -> returned BCD number 0x17345168.

void RTC_Update (void)

This API updates the time registers and performs alarm/DST check.

This function increments the time/date registers by an input clock period. The period is set by RTC_SetPeriod() API or WDT period selected for RTC in the clocks configuration window (low frequency clocks tab) interface every time it is called.

API is automatically mapped to the callback slot of WDT or DeepSleep Timer and period, if the configuration is as follows: 1) Option "Implement RTC update manually" in the customizer is unchecked 2) One of WDTs or DeeSleep Timers is selected in the "Use for RTC" panel of the low frequency clocks tab 3) Option "Implementation by IDE" is selected in the "Timer (WDT) ISR" panel.

If option "Implement RTC update manually" is checked in the customizer or option "None" is selected in the "Use for RTC" panel, it is the user's responsibility: 1) to call this API from the clock ISR to be used as the RTC's input 2) set the period of the RTC through RTC_SetPeriod() API.

Note:

Updates the Unix time register, updates the alarm and DST status.

void* RTC_SetAlarmHandler (void(*)(void) *CallbackFunction*)

This API sets the function to be called when the alarm goes off / triggers. This API is generated only if the alarm functionality is enabled in the customizer.

Parameters:

CallbackFunction	The callback function address.

Returns:

A previous callback function address.

static CY_INLINE uint32 RTC_ConstructDate (uint32 *month*, uint32 *day*, uint32 *year*)[static]

Returns the date in the format used in APIs from individual elements passed (day. Month and year)

Parameters:

month	The month.
day	The day.
year	The year.

Returns:

The date in the format used in API.

static CY_INLINE uint32 RTC_ConstructTime (uint32 *timeFormat*, uint32 *stateAmPm*, uint32 *hour*, uint32 *min*, uint32 *sec*)[static]

Returns the time in the format used in APIs from individual elements passed (hour, min, sec etc)

Parameters:

timeFormat	The 12/24 hours time format, see Hour format definitions

stateAmPm	The AM/PM status, see AM/PM status definitions.
hour	The hour.
min	The minute.
sec	The second.

Returns:

Time in the format used in API.

static CY_INLINE uint32 RTC_LeapYear (uint32 *year*)[static]

Checks whether the year passed through the parameter is leap or no.

Parameters:

year	The year to be checked.

Returns:

0u - The year is not leap
1u - The year is leap.

static CY_INLINE uint32 RTC_IsBitSet (uint32 *var*, uint32 *mask*)[static]

Checks the state of a bit passed through parameter.

Parameters:

var	The variable to be checked.
mask	The mask for a bit to be checked.

Returns:

0u - Bit is not set.
1u - Bit is set.

static CY_INLINE uint32 RTC_GetSecond (uint32 *inputTime*)[static]

Returns the seconds value from the time value that is passed as a/the parameter.

Parameters:

inputTime	The time value.

Returns:

The seconds value.

static CY_INLINE uint32 RTC_GetMinutes (uint32 *inputTime*)[static]

Returns the minutes value from the time value that is passed as a/the parameter.

Parameters:

inputTime	The time value.

Returns:

The minutes value.

static CY_INLINE uint32 RTC_GetHours (uint32 *inputTime*)[static]

Returns the hours value from the time value that is passed as a/the parameter.

Parameters:

inputTime	The time value.

Returns:
The hours value.

static CY_INLINE uint32 RTC_GetAmPm (uint32 *inputTime*)[static]

Returns the AM/PM status from the time value that is passed as parameter.

Parameters:

inputTime	The time value.

Returns:
The am/pm period of day, see AM/PM status definitions.

static CY_INLINE uint32 RTC_GetDay (uint32 *date*)[static]

Returns the day value from the date value that is passed as parameter.

Parameters:

date	The date value.

Returns:
The day value.

static CY_INLINE uint32 RTC_GetMonth (uint32 *date*)[static]

Returns the month value from the date value that is passed as parameter.

Parameters:

date	The date value.

Returns:
The month value.

static CY_INLINE uint32 RTC_GetYear (uint32 *date*)[static]

Returns the year value from the date value that is passed as parameter.

Parameters:

date	The date value.

Returns:
The year value.

void RTC_UnixToDateTime (RTC_DATE_TIME **dateTime*, uint64 *unixTime*, uint32 *timeFormat*)

This is an internal function to convert the date and time from the UNIX time format into the regular time format

Parameters:

*RTC_DATE_TIME**	dateTime The time and date structure which will be updated time from unixTime value
unixTime	time in unix format
timeFormat	dst format of time, see RTC_DST_DATETYPE_ENUM

uint64 RTC_DateTimeToUnix (uint32 *inputDate*, uint32 *inputTime*)

This is an internal function to convert the date and time from the regular time format into the UNIX time format.

Parameters:

inputDate	The date in the selected in the customizer "date format"

inputTime	The time in the defined "time format"

Returns:

Returns the date and time in the UNIX format

static CY_INLINE void RTC_CySysRtcSetCallback (uint32 *wdtNumber*)[static]

This is an internal function that registers a callback for the RTC_Update() function by address "0".

Parameters:

wdtNumber	The number of the WDT or DeepSleep Timer to be used to pull the RTC_Update() function.

The callback registered before by address "0" is replaced by the RTC_Update() function.

static CY_INLINE void RTC_CySysRtcResetCallback (uint32 *wdtNumber*)[static]

This is an internal function that clears a callback by address "0".

Parameters:

wdtNumber	The number of the WDT or DeeSleep Timer to be cleared callback for.

The callback registered before by address "0" is replaced by the NULL pointer.

Global Variables

The following global variables are used in the component.

Variables

- uint8 RTC_initVar
- uint8 RTC_dstStatus
- volatile uint64 RTC_unixTime
- RTC_DST_TIME RTC_dstStartTime
- RTC_DST_TIME RTC_dstStopTime
- RTC_DATE_TIME RTC_currentTimeDate
- RTC_DATE_TIME RTC_alarmCfgTimeDate
- uint32 RTC_alarmCfgMask
- uint32 RTC_alarmCurStatus

Variable Documentation

uint8 RTC_initVar

Indicates whether the RTC has been initialized; The variable is initialized to 0 and set to 1 the first time RTC_Start() is called. This allows the component to restart without reinitialization after the first call to the RTC_Start() routine.

uint8 RTC_dstStatus

The DST start/stop status

volatile uint64 RTC_unixTime

The uint64 variable represents the standard Unix time (number of seconds elapsed from January 1, 1970 00:00 hours UTC) in 64-bit

RTC_DST_TIME RTC_dstStartTime

The values for the time and date of the DST start

RTC_DST_TIME RTC_dstStopTime

The values for the time and date of the DST stop

RTC_DATE_TIME RTC_currentTimeDate

The last updated time and date values are stored in this structure (update happens in Get time/date APIs)

RTC_DATE_TIME RTC_alarmCfgTimeDate

The alarm time and date values are stored in this variable

uint32 RTC_alarmCfgMask

This variable is used to mask alarm events; mask seconds alarm, mask minutes alarm, and so on. It will have bit masks for each time item masking that item for alarm generation

uint32 RTC_alarmCurStatus

This variable is used to indicate current active alarm status per time item used in the alarm; whether seconds alarm is active, minute's alarm is active, and so on. It will have bit masks for each time item (seconds, minutes, hours, day, and so on) showing the status

API Constants

Component API functions are designed to work with pre-defined enumeration values.

These values should be used with the API functions that reference them.

- Day of the week definitions
- DST Week of month setting constants definitions
- Month definitions
- AM/PM status definitions
- Hour format definitions
- Number of days in month definitions
- Definitions of the RTC status values
- Definitions for Alarm Mask software register

Day of the week definitions

Macros

- #define RTC_SUNDAY (1)
- #define RTC_MONDAY (2)
- #define RTC_TUESDAY (3)
- #define RTC_WEDNESDAY (4)
- #define RTC_THURSDAY (5)
- #define RTC_FRIDAY (6)
- #define RTC_SATURDAY (7)

Macro Definition Documentation

#define RTC_SUNDAY (1)

Sequential number of Sunday in the week

#define RTC_MONDAY (2)

Sequential number of Monday in the week

#define RTC_TUESDAY (3)

Sequential number of Tuesday in the week

#define RTC_WEDNESDAY (4)

Sequential number of Wednesday in the week

#define RTC_THURSDAY (5)

Sequential number of Thursday in the week

#define RTC_FRIDAY (6)

Sequential number of Friday in the week

#define RTC_SATURDAY (7)

Sequential number of Saturday in the week

DST Week of month setting constants definitions

Macros

- #define RTC_FIRST (1u)
- #define RTC_SECOND (2u)
- #define RTC_THIRD (3u)
- #define RTC_FOURTH (4u)
- #define RTC_FIFTH (5u)
- #define RTC_LAST (6u)

Macro Definition Documentation

#define RTC_FIRST (1u)

First week in the month

#define RTC_SECOND (2u)

Second week in the month

#define RTC_THIRD (3u)

Third week in the month

#define RTC_FOURTH (4u)

Fourth week in the month

#define RTC_FIFTH (5u)

Fifth week in the month

#define RTC_LAST (6u)

Last week in the month

Month definitions

Macros

- #define RTC_JANUARY (1u)
- #define RTC_FEBRUARY (2u)
- #define RTC_MARCH (3u)
- #define RTC_APRIL (4u)
- #define RTC_MAY (5u)
- #define RTC_JUNE (6u)
- #define RTC_JULY (7u)
- #define RTC_AUGUST (8u)
- #define RTC_SEPTEMBER (9u)
- #define RTC_OCTOBER (10u)
- #define RTC_NOVEMBER (11u)
- #define RTC_DECEMBER (12u)

Macro Definition Documentation

#define RTC_JANUARY (1u)

Sequential number of January in the year

#define RTC_FEBRUARY (2u)

Sequential number of February in the year

#define RTC_MARCH (3u)

Sequential number of March in the year

#define RTC_APRIL (4u)

Sequential number of April in the year

#define RTC_MAY (5u)

Sequential number of May in the year

#define RTC_JUNE (6u)

Sequential number of June in the year

#define RTC_JULY (7u)

Sequential number of July in the year

#define RTC_AUGUST (8u)

Sequential number of August in the year

#define RTC_SEPTEMBER (9u)

Sequential number of September in the year

#define RTC_OCTOBER (10u)

Sequential number of October in the year

#define RTC_NOVEMBER (11u)

Sequential number of November in the year

#define RTC_DECEMBER (12u)

Sequential number of December in the year

AM/PM status definitions

Definitions for 12 hour format for indicating the AM/PM period of day

Macros

- #define RTC_AM (0u)
- #define RTC_PM (1u)

Macro Definition Documentation

#define RTC_AM (0u)

AM period of day

#define RTC_PM (1u)

PM period of day

Hour format definitions

Macros

- #define RTC_12_HOURS_FORMAT (1u)
- #define RTC_24_HOURS_FORMAT (0u)

Macro Definition Documentation

#define RTC_12_HOURS_FORMAT (1u)

The 24 hour format

#define RTC_24_HOURS_FORMAT (0u)

The 12 hour (AM/PM) format

Number of days in month definitions

Macros

- #define RTC_DAYS_IN_JANUARY (31u)
- #define RTC_DAYS_IN_FEBRUARY (28u)
- #define RTC_DAYS_IN_MARCH (31u)
- #define RTC_DAYS_IN_APRIL (30u)
- #define RTC_DAYS_IN_MAY (31u)
- #define RTC_DAYS_IN_JUNE (30u)
- #define RTC_DAYS_IN_JULY (31u)
- #define RTC_DAYS_IN_AUGUST (31u)
- #define RTC_DAYS_IN_SEPTEMBER (30u)
- #define RTC_DAYS_IN_OCTOBER (31u)
- #define RTC_DAYS_IN_NOVEMBER (30u)
- #define RTC_DAYS_IN_DECEMBER (31u)

Macro Definition Documentation

#define RTC_DAYS_IN_JANUARY (31u)

Number of days in January

#define RTC_DAYS_IN_FEBRUARY (28u)

Number of days in February

#define RTC_DAYS_IN_MARCH (31u)

Number of days in March

#define RTC_DAYS_IN_APRIL (30u)

Number of days in April

#define RTC_DAYS_IN_MAY (31u)

Number of days in May

#define RTC_DAYS_IN_JUNE (30u)

Number of days in June

#define RTC_DAYS_IN_JULY (31u)

Number of days in July

#define RTC_DAYS_IN_AUGUST (31u)

Number of days in August

#define RTC_DAYS_IN_SEPTEMBER (30u)

Number of days in September

#define RTC_DAYS_IN_OCTOBER (31u)

Number of days in October

#define RTC_DAYS_IN_NOVEMBER (30u)

Number of days in November

#define RTC_DAYS_IN_DECEMBER (31u)

Number of days in December

Definitions of the RTC status values

Definitions for status software register, which has flags for DST (DST), Leap Year (LY), AM/PM (AM_PM).

Macros

- #define RTC_STATUS_DST (1uL << 1u)
- #define RTC_STATUS_LY (1uL << 3u)
- #define RTC_STATUS_AM_PM (1uL << 4u)

Macro Definition Documentation

#define RTC_STATUS_DST (1uL << 1u)

Status of Daylight Saving Time. This bit goes high when the current time and date match the DST time and date and the time is incremented. This bit goes low after the DST interval and the time is decremented.

#define RTC_STATUS_LY (1uL << 3u)

Status of Leap Year. This bit goes high when the current year is a leap year

#define RTC_STATUS_AM_PM (1uL << 4u)

Status of Current Time. This bit is low from midnight to noon and high from noon to midnight.

Definitions for Alarm Mask software register

Definitions for Alarm Mask software register. These masks allow matching the alarm value register with the current value register.

Macros

- #define RTC_ALARM_SEC_MASK (0x00000001uL)
- #define RTC_ALARM_MIN_MASK (0x00000002uL)
- #define RTC_ALARM_HOUR_MASK (0x00000004uL)
- #define RTC_ALARM_DAYOFWEEK_MASK (0x00000008uL)
- #define RTC_ALARM_DAYOFMONTH_MASK (0x00000010uL)
- #define RTC_ALARM_MONTH_MASK (0x00000020uL)
- #define RTC_ALARM_YEAR_MASK (0x00000040uL)

Macro Definition Documentation

#define RTC_ALARM_SEC_MASK (0x00000001uL)

The second alarm mask allows matching the alarm second register with the current second register.

#define RTC_ALARM_MIN_MASK (0x00000002uL)

The minute alarm mask allows matching the alarm minute register with the current minute register.

#define RTC_ALARM_HOUR_MASK (0x00000004uL)

The hour alarm mask allows matching the alarm hour register with the current hour register.

#define RTC_ALARM_DAYOFWEEK_MASK (0x00000008uL)

The day of the week alarm mask allows matching the alarm hour register with the current day of the week register.

#define RTC_ALARM_DAYOFMONTH_MASK (0x00000010uL)

The day of the Month alarm mask allows matching the alarm hour register with the current day of the Month register.

#define RTC_ALARM_MONTH_MASK (0x00000020uL)

The month alarm mask allows matching the alarm hour register with the current month register.

#define RTC_ALARM_YEAR_MASK (0x00000040uL)

The year alarm mask allows matching the alarm hour register with the current year register.

Data Structures

Data Structures are used to group related elements.

The following data structures are used by the component API functions.

Data Structures

- struct RTC_DATE_TIME
- struct RTC_DST_TIME

Data Structure Documentation

struct RTC_DATE_TIME

Data Fields

- uint32 time
- uint32 date
- uint32 dayOfWeek
- uint32 status

Field Documentation

uint32 RTC_DATE_TIME::time

Time in the format used in API

uint32 RTC_DATE_TIME::date

Date in the format used in API

uint32 RTC_DATE_TIME::dayOfWeek

Day of the week, see Day of the week definitions

uint32 RTC_DATE_TIME::status

RTC status, see Definitions of the RTC status values

struct RTC_DST_TIME

Data Fields

- uint32 hour
- uint32 dayOfWeek
- uint32 dayOfMonth
- uint32 weekOfMonth
- uint32 month
- uint8 timeFormat

Field Documentation

uint32 RTC_DST_TIME::hour
Hour value

uint32 RTC_DST_TIME::dayOfWeek
Day of the week, see Day of the week definitions

uint32 RTC_DST_TIME::dayOfMonth
Day of the month

uint32 RTC_DST_TIME::weekOfMonth
Week of the month, see DST Week of month setting constants definitions

uint32 RTC_DST_TIME::month
Month value, see Month definitions

uint8 RTC_DST_TIME::timeFormat
The DST operation mode, see RTC_DST_DATETYPE_ENUM

Enumerations

Enumerations used by the component API functions.
The following enumerations are used by the component API functions.

Enumerations

- enum RTC_DST_DATETYPE_ENUM

Enumeration Type Documentation

enum RTC_DST_DATETYPE_ENUM
Daylight saving Time format enumeration
Enumerator
RTC_DST_DATE_RELATIVE Relative DST format
RTC_DST_DATE_FIXED Fixed DST format

Sample Firmware Source Code

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

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

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)
1.1	R	This rule states that code shall conform to C ISO/IEC 9899:1990 standard.	Nesting of control structures (statements) exceeds 15 - program does not conform strictly to ISO:C90. In practice, most compilers will support a much more liberal nesting limit and therefore this limit may only be relevant when strict conformance is required. By comparison, ISO:C99 specifies a limit of 127 "nesting levels of blocks.
1.2, 11.1	R	Cast between a pointer to object and a pointer to function.	The reason of this violation I the (void*) return type in the RTC_SetAlarmHandler() function.
12.4	R	Right hand operand of '&&' or '||' is an expression with possible side effects.	The reason of this violation that the operand is declared with the "volatile" modifier.
13.2	A	The result of this logical operation is always 'true'.	Actually the result of operation can be false since the unixTime variable changes in the ISR.
13.7	R	Boolean operations whose results are invariant shall not be permitted.	Actually the result of operation can be false since the unixTime variable changes in the ISR.

API Memory Usage

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

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

PSoC 4 (GCC)

Configuration	Flash Bytes	SRAM Bytes
Default	1436	48
RTC with DST	2204	120
RTC with alarm	2988	88
RTC with DST and alarm	3888	152

Functional Description

Time and date

All time and date registers are as accessible as software variables. The time and date change is based on an interrupt event that drives the call of the RTC_Update() function. The following variables are provided:

- Sec – Seconds: 0 to 59
- Min – Minutes: 0 to 59
- Hour – Hours (24 or 12 hours format): 0 to 23 or 0 to 12
- DayOfMonth – Day of month: 1 to 31
- DayOfWeek – Day of week: 1 to 7. Sunday - 1, Monday - 2, … , Saturday – 7.
- Month – Month: 1 to 12
- Year – Year: 1900 to 2200 (the actual range is 1 to 65536)

The DayOfWeek is calculated using Zeller's congruence. Zeller's congruence is a simple algorithm optimized for integer math that calculates the day of the week based on year, month, and day of the month. It accounts for leap years and leap centuries.

When you call the RTC_Start() function, an RTC_Init() function is called and all required flags and date calculations are executed. This includes all variables that need calculation:

- DayOfWeek
- LY
- AM_PM
- DST

Ticks counter updating behavior

The RTC_P4 Component is a software Component. Time values, such as seconds, minutes, etc., are incremented based on the RTC interrupt. The RTC interrupt period depends on the RTC_P4 input clock source. If you choose to use sources other than a WDT or DeepSleep Timer, then you need to set the period of the input clock source manually.

To set the period of a 1-second increment manually, call the RTC_P4_SetPeriod() function and provide two parameters: the *tick value* and the *tick reference number*. The second value is the *reference number of ticks*.

The RTC_P4 Component has an internal counter, the *tick counter*, that is incremented by the *tick value* on every RTC_P4 interrupt in the RTC_P4_Update() function. The seconds value is incremented in the RTC_P4_Update() function when the *tick counter value* is equal to or more than the *tick reference number*. The tick counter is decremented by the *tick reference number* after the seconds value is incremented.

For example, for a 32-kHz clock source and an RTC period of 100 ms, the *tick value* is 3200 and the *tick reference number* is 32000. On every RTC interrupt, the *tick counter value* is incremented by 3200. When the *tick counter value* reaches 32000, the *counter value* is decremented by 32000:

tick counter value = tick counter value – 32000 (and the seconds value is incremented)

For more details, refer to the RTC_P4_SetPeriod() and RTC_P4_Update() function descriptions.

Alarm Function

The alarm function provides for seconds, minutes, hours, days of the month, days of the week, month, year, and day of the year. The same variable names are provided for alarm settings. You can set any or all of these alarm settings and configure which of these settings are used in tripping the alarm.

Daylight Savings Time

To enable the DST feature, select the check box on the Configure dialog (see the Component Parameters section of this datasheet). DST is implemented as a set of API update times, dates, and durations. If the current time and date match the start of DST time and date, then the DST flag is set and the time is incremented by the set duration.

The start and stop date of DST can be given as fixed or relative. The relative date converts to the fixed one and is checked against the current time as if it were an alarm function. An example of a fixed date is "24 March." An example of a relative date is "fourth Sunday in May."

The conversion of a relative date to a fixed date is implemented as a separate function.

The DST variables for start and stop time and date are as follows:

- Hour – Hour: 0 to 23 (fixed and relative)
- DayOfWeek – Day of week 1 to 7. Sunday - 1, Monday - 2, …, Saturday – 7 (relative)
- Week – Week in month: 1 to 5 (relative)
- DayOrWeekOfMonth – Day of month: (1 to 31) or week of month (1-FIRST, 2-SECOND, 3-THIRD, 4-FOURTH, 5-FIFTH, 6-LAST)
- Month – Month: 1 to 12 (fixed and relative)

Registers

Status Register

The status register is a read-only register that contains various RTC_P4 status bits. This value can be read using the RTC_ReadStatus() function. There are several bit-field masks defined for the status register. The #defines are available in the generated header file (.h) as follows:

- **RTC_STATUS_DST** – Status of Daylight Saving Time. This bit goes high when the current time and date match DST time and date and the time is incremented. This bit goes low after the DST interval and the time is decremented.
- **RTC_STATUS_LY** – Status of leap year. This bit goes high when the current year is a leap year.
- **RTC_STATUS_AM_PM** – Status of current time. This bit is low from midnight to noon and high from noon to midnight.

Alarm Mask Register

The alarm mask register is a write-only register that allows you to control the alarm bit in the status register. The alarm bit is generated by ORing the masked bit fields within this register. This register is written with the RTC_WriteAlarmMask() function call. When writing the alarm mask register you must use the bit-field definitions as defined in the header (.h) file. The definitions for the alarm mask register are as follows:

- **RTC_ALARM_SEC_MASK** – The second alarm mask allows you to match the alarm second register with the current second register.
- **RTC_ALARM_MIN_MASK** – The minute alarm mask allows you to match the alarm minute register with the current minute register.
- **RTC_ALARM_HOUR_MASK** – The hour alarm mask allows you to match the alarm hour register with the current hour register.
- **RTC_ALARM_DAYOFWEEK_MASK** – The day of week alarm mask allows you to match the alarm day of week register with the current day of week register.
- **RTC_ALARM_DAYOFMONTH_MASK** – The day of month alarm mask allows you to match the alarm day of month register with the current day of month register.
- **RTC_ALARM_MONTH_MASK** – The month alarm mask allows you to match the alarm month register with the current month register.
- **RTC_ALARM_YEAR_MASK** – The year alarm mask allows you to match the alarm year register with the current year register.

Conditional Compilation Information

The RTC_P4 API requires one conditional compile definition to handle daylight savings time functionality. The DST Alarm related functions are conditionally compiled only if this option is enabled in the Configure dialog. The software should never use this parameter directly. Instead, use the symbolic name defined.

- **RTC _INITIAL_DST_STATUS** – The daylight savings time functionality enable define is assigned to be equal to the "Daylight Savings (DST) Settings" value (from the Configure dialog) at build time. It is used throughout the API to compile data saving time functions.
- **RTC _INITIAL_ALARM_STATUS** – The alarm functionality enable define is assigned to be equal to the "Enable alarm functionality" value (from the Configure dialog) at build time. It is used throughout the API to compile data saving time functions.

Time register

This register contains the time value in the "HH:MM:SS" format. Each number is in BCD format.

The defines that contains offset to each time value are as follows:

- **RTC _TIME_FORMAT_OFFSET** – Offset to the bit that defines the current time format (12-hour or 24-hour).
- **RTC _PERIOD_OF_DAY_OFFSET** – Offset to the bit that indicates the period of day in 12-hour time format (AM or PM).
- **RTC _HOUTS_OFFSET** - Offset to the field that contains the hour value in BCD format.
- **RTC _MINUTES_OFFSET** – Offset to the field that contains the minute value in BCD format.
- **RTC _SECONDS_OFFSET** – Offset to the field that contains the second value in BCD format.

Date register

This register contains the time value in the format that is selected in customizer. Each number is in BCD format. The defines that contains offset to each time value are as follows:

- **RTC_MONTH_OFFSET –** Offset to the field that contains the month value in BCD format.
- **RTC_DAY_OFFSET –** Offset to the field that contains the day value in the BCD format.
- **RTC_YEAR_OFFSET –** Offset to the field that contains the year value in the BCD format.

Resources

The RTC_P4 Component does not utilize any hardware resources by itself.

DC and 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.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.20	Add information about how ticks counter and seconds are updated in the Component. Improved time update behavior. Improved function to convert the date and time from the UNIX time format into regular time format. Added information about RTC clock sources WDT and DeepSleep Timers in General Description	Documentation improvements. Added Ticks counter updating behavior section Now in the RTC_P4_Update() function, when seconds value is incremented, the ticks counter is not reset to zero but decremented on reference value. Now callback function is called after the new alarm time was obtained. Improved functionality. Provide more information about RTC clock sources.
1.10.b	Removed redundant information about deleted StartOfWeek parameter in the Time and date and Daylight Savings Time sections. Added additional information about 'RTC_sel Mux' and 'Timer (WDT) ISR' panel in Implement RTC update manually section. Removed characterization note because it does not apply to this Component.	To synchronize information with current Component implementation. Provide more detailed information about where 'RTC_sel Mux' and 'Timer (WDT) ISR' panel can be configured.
1.10.a	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 the ability to drive the RTC by Timers. Timers are available in PSoC 4000S, PSoC 4100S, and PSoC Analog Coprocessor devices.	To have accurate RTC in new PSoC 4 devices.
1.0.a	Updated Description for RTC_SetAlarmDateAndTime() function and "Implement RTC update manually" check box	To make it more clear.
1.0	Initial Component version.	

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

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

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

