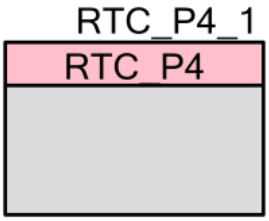
PSoC 4 Real-Time Clock (RTC_P4)

1.0



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 WDT available in the device as its input clock source or have a user configured clock source like SysTick, TCPWM etc. If you choose to use WDT as the input clock source in the "Low Frequency Clocks" configuration window in the CYDWR page, the component derives the input clock period from the WDT frequency configured. If you choose to use other sources, then you need to set the period of the input clock source manually. It should be noted that 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 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.

Daylight savings time 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 also 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 also 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.

Refer also to the Real-Time Clock (RTC) in PSoC® 4 – KBA96934 document located on the cypress.com website. You can use the Cypress Document Manager tool to help locate it.

When to Use an RTC_P4 Component

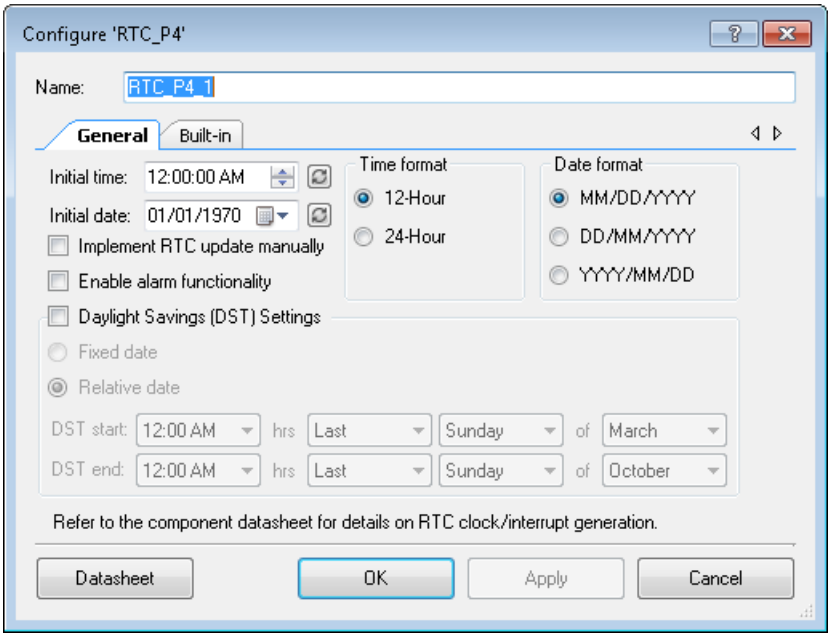
Use the RTC_P4 component when the system requires the current time or date. You can also use the RTC_P4 when you do not need the current time and date but you need accurate timing of events with one-second resolution.

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.



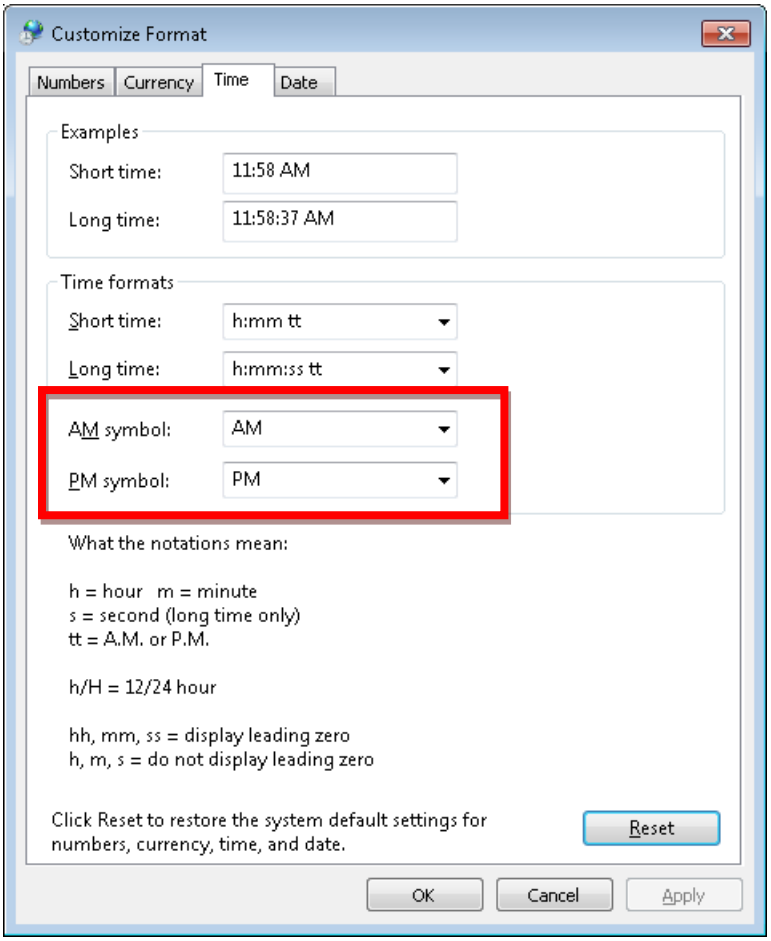
The RTC_P4 component contains the following options:

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 the "PM-symbol" should be properly configured in the "Customize Format" dialog in Windows. See the following figure.



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 checkbox

This parameter is used to map the RTC time update API automatically during RTC start to one of the WDTs 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.

Enable Alarm Functionality checkbox

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 check box

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, 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.

Functions

Function	Description
RTC_Start()	Starts the RTC_P4 component by initializing all the parameters and registers
RTC_Stop()	Stops RTC_P4 component operation
RTC_SetDateAndTime()	Sets time and date values
RTC_GetDateAndTime()	Returns the current time and date
RTC_GetTime()	Returns the current time value
RTC_GetDate()	Returns the current date value
RTC_SetUnixTime()	Sets the time in Unix/Epoch time format
RTC_GetUnixTime()	Returns the time in Unix/Epoch time format
RTC_SetPeriod()	Sets the RTC update period in number of ticks along with reference number of ticks for 1 second period;
RTC_GetPeriod()	Gets the RTC time update period in number of ticks;
RTC_GetRefOneSec(void)	Gets the reference number of ticks taken by the RTC clock source for one second
RTC_SetAlarmDateAndTime()	Sets the alarm time and date values

Function	Description
RTC_GetAlarmDateAndTime()	Returns the set alarm time, date and day of week values
RTC_SetAlarmMask()	Writes the Alarm Mask software register with one bit per time/date entry
RTC_GetAlarmMask()	Reads the Alarm Mask software register with one bit per time/date entry.
RTC_ReadStatus()	Reads the Status software register, which has flags for DST (DST), Leap Year (LY), AM/PM (AM_PM).
RTC_GetAlarmStatus()	Returns the alarm status.
RTC_ClearAlarmStatus()	Clears the alarm status.
RTC_SetDSTStartTime()	Writes the DST Start time software register.
RTC_SetDSTStopTime()	Writes the DST Stop time software register.
RTC_ConvertBCDToDec()	Converts a 4-byte BCD number into a 4-byte hexadecimal number.
RTC_ConvertDecToBCD()	Converts a 4-byte hexadecimal into a 4-byte BCD number.
RTC_Init()	Initializes and restores default configuration provided with the customizer.
RTC_Update()	Does the one-step update to the RTC time and date registers and performs alarm check if enabled.
RTC_SetAlarmHandler()	Sets the function to be called when alarm triggers. Defined only when alarm functionality is enabled in the customizer.
RTC_LeapYear()	Checks if a year is leap or not.
RTC_IsBitSet()	Checks if a bit is set in the value passed.
RTC_GetSecond()	Gets the seconds value from the time value passed.
RTC_GetMinutes()	Gets the minutes value from the time value passed.
RTC_GetHours()	Gets the hour value from the time value passed.
RTC_GetAmPm()	Gets the AM/PM status from the time value passed.
RTC_GetDay()	Gets the day value from the date value passed.
RTC_GetMonth()	Gets the Month value from the date value passed
RTC_GetYear()	Gets the Year value from the date value passed.
RTC_ConstructTime()	Returns the time in format used in the APIs from individual elements passed (hour, min, sec etc).
RTC_ConstructDate()	Returns the date in format used in the APIs from individual elements passed (day. Month and year).

void RTC_Start(void)

Description: 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 is selected as a source in the clocks configuration window (low frequency clocks tab), attaches the RTC_Update API to the corresponding WDT's ISR callback.

Parameters: None

Return Value: None

Side Effects: None

void RTC_Stop(void)

Description: Stops the time and date updates.

Parameters: None

Return Value: None

Side Effects: None

void RTC_SetDateAndTime(uint32 time, uint32 date)

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

Parameters: time: Time value in "HH:MM:SS" format.
date: Date value in format selected in customizer.

Return Value: None

Side Effects: None

void RTC_SetUnixTime(uint64 unixTime)

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

Parameters: time: Time value in Unix time/Epoch time format.

Return Value: None

Side Effects: None

uint64 RTC_GetUnixTime(void)

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

Parameters: None

Return Value: time: Time value in Unix time/Epoch time format.

Side Effects: None

void RTC_SetPeriod(int32 ticks, int32 refOneSecTicks)

Description: Sets the RTC time update API period. You need to pass the period as 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 a RTC period of 100 ms, the 'ticks' value will be 3200 and 'refOneSecTicks' value will be 32000. This value is used to increment the time everytime the RTC_Update() API is called.

Parameters: ticks: Clock period in number of ticks.

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

Return Value: None

Side Effects: None

uint32 RTC_GetPeriod(void)

Description: Gets the RTC time update API period.

Parameters: period: Clock period in number of ticks.

Return Value: None

Side Effects: None

uint32 RTC_GetRefOneSec(void)

Description: Gets the reference number of ticks taken by the RTC clock source for one second.

Parameters: period: Reference number of ticks taken by the RTC clock source for one second.

Return Value: None

Side Effects: None

void RTC_GetDateAndTime(RTC_DATE_TIME* dateTime)

Description: Reads the current time and date.

Parameters: dateTime: Pointer to the RTC_date_time structure in which Time and Date is returned.

Return Value: None

Side Effects: None

uint32 RTC_GetTime (void)

Description: Reads the current time.

Parameters: None

Return Value: Time: Value in the format selected (12/24 hr); The time value is available in BCD format.

Side Effects: 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)

Description: Reads the current date.

Parameters: None

Return Value: Value of date in the selected format; The date value is available in BCD format.

Side Effects: 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)

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

Parameters: alarmTime: Pointer to the RTC_date_time global structure where new values of alarm time and date are stored.

Return Value: None

Side Effects: Invalid time entries will be written with "00:00:00" for 24-hour and "AM 12:00:00" for 12-hour. Invalid date entries will be written with a date equivalent to 01-JAN-2000

void RTC_GetAlarmDateAndTime(RTC_date_time* alarmTimeDate)

Description: Reads the current alarm time and date.

Parameters: alarmTimeDate: Pointer to the RTC_date_time structure in which alarm date and time are returned.

Return Value: None

Side Effects: None

void RTC_SetAlarmMask(uint32 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. Generated only if alarm functionality is enabled.

Parameters: mask: Alarm Mask software register value.

Parameter Value (Bit MASK)	Description
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.

Return Value: None

Side Effects: None

uint32 RTC_GetAlarmMask(void)

Description: Reads the Alarm Mask software register. Generated only if alarm functionality is enabled.

Parameters: None

Return Value: Alarm mask value with each bit representing the status of the alarm time/date match enable

Parameter Value (Bit MASK)	Description
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.

Side Effects: None

uint32 RTC_ReadStatus(void)

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

Parameters: None

Return Value: The values shown below shall be OR'ed and returned if more than one status bits are set.

Parameter Value (Bit MASK)	Description
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.

Side Effects: Reading the status without sync with Date and Time read might cause an error due to roll-over at AM/PM, end of year, end of day; RTC_GetDateAndTime() API can be used to obtain the status and the status member of the returned structure can be checked with the masks.

uint32 RTC_GetAlarmStatus(void)

Description: Returns the alarm status of the RTC.

Parameters: None

Return Value: Alarm Status:

Parameter Value	Description
0 - 1	Alarm active status. This bit is high when current time and date match alarm time and date.

Side Effects: None

void RTC_ClearAlarmStatus(void)

Description: Clears the alarm status of the RTC.

Parameters: None

Return Value: None

Side Effects: Alarm active (AA) flag clear 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 will trigger once every 20th minute of every hour.

void RTC_SetDSTStartTime(const RTC_dst_time* dstStartTime, RTC_dst_dateType_enum type)

Description: Stores the DST Start time. Only generated if DST is enabled. The date passed can be relative or fixed. For a relative date, provide a valid day of week, week of month and month in the dstStartTime structure. For a fixed date, enter a valid day of month and month in the dstStartTime structure. Hour value is optional and if invalid, will be taken as 00 hrs. Invalid entries will not be stored and DST start date will retain previous value or no value at all.

Parameters: dstStartTime: Pointer to the RTC_DST_TIME structure.

type: Defined the DST time format.

Parameter Value	Description
DST_DATE_RELATIVE - 0	DST start time is relative
DST_DATE_FIXED - 1	DST start time is fixed

Return Value: None

Side Effects: None

void RTC_SetDSTStopTime(const RTC_dst_time* dstStopTime, RTC_dst_dateType_enum type)

Description: Stores the DST Stop time. Only generated if DST is enabled. The date passed can be relative or fixed. For a relative date, provide a valid day of week, week of month and month in the dstStopTime structure. For a fixed date, enter valid day of month and month in the dstSoptTime structure. Hour value is optional and if invalid, will be taken as 00 hrs. Invalid entries will not be stored and DST start date will retain previous value or no value at all.

Parameters: dstStopTime: Pointer to the RTC_DST_TIME structure.

type: Defined the DST time format.

Parameter Value	Description
DST_DATE_RELATIVE - 0	DST start time is relative
DST_DATE_FIXED - 1	DST start time is fixed

Return Value: None

Side Effects: None

uint32 RTC_ConvertBCDToDec(uint32 bcdNum)

Description: Converts a 4-byte BCD number into a 4-byte hexadecimal number; each byte is converted individually and returned as individual bytes in the 32-bit variable.

Parameters: bcdNum: 4-byte BCD number.

Return Value: Returns the 4-byte decimal number.

Side Effects: None

uint32 RTC_ConvertDecToBCD(uint32 decNum)

Description: Converts a 4-byte hexadecimal number into a 4-byte BCD number; each byte is converted individually and returned as individual bytes in the 32-bit variable.

Parameters: decNum: 4-byte decimal number.

Return Value: Returns the 4-byte BCD number.

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. Default date value if not set by user before this function call, will be 12:00:00 AM January 1, 2000

void RTC_Update(void)

Description: This API updates the time registers (increments the time/date registers by input clock period 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) and performs alarm/DST check.

If 'Implement RTC update manually' option in the customizer is unchecked and one of the WDT is selected as the output of 'RTC_sel Mux' of Low frequency clocks tab (also 'Implementation by IDE' option selected in 'Timer (WDT) ISR' panel), then the API is automatically mapped to the WDT's callback slot and period.

If 'Implement RTC update manually' option is checked, or 'None' option is selected in 'RTC_sel Mux', then it is your responsibility to call this API from the appropriate clock ISR to use as the RTC's input and also set the period of the RTC through RTC_SetPeriod() API.

Parameters: None

Return Value: None

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

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

Description: This API sets the function to be called when the alarm goes off / triggers.
The API is generated only if alarm functionality is enabled in the Configure dialog.

Parameters: Callback function address.

Return Value: Previous Callback function address.

Side Effects: None

static CY_INLINE uint32 RTC_LeapYear(uint32 year)

Description: Checks if a year is leap or not.

Parameters: year: Year to be checked whether leap year or not.

Return Value: 0-Year is not leap year.
1-Year is a leap year.

Side Effects: None

static CY_INLINE uint32 RTC_IsBitSet(uint32 var, uint32 bitPos)

Description: Checks if a bit is set in the value passed.

Parameters: value: The value in which the bit(s) need to checked.

Return Value: 0-If appropriate bit is 0.
1-If appropriate bit is 1.

Side Effects: None

static CY_INLINE uint32 RTC_GetSecond(uint32 inputTime)

Description: Gets the seconds value from the time value passed.

Parameters: inputTime: Time value in "HH:MM:SS" format.

Return Value: Previous Callback function address.

Side Effects: Returns the seconds value ("SS" byte of the time value).

static CY_INLINE uint32 RTC_GetMinutes(uint32 inputTime)

Description: Gets the Minutes value from the time value passed.

Parameters: inputTime: Time value in "HH:MM:SS" format.

Return Value: Returns the minutes value ("MM" byte of the time value).

Side Effects: None

inline uint32 RTC_GetHours(uint32 inputTime)

Description: Gets the hours value from the time value passed.

Parameters: time: Time value in "HH:MM:SS" format.

Return Value: Returns the hours value ("HH" byte of the time value without AM/PM status).

Side Effects: None

static CY_INLINE uint32 RTC_GetAmPm(uint32 inputTime)

Description: Gets the AM/PM status from the time value passed.

Parameters: time: Time value in "HH:MM:SS" format.

Return Value: 0-AM
1-PM

Side Effects: None

static CY_INLINE uint32 RTC_GetDay(uint32 date)

Description: Gets the day of month value from the date value passed.

Parameters: date: Date value in the selected format ("MM/DD/YYYY", "DD/MM/YYYY" or "YYYY/MM/DD").

Return Value: Returns the day of month value ("DD" byte of the date value).

Side Effects: None

static CY_INLINE uint32 RTC_GetMonth(uint32 date)

Description: Gets the month value from the date value passed.

Parameters: date: Date value in the selected format ("MM/DD/YYYY", "DD/MM/YYYY" or "YYYY/MM/DD").

Return Value: Returns the month value ("MM" byte of the date value).

Side Effects: None

static CY_INLINE uint32 RTC_GetYear(uint32 date)

Description: Gets the Year value from the date value passed.

Parameters: date: Date value in the selected format ("MM/DD/YYYY", "DD/MM/YYYY" or "YYYY/MM/DD").

Return Value: Returns the year value ("YYYY" bytes of the date value).

Side Effects: None

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

Description: Constructs the uint32 time variable in "HH:MM:SS" format from the values passed.

Parameters: timeFormat: Intended to pass the time format of the constructed time.

0-AM

1-PM

stateAmPm: AM/PM value passed (matters only for 12-hr format).

hour: Hour value to be used for "HH".

min: Minutes value to be used for "MM".

sec: Seconds value to be used for "SS".

Return Value: Returns the time value in "HH:MM:SS" format.

Side Effects: None

static CY_INLINE uint32 RTC_ConstructDate(uint32 month, uint32 day, uint32 year)

Description: Constructs the uint32 year variable in "MMDDYYYY" format from the values passed.

Parameters: month: Month value to be used for "MM".

day: Month value to be used for "MM".

year: 4-digit year value to be used for "YYYY".

Return Value: Returns the date value in "MMDDYYYY" format.

Side Effects: None

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.
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_currentTimeDate	The int64 variable represents the standard Unix time (number of seconds elapsed from January 1, 1970 00:00 hours UTC) in 64-bit
RTC_alarmCfgTimeDate	The last updated time and date values are stored in this structure (update happens in Get time/date APIs).
RTC_alarmCurStatus	The alarm time and date values are stored in this variable.
RTC_alarmCfgMask	This variable is used to indicate current active alarm status per time item used in the alarm; whether seconds alarm is active, minutes 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.
RTC_dstStartTime	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.
RTC_dstStopTime	The values for the time and date of the DST start.
RTC_dstStatus	The values for the time and date of the DST stop.
RTC_initVar	The DST start/stop status.

Data Structures

RTC_DATE_TIME

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
{
    uint32 time;
    uint32 date;
    uint32 dayOfWeek;
    uint32 status;
}RTC_DATE_TIME;
```

RTC_DST_TIME

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
{
    uint32 hour;
    uint32 dayOfWeek;
    uint32 dayOfMonth;
    uint32 weekOfMonth;
    uint32 month;
    uint8  timeFormat;
}RTC_DST_TIME;
```

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.

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	1658	148
RTC with DST	2212	148
RTC with alarm	3052	152
RTC with DST and alarm	3590	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.
- Month – Month: 1 to 12
- Year – Year: 1900 to 2200 (the actual range is 1 to 65536)

The Day Of Week 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:LY
- AM_PM
- DST

Alarm Feature

The alarm feature 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 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.

The DST variables (**RTC_DST_TIME** structure) 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.
- weekOfMonth – Week in month:(1-FIRST, 2-SECOND, 3-THIRD, 4-FOURTH, 5-FIFTH, 6-LAST).
- dayOfMonth – Day of month: (1 to 31).
- month – Month: 1 to 12 (fixed and relative)

Registers

Status Register

The status register (RTC_currentTimeDate.status variable) 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 (RTC_alarmCfgMask variable) 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.

Time register

This register (RTC_currentTimeDate.time variable) 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.

Unix time register

This register (RTC_unixTime variable) contains the time value in the Unix/Epoch time format - the number of seconds elapsed from January 1, 1970 UTC 00:00 hrs.

Date register

This register (RTC_currentTimeDate.date variable) 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.

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.

Resources

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

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.0	Initial component version.	

© Cypress Semiconductor Corporation, 2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.