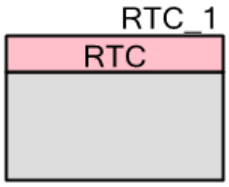
Real-Time Clock (RTC)
2.0

Features



- Multiple alarm options
- Multiple overflow options
- Daylight Savings Time (DST) option

General Description

The Real-Time Clock (RTC) component provides accurate time and date information for the system. The time and date are updated every second based on a one pulse per second interrupt from a 32.768-kHz crystal. Clock accuracy is based on the crystal provided and is typically 20 ppm.

The RTC keeps track of the second, minute, hour, day of the week, day of the month, day of the year, month, and year. The day of the week is automatically calculated from the day, month, and year. Daylight savings time may be optionally enabled and supports any start and end date, as well as a programmable saving time. The start and end dates may be absolute like 24 March or relative like the second Sunday in May.

The Alarm provides match detection for a second, minute, hour, day of week, day of month, day of year, 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.

User code stubs are provided for periodic code execution based on each of the primary time intervals. Timer intervals are provided at one second, one minute, one hour, one day, one week, one month, and one year.

When to Use an RTC Component

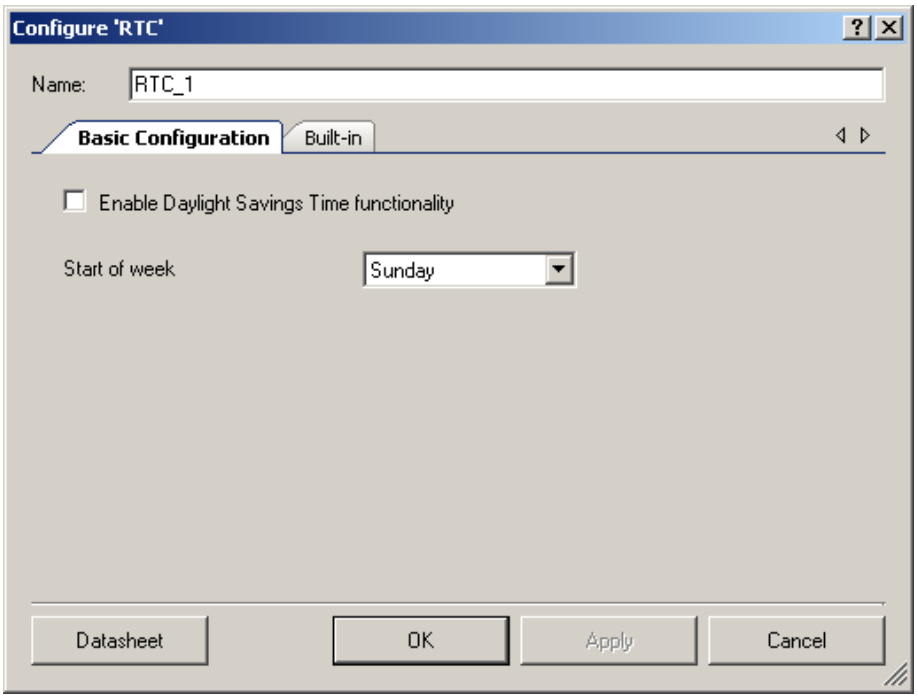
Use the RTC component when the system requires the current time or date. You can also use the RTC when you do not need the current time and date but you need accurate timing of events with one-second resolution.

Input/Output Connections

The RTC component does not have input or output connections.

Component Parameters

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



The RTC component contains the following options:

Enable Daylight Savings Time functionality

This parameter allows you to choose whether the daylight savings time functionality is enabled in the RTC component. The default value is cleared (false).

Start of week

The **Start of week** parameter allows you to choose start day of the week. Options include: **Sunday** (default), **Monday**, **Tuesday**, **Wednesday**, **Thursday**, **Friday**, and **Saturday**.

Clock Selection

You should provide a 32.768-kHz clock from an external crystal oscillator. The accuracy of this component is defined by the accuracy of the connected external clock source. Refer to the Clock Editor section of the PSoC Creator Help for information about how to connect and configure the built-in XTAL_32KHZ clock in your design.

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.

Functions

Function	Description
RTC_Start()	Enables the RTC component
RTC_Stop()	Stops RTC component operation
RTC_EnableInt()	Enables interrupts of RTC component
RTC_DisableInt()	Disables interrupts of the RTC component; time and date stop running
RTC_WriteTime()	Writes time and date values as current time and date
RTC_ReadTime()	Reads the current time and date
RTC_WriteSecond()	Writes Sec software register value
RTC_WriteMinute()	Writes Min software register value
RTC_WriteHour()	Writes Hour software register value
RTC_WriteDayOfMonth()	Writes DayOfMonth software register value
RTC_WriteMonth()	Writes Month software register value
RTC_WriteYear()	Writes Year software register value
RTC_WriteAlarmSecond()	Writes Alarm Sec software register value
RTC_WriteAlarmMinute()	Writes Alarm Min software register value
RTC_WriteAlarmHour()	Writes Alarm Hour software register value
RTC_WriteAlarmDayOfMonth()	Writes Alarm DayOfMonth software register value
RTC_WriteAlarmMonth()	Writes Alarm Month software register value
RTC_WriteAlarmYear()	Writes Alarm Year software register value
RTC_WriteAlarmDayOfWeek()	Writes Alarm DayOfWeek software register value

Function	Description
RTC_WriteAlarmDayOfYear()	Writes Alarm DayOfYear software register value
RTC_ReadSecond()	Reads Sec software register value
RTC_ReadMinute()	Reads Min software register value
RTC_ReadHour()	Reads Min software register value
RTC_ReadDayOfMonth()	Reads DayOfMonth software register value
RTC_ReadMonth()	Reads Month software register value
RTC_ReadYear()	Reads Year software register value
RTC_ReadAlarmSecond()	Reads Alarm Sec software register value
RTC_ReadAlarmMinute()	Reads Alarm Min software register value
RTC_ReadAlarmHour()	Reads Alarm Hour software register value
RTC_ReadAlarmDayOfMonth()	Reads Alarm DayOfMonth software register value
RTC_ReadAlarmMonth()	Reads Alarm Month software register value
RTC_ReadAlarmYear()	Reads Alarm Year software register value
RTC_ReadAlarmDayOfWeek()	Reads Alarm DayOfWeek software register value
RTC_ReadAlarmDayOfYear()	Reads Alarm DayOfYear software register value
RTC_WriteAlarmMask()	Writes the Alarm Mask software register with one bit per time/date entry
RTC_WriteIntervalMask()	Configures what interval handlers will be called from the RTC ISR
RTC_ReadStatus()	Reads the Status software register, which has flags for DST (DST), Leap Year (LY), AM/PM (AM_PM), and Alarm active (AA)
RTC_WriteDSTMode()	Writes the DST Mode software register
RTC_WriteDSTStartHour()	Writes the DST Start Hour software register
RTC_WriteDSTStartDayOfMonth()	Writes the DST Start DayOfMonth software register
RTC_WriteDSTStartMonth()	Writes the DST Start Month software register
RTC_WriteDSTStartDayOfWeek()	Writes the DST Start DayOfWeek software register
RTC_WriteDSTStartWeek()	Writes the DST Start Week software register
RTC_WriteDSTStopHour()	Writes the DST Stop Hour software register
RTC_WriteDSTStopDayOfMonth()	Writes the DST Stop DayOfMonth software register
RTC_WriteDSTStopMonth()	Writes the DST Stop Month software register
RTC_WriteDSTStopDayOfWeek()	Writes the DST Stop DayOfWeek software register
RTC_WriteDSTStopWeek()	Writes the DST Stop Week software register
RTC_WriteDSTOffset()	Writes the DST Offset register

Function	Description
RTC_Init()	Initializes and restores default configuration provided with the customizer
RTC_Enable()	Enables the interrupts, one pulse per second, and interrupt generation on OPPS event

void RTC_Start(void)

Description: Enables the RTC component. This function configures the counter, sets up interrupts, does all required calculation, and starts the counter.

Parameters: None

Return Value: None

Side Effects: Enables the one pulse per second and central time wheel signals to wake up the device from low-power modes (Sleep and Alternate Active) and leaves them enabled.

void RTC_Stop(void)

Description: Stops RTC component operation.

Parameters: None

Return Value: None

Side Effects: Leaves the one pulse per second and the central time wheel signals enabled, which wakes up the device from low-power modes (Sleep and Alternate Active).

void RTC_EnableInt(void)

Description: Enables interrupts from the RTC component.

Parameters: None

Return Value: None

Side Effects: None

void RTC_DisableInt(void)

Description: Disables interrupts from the RTC component. Time and date stop running.

Parameters: None

Return Value: None

Side Effects: None

RTC_TIME_DATE* RTC_ReadTime(void)

Description: Reads the current time and date.

Parameters: None

Return Value: Pointer to the RTC_TIME_DATE.

Side Effects: None

void RTC_WriteTime(const RTC_TIME_DATE * timeDate)

Description: Writes the time and date values as current time and date. Only passes the Second, Minute, Hour, Month, Day of Month, and Year.

Parameters: timeDate: Pointer to the RTC_TIME_DATE global structure where new values of time and date are stored

Return Value: None

Side Effects: The RTC component's interrupt should be disabled before calling this function and enabled afterwards to avoid RTC counter increment in the middle of the time and date writing.

void RTC_WriteSecond(uint8 second)

Description: Writes the Sec software register value.

Parameters: second: Seconds value

Return Value: None

Side Effects: None

void RTC_WriteMinute(uint8 minute)

Description: Writes the Min software register value.

Parameters: minute: Minutes value

Return Value: None

Side Effects: None

void RTC_WriteHour(uint8 hour)

Description: Writes the Hour software register value.

Parameters: hour: Hours value

Return Value: None

Side Effects: None

void RTC_WriteDayOfMonth(uint8 dayOfMonth)

Description: Writes the DayOfMonth software register value.

Parameters: dayOfMonth: DayOfMonth value

Return Value: None

Side Effects: None

void RTC_WriteMonth(uint8 month)

Description: Writes the Month software register value.

Parameters: month: Month value

Return Value: None

Side Effects: None

void RTC_WriteYear(uint16 year)

Description: Writes the Year software register value.

Parameters: year: Years value

Return Value: None

Side Effects: None

void RTC_WriteAlarmSecond(uint8 second)

Description: Writes the Alarm Sec software register value.

Parameters: second: Alarm Seconds value

Return Value: None

Side Effects: None

void RTC_WriteAlarmMinute(uint8 minute)

Description: Writes the Alarm Min software register value.

Parameters: minute: Alarm Minutes value

Return Value: None

Side Effects: None

void RTC_WriteAlarmHour(uint8 hour)

Description: Writes the Alarm Hour software register value.

Parameters: hour: Alarm Hours value

Return Value: None

Side Effects: None

void RTC_WriteAlarmDayOfMonth(uint8 dayOfMonth)

Description: Writes the Alarm DayOfMonth software register value.

Parameters: dayOfMonth: Alarm Day Of Month value

Return Value: None

Side Effects: None

void RTC_WriteAlarmMonth(uint8 month)

Description: Writes the Alarm Month software register value.

Parameters: month: Alarm Months value

Return Value: None

Side Effects: None

void RTC_WriteAlarmYear(uint16 year)

Description: Writes the Alarm Year software register value.

Parameters: year: Alarm Years value

Return Value: None

Side Effects: None

void RTC_WriteAlarmDayOfWeek(uint8 dayOfWeek)

Description: Writes the Alarm DayOfWeek software register value.

Parameters: dayOfWeek: Alarm Day Of Week value

Return Value: None

Side Effects: None

void RTC_WriteAlarmDayOfYear(uint16 dayOfYear)

Description: Writes the Alarm DayOfYear software register value.

Parameters: dayOfYear: Alarm Day Of Year value

Return Value: None

Side Effects: None

uint8 RTC_ReadSecond(void)

Description: Reads the Sec software register value.

Parameters: None

Return Value: None

Side Effects: None

uint8 RTC_ReadMinute(void)

Description: Reads the Min software register value.

Parameters: None

Return Value: Returns the current minute's value.

Side Effects: None

uint8 RTC_ReadHour(void)

Description: Reads the Min software register value.

Parameters: None

Return Value: Returns the current hour's value.

Side Effects: None

uint8 RTC_ReadDayOfMonth(void)

Description: Reads the DayOfMonth software register value.

Parameters: None

Return Value: Returns the current day of month's value.

Side Effects: None

uint8 RTC_ReadMonth(void)

Description: This function reads the Month software register value.

Parameters: None

Return Value: Returns the current month's value.

Side Effects: None

uint16 RTC_ReadYear(void)

Description: Reads the Year software register value.

Parameters: None

Return Value: Returns the current year's value.

Side Effects: None

uint8 RTC_ReadAlarmSecond(void)

Description: Reads the Alarm Sec software register value.

Parameters: None

Return Value: Returns the current second's alarm value.

Side Effects: None

uint8 RTC_ReadAlarmMinute(void)

Description: Reads the Alarm Min software register value.

Parameters: None

Return Value: Returns the current minute's alarm value.

Side Effects: None

uint8 RTC_ReadAlarmHour(void)

Description: Reads the Alarm Hour software register value.

Parameters: None

Return Value: Returns the current hour's alarm value is returned

Side Effects: None

uint8 RTC_ReadAlarmDayOfMonth(void)

Description: Reads the Alarm DayOfMonth software register value.

Parameters: None

Return Value: Returns the current day of month's alarm value.

Side Effects: None

uint8 RTC_ReadAlarmMonth(void)

Description: Reads the Alarm Month software register value.

Parameters: None

Return Value: Returns the current month's alarm value.

Side Effects: None

uint16 RTC_ReadAlarmYear(void)

Description: Reads the Alarm Year software register value.

Parameters: None

Return Value: Returns the current year's alarm value.

Side Effects: None

uint8 RTC_ReadAlarmDayOfWeek(void)

Description: Reads the Alarm DayOfWeek software register value.

Parameters: None

Return Value: Returns the current day of week's alarm value.

Side Effects: None

uint16 RTC_ReadAlarmDayOfYear(void)

Description: Reads the Alarm DayOfYear software register value.

Parameters: None

Return Value: Returns the current day of year's alarm value.

Side Effects: None

void RTC_WriteAlarmMask(uint8 mask)

Description: Writes the Alarm Mask software register with one bit per time/date entry. Alarm true when all masked time/date values match Alarm values.

Parameters: mask: Alarm Mask software register value. See the Alarm Mask Register section for more information about this parameter.

Return Value: None.

Side Effects: None.

void RTC_WriteIntervalMask(uint8 mask)

Description: Configures what interval handlers will be called from the RTC ISR. See the Interrupt Service Routines section for information about how to use this functionality.

Parameters: mask: Interval Mask software register value. See the Interval Mask Register section for more information about this parameter.

Return Value: None.

Side Effects: None.

uint8 RTC_ReadStatus(void)

Description: Reads the Status software register, which has flags for DST (DST), Leap Year (LY), AM/PM (AM_PM), and Alarm active (AA).

Parameters: None

Return Value: Current component's status with the active alarm bit cleared. Refer to the Status Register section for more information about the return values.

Side Effects: Alarm active (AA) flag clear after read.

void RTC_WriteDSTMode(uint8 mode)

Description: Writes the DST Mode software register That enables or disables DST changes and sets the date mode to fixed date or relative date. Only generated if DST is enabled.

Parameters: mode: DST Mode software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStartHour(uint8 hour)

Description: Writes the DST Start Hour software register. Used for absolute date entry. Only generated if DST is enabled.

Parameters: hour: DST Start Hour software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStartDayOfMonth(uint8 dayOfMonth)

Description: Writes the DST Start DayOfMonth software register. Used for absolute date entry. Only generated if DST is enabled.

Parameters: dayOfMonth: DST Start DayOfMonth software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStartMonth(uint8 month)

Description: Writes the DST Start Month software register. Used for absolute date entry. Only generated if DST is enabled.

Parameters: month: DST Start Month software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStartDayOfWeek(uint8 dayOfWeek)

Description: Writes the DST Start DayOfWeek software register. Used for relative date entry. Only generated if DST is enabled.

Parameters: dayOfWeek: DST Start DayOfWeek software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStartWeek(uint8 week)

Description: Writes the DST Start Week software register. Used for relative date entry. Only generated if DST is enabled.

Parameters: Week: DST Start Week software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStopHour(uint8 hour)

Description: Writes the DST Stop Hour software register. Used for absolute date entry. Only generated if DST is enabled.

Parameters: hour: DST Stop Hour software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStopDayOfMonth(uint8 dayOfMonth)

Description: Writes the DST Stop DayOfMonth software register. Used for absolute date entry. Only generated if DST is enabled.

Parameters: dayOfMonth: DST Stop DayOfMonth software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStopMonth(uint8 month)

Description: Writes the DST Stop Month software register. Used for absolute date entry. Only generated if DST is enabled.

Parameters: month: DST Stop Month software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStopDayOfWeek(uint8 dayOfWeek)

Description: Writes the DST Stop DayOfWeek software register. Used for relative date entry. Only generated if DST is enabled.

Parameters: dayOfWeek: DST Stop DayOfWeek software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTStopWeek(uint8 week)

Description: Writes the DST Stop Week software register. Used for relative date entry. Only generated if DST is enabled.

Parameters: week: DST Stop Week software register value

Return Value: None

Side Effects: None

void RTC_WriteDSTOffset(uint8 offset)

Description: Writes the DST Offset register. Allows a configurable increment or decrement of time between 0 and 255 minutes. Increment occurs on DST start and decrement on DST stop. Only generated if DST is enabled.

Parameters: offset: DST Offset software register value

Return Value: None

Side Effects: None

void RTC_Init (void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call RTC_Init() because the RTC_Start() API calls this function and is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: All registers will be set to values according to the customizer Configure dialog.

void RTC_Enable(void)

Description: Enables the interrupts, one pulse per second, and interrupt generation on OPPS event.

Parameters: None

Return Value: None

Side Effects: Enables the one pulse per second and central time wheel signals to wake up device from the low-power (Sleep and Alternate Active) modes and leaves them enabled.

Data Structures

RTC_TIME_DATE

This is the data structure that is used to save the current time and date (RTC_currentTimeDate), and Alarm time and date (RTC_alarmCfgTimeDate).

```
typedef struct
{
    uint8 Sec;
    uint8 Min;
    uint8 Hour;
    uint8 DayOfWeek;
    uint8 DayOfMonth;
    uint16 DayOfYear;
    uint8 Month;
    uint16 Year;
} volatile RTC_TIME_DATE;
```

RTC_DSTIME

This is the data structure that is used to save time and date values for Daylight Savings Time Start and Stop (RTC_dstTimeDateStart and RTC_dstTimeDateStop).

```
typedef struct
{
    uint8 Hour;
    uint8 DayOfWeek;
    uint8 Week;
    uint8 DayOfMonth;
    uint8 Month;
} volatile RTC_DSTIME;
```

Constants

There are several constants that define day of week, day in month, and month. When writing code use the constants defined in the header (.h) file.

Global Variables

Variable	Description
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. If reinitialization of the component is required, then the RTC_Init() function can be called before the RTC_Start() or RTC_Enable() function.
RTC_currentTimeDate	The current time and date values are stored in this variable.
RTC_statusDateTime	This variable has following flags: DST, Leap Year, AM/PM, and Active Alarm statuses.
RTC_intervalCfgMask	This variable is used to define what interrupt stub should be executed.
RTC_alarmCfgTimeDate	The alarm time and date values are stored in this variable..
RTC_alarmCurStatus	This variable is used to indicate current active alarm: seconds alarm is active, minutes alarm is active, and so on.
RTC_alarmCfgMask	This variable is used to mask alarm events: mask seconds alarm, mask minutes alarm, and so on.
RTC_dstModeType	This variable stores DST mode type's value: '0' – fixed and '1' – relative.
RTC_dstTimeDateStart	The values for the time and date of the DST start.
RTC_dstTimeDateStop	The values for the time and date of the DST stop.
RTC_dstStartStatus	The DST start status.
RTC_dstStopStatus	The DST stop status.
RTC_dstOffsetMin	The DST offset value in minutes.

Macro Callbacks

Macro callbacks allow users to execute code from the API files that are automatically generated by PSoC Creator. Refer to the PSoC Creator Help and *Component Author Guide* for the more details.

In order to add code to the macro callback present in the component's generated source files, perform the following:

- Define a macro to signal the presence of a callback (in *cyapicallbacks.h*). This will "uncomment" the function call from the component's source code.
- Write the function declaration (in *cyapicallbacks.h*). This will make this function visible by all the project files.
- Write the function implementation (in any user file).

Callback Function [1]	Associated Macro	Description
RTC_ISR_EntryCallback	RTC_ISR_ENTRY_CALLBACK	Used at the beginning of the RTC_ISR() interrupt handler to perform additional application-specific actions.
RTC_ISR_ExitCallback	RTC_ISR_EXIT_CALLBACK	Used at the end of the RTC_ISR() interrupt handler to perform additional application-specific actions.
RTC_EverySecondHandler_Callback	RTC_EVERY_SECOND_HANDLER_CALLBACK	Used in the RTC_EverySecondHandler() function to perform additional application-specific actions.
RTC_EveryMinuteHandler_Callback	RTC_EVERY_MINUTE_HANDLER_CALLBACK	Used in the RTC_EveryMinuteHandler() function to perform additional application-specific actions.
RTC_EveryHourHandler_Callback	RTC_EVERY_HOUR_HANDLER_CALLBACK	Used in the RTC_EveryHourHandler() function to perform additional application-specific actions.
RTC_EveryDayHandler_Callback	RTC_EVERY_DAY_HANDLER_CALLBACK	Used in the RTC_EveryDayHandler() function to perform additional application-specific actions.
RTC_EveryWeekHandler_Callback	RTC_EVERY_WEEK_HANDLER_CALLBACK	Used in the RTC_EveryWeekHandler() function to perform additional application-specific actions.
RTC_EveryMonthHandler_Callback	RTC_EVERY_MONTH_HANDLER_CALLBACK	Used in the RTC_EveryMonthHandler() function to perform additional application-specific actions.
RTC_EveryYearHandler_Callback	RTC_EVERY_YEAR_HANDLER_CALLBACK	Used in the RTC_EveryYearHandler() function to perform additional application-specific actions.

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

[1] The callback function name is formed by component function name optionally appended by short explanation and “Callback” suffix.

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.

The RTC component has the following specific deviations:

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. The supported compilers support larger number nesting of control structures.

19.7	A	A function should be used in preference to a function-like macro.	This deviation is caused by the macro statements used for applying a binary mask. The deviation can be avoided if the macro statements are replaced by functions, but this will cause lower performance:
			`RTC_LEAP_YEAR(),`
			`RTC_IS_BIT_SET()`

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 the 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.

Configuration	PSoC 3 (Keil_PK51)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default	2719	25	2232	29

Interrupt Service Routines

The RTC component uses a single interrupt that triggers every second. The interrupt handler updates the internal date and time structure, and then calls specific functions at appropriate

intervals based on the setting configured with RTC_WriteIntervalMask() function. The following functions are called if the corresponding bit is set in the Interval Mask Register:

- Every Second handler – RTC_EverySecondHandler()
- Every Minute handler – RTC_EveryMinuteHandler()
- Every Hour handler – RTC_EveryHourHandler()
- Every Day handler – RTC_EveryDayHandler()
- Every Week handler – RTC_EveryWeekHandler()
- Every Month handler – RTC_EveryMonthHandler()
- Every Year handler – RTC_EveryYearHandler()

Stub routines for these functions are provided in which you can add your own code. The routine stubs are generated in the *RTC_INT.c* file the first time the project is built. Your code must be added between the provided comment tags as follows:

```
static void RTC_EverySecondHandler( void )
{
    /*  Place your every second handler code here. */
    /* `#START EVERY_SECOND_HANDLER_CODE` */

    /* `#END` */
}
```

All interrupt status bits of the Power Manager Interrupt Status register are cleared in the interrupt handler. If an interrupt is generated at the same time as a clear, the bit remains set (which causes another interrupt).

Obsolete Definitions

Definitions Removed from the Component	Applicable definitions
RTC_IsLeapYear	RTC_LEAP_YEAR
RTC_Dst	RTC_DSTIME
RTC_TimeDate	RTC_TIME_DATE
RTC_CurTimeDate	RTC_currentTimeDate
RTC_AlarmTimeDate	RTC_alarmCfgTimeDate
RTC_DstMode	RTC_dstModeType
RTC_DstStartTimeDate	RTC_dstTimeDateStart
RTC_DstStopTimeDate	RTC_dstTimeDateStop
RTC_DstOffset	RTC_dstOffsetMin
RTC_DstStatusStart	RTC_dstStartStatus
RTC_DstStatusStop	RTC_dstStopStatus

Definitions Removed from the Component	Applicable definitions
RTC_AlarmMask	RTC_alarmCfgMask
RTC_AlarmStatus	RTC_alarmCurStatus
RTC_IntervalMask	RTC_intervalCfgMask
RTC_Status	RTC_statusDateTime
RTC_Dim	RTC_daysInMonths
RTC_Seq	RTC_monthTemplate

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 from the Counter component. The following variables are provided:

- Sec – Seconds: 0 to 59
- Min – Minutes: 0 to 59
- Hour – Hours (24 format only): 0 to 23
- DayOfMonth – Day of month: 1 to 31
- DayOfWeek – Day of week: 1 to 7. The number depends on StartOfWeek parameter settings. If **Start of week** is set to **Sunday** then: 1 – Sunday, 2 – Monday…,7 – Saturday
- DayOfYear – Day of year: 1 to 366
- 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
- DayOfYear
- LY
- AM_PM

- DST

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.

Periodic Interrupts

Interrupt stubs (locations for user code in separate functions) are provided that can run every second, minute, hour, day, week, month, and year. If code is present in the stub it will be run at the appropriate interval.

Daylight Savings Time

To enable the Daylight Savings Time feature, select the check box on the Configure dialog (see the Component Parameters section of this datasheet). Daylight Savings Time 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. It is called at the end of the first hour after the RTC_Start() function is called, and it sets a conversion flag in the RTC_Start() function itself that indicates the conversion is done. The next conversion will be in the next year.

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. The number depends on StartOfWeek parameter settings. If **Start of week** is set to **Sunday** then: 1 – Sunday, 2 – Monday, … ,7 – Saturday (relative)
- Week – Week in month: 1 to 5 (relative)
- DayOfMonth – Day of month: 1 to 31 (fixed)
- Month – Month: 1 to 12 (fixed and relative)

Registers

Status Register

The status register is a read-only register that contains various RTC 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.
- **RTC_STATUS_AA** – Status of alarm active (that is, the alarm bit). This bit is high when current time and date match alarm time and date. After the status is read this bit goes low.

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. The alarm second register is written with the RTC_WriteAlarmSecond() function call and read with RTC_ReadAlarmSecond().
- **RTC_ALARM_MIN_MASK** – The minute alarm mask allows you to match the alarm minute register with the current minute register. The alarm minute register is written with the RTC_WriteAlarmMinute() function call and read with the RTC_ReadAlarmMinute().
- **RTC_ALARM_HOUR_MASK** – The hour alarm mask allows you to match the alarm hour register with the current hour register. The alarm hour register is written with the RTC_WriteAlarmHour() function call and read with the RTC_ReadAlarmHour().
- **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. The alarm day of week register is written with the RTC_WriteAlarmDayOfWeek() function call and read with the RTC_ReadAlarmDayOfWeek().

- **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. The alarm day of month register is written with the RTC_WriteAlarmDayOfMonth() function call and read with the RTC_ReadAlarmDayOfMonth().
- **RTC_ALARM_DAYOFYEAR_MASK** – The day of year alarm mask allows you to match the alarm day of year register with the current day of year register. The alarm day of year register is written with the RTC_WriteAlarmDayOfYear() function call and read with the RTC_ReadAlarmDayOfYear().
- **RTC_ALARM_MONTH_MASK** – The month alarm mask allows you to match the alarm month register with the current month register. The alarm month register is written with the RTC_WriteAlarmMonth() function call and read with the RTC_ReadAlarmMonth().
- **RTC_ALARM_YEAR_MASK** – The year alarm mask allows you to match the alarm year register with the current year register. The alarm year register is written with the RTC_WriteAlarmYear() function call and read with the RTC_ReadAlarmYear().

Interval Mask Register

The interval mask register is a write-only register that allows you to control handling of interrupt stubs of the RTC component. The interrupt stubs are provided for every second, minute, hour, day, week, month, and year. To enable interrupt stub execution, set the appropriate bit in this register. This register is written with the RTC_WriteIntervalMask() function call. When writing the interval mask register, you must use the bit-field definitions as defined in the header (.h) file. The definitions for the interval mask register are as follows:

- **RTC_INTERVAL_SEC_MASK** – The second interval mask allows handling an interrupt stub every second.
- **RTC_INTERVAL_MIN_MASK** – The minute interval mask allows handling an interrupt stub every minute.
- **RTC_INTERVAL_HOUR_MASK** – The hour interval mask allows handling an interrupt stub every hour.
- **RTC_INTERVAL_DAY_MASK** – The day interval mask allows handling an interrupt stub every day.
- **RTC_INTERVAL_WEEK_MASK** – The week interval mask allows handling an interrupt stub every week.
- **RTC_INTERVAL_MONTH_MASK** – The month interval mask allows handling an interrupt stub every month.

- **RTC_INTERVAL_YEAR_MASK** – The year interval mask allows handling an interrupt stub every year.

DST Mode Register

The DST mode register is a write-only register that allows you to set the Daylight Savings Time mode and enable DST operation. This register is written with the RTC_WriteDSTMode() function call. When writing the DST mode register you must use the bit-field definition as defined in the header (.h) file. The definitions for the DST mode register are as follows:

- **RTC_DST_ENABLE** – The enable bit controls enabling the daylight savings time functionality.
- **RTC_DST_FIXDATE** – Defines the fixed format of the times and dates (start and stop) for daylight savings time functionality. For example, fixed date is 24 March.
- **RTC_DST_RELDATE** – Defines the relative format of the times and dates (start and stop) for daylight savings time functionality. For example, relative date is second Sunday in May.

Conditional Compilation Information

The RTC API requires one conditional compile definition to handle daylight savings time functionality. The DST 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_DST_FUNC_ENABLE** – The daylight savings time functionality enable define is assigned to be equal to the DstEnable value (from the Configure dialog) at build time. It is used throughout the API to compile data saving time functions.

Resources

The RTC component uses the one-pulse-per-second Interrupt from Power Management.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.0.b	Minor datasheet edit.	
2.0.a	Datasheet update.	Added Macro Callbacks section.
2.0	Updated MISRA Compliance section.	The component has specific deviations described.
	Made the following functions static: RTC_EverySecondHandler(), RTC_EveryMinuteHandler(), RTC_EveryHourHandler(), RTC_EveryDayHandler(), RTC_EveryWeekHandler(), RTC_EveryMonthHandler(), RTC_EveryYearHandler().	These functions are not API functions. They are just Subroutines that are called from RTC ISR.
	Updated implementation of RTC_SET_ALARM() macros. According to new implementation the semicolon is required after macros call.	Previously RTC_SET_ALARM() macros was implemented through if() {} construction and using of semicolon after macros call was optional. The macros implementation was updated according to MIRSA requirements with using do {} while () construction and semicolon is required after macros call.
	Add handling of February 29 in case of leap year to DST sub-routine in RTC ISR handler.	Issue with switching to February 29 of leap year when DST is triggered.
	Added an Obsolete Definitions table.	Numerous definitions were replaced, as described in the table.
1.80	Updated internal Interrupt component.	
	Added MISRA Compliance section.	The component was not verified for MISRA compliance.
1.70	Added PSoC 5LP device support.	
	Added all RTC APIs with CYREENTRANT keyword when they included in .cyre file.	Not all APIs are truly reentrant. Comments in the component API source files indicate which functions are candidates. This change is required to eliminate compiler warnings for functions that are not reentrant used in a safe way: protected from concurrent calls by flags or Critical Sections.
	Minor datasheet edits.	Improve readability.
1.60.a	Updated the description of the RTC_WriteIntervalMask() function. Updated the “Interrupt Service Routines” section to make it clear how interval handlers are configured.	Interrupt handlers usage description update.

Version	Description of Changes	Reason for Changes / Impact
1.60	Fixed issues when global variables used in both code and ISR could potentially be optimized out by compiler.	Prevents optimization issues that could lead to unexpected result.
	The merger region was added to ISR (RTC_INT.c file).	Gives more flexibility to RTC component.
	Updated the code comments for clarification. Fixed a few typos.	Gives correct information on component operation.
	Minor datasheet edits and updates	
1.50.a	The RTC version 1.50 requires cy_boot version 2.x to be used.	The cy_boot component should be updated up to version 2.x for RTC version 1.50 could be used.
	The “Sample Firmware Source Code” section was updated to note that datasheet example project has been added.	The datasheet example project was added to the PSoC Creator.
	RTC_Enable() function description was added.	Every function that is exposed to user should be described in the datasheet.
	Added reference for the parameter descriptions of the following functions: RTC_WriteAlarmMask(), RTC_WriteIntervalMask(), and RTC_ReadStatus().	Clarification comments on the functions’ usage.
	Added Keil reentrancy support in version 1.50.	PSoC 3 with the Keil compiler supports the capability for functions to be called from multiple flows of control.
1.50	Changed the API flow - added the RTC_Init() function.	To comply with corporate standard and provide an API to initialize or restore the component without starting it.
	Renamed some global variables to comply with coding standard. Updated function headers to describe functions global parameters usage and returns.	Comply with coding rules. No impacts - macros were created for backward compatibility.
	Updated the datasheet.	Added "Component Changes" section. Added empty "Side Effects" section to the "Application Programming Interface" section.
	The RTCIsLeapYear(uint16 year) function was replaced by RTC_LEAP_YEAR macros. The RTC_SET_ALARM and RTC_IS_BIT_SET were created. The RTC_IS_BIT_SET macro used in a few more conditional expressions.	Fixed a bad practice to call functions within ISR, they were moved out or replaced by macros. One-line functions should be replaced by macros. The code, which is used in a few places, should be replaced by macros.

© Cypress Semiconductor Corporation, 2013-2016. 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, 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.