AT03266: SAM D/R/L/C RTC Calendar (RTC CAL) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the device's Real Time Clock functionality in Calendar operating mode, for the configuration and retrieval of the current time and date as maintained by the RTC module. The following driver API modes are covered by this manual:

- Polled APIs
- Callback APIs

The following peripheral is used by this module:

- RTC (Real Time Clock)

The following devices can use this module:

- Atmel | SMART SAM D20/D21
- Atmel | SMART SAM R21
- Atmel | SMART SAM D09/D10/D11
- Atmel | SMART SAM L21/L22
- Atmel | SMART SAM DA1
- Atmel | SMART SAM C20/C21

The outline of this documentation is as follows:

Table of Contents

1. Software License

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

The RTC module in the SAM devices is a 32-bit counter, with a 10-bit programmable prescaler. Typically, the RTC clock is run continuously, including in the device's low-power sleep modes, to track the current time and date information. The RTC can be used as a source to wake up the system at a scheduled time or periodically using the alarm functions.

In this driver, the RTC is operated in Calendar mode. This allows for an easy integration of a real time clock and calendar into a user application to track the passing of time and/or perform scheduled tasks.

Whilst operating in Calendar mode, the RTC features:

- Time tracking in seconds, minutes, and hours
- 12 or 24 hour mode
- Date tracking in day, month, and year
- Automatic leap year correction

3.1. Driver Feature Macro Definition

Driver Feature Macro	Supported devices
FEATURE_RTC_PERIODIC_INT	SAM L21/L22/C20/C21
FEATURE_RTC_PRESCALER_OFF	SAM L21/L22/C20/C21
FEATURE_RTC_CLOCK_SELECTION	SAM L21/L22/C20/C21
FEATURE_RTC_GENERAL_PURPOSE_REG	SAM L21/L22
FEATURE_RTC_CONTINUOUSLY_UPDATED	SAM D20, SAM D21, SAM R21, SAM D10, SAM D11, SAM DA1
FEATURE_RTC_TAMPER_DETECTION	SAM L22

Note: The specific features are only available in the driver when the selected device supports those features.

3.2. Alarms and Overflow

The RTC has up to four independent hardware alarms that can be configured by the user application. These alarms will be triggered on match with the current clock value, and can be set up to trigger an interrupt, event, or both. The RTC can also be configured to clear the clock value on alarm match, resetting the clock to the original start time.

If the RTC is operated in clock-only mode (i.e. with calendar disabled), the RTC counter value will instead be cleared on overflow once the maximum count value has been reached:

$$COUNT_{MAX} = 2^{32} - 1$$

When the RTC is operated with the calendar enabled and run using a nominal 1Hz input clock frequency, a register overflow will occur after 64 years.

3.3. Periodic Events

The RTC can generate events at periodic intervals, allowing for direct peripheral actions without CPU intervention. The periodic events can be generated on the upper eight bits of the RTC prescaler, and will be generated on the rising edge transition of the specified bit. The resulting periodic frequency can be calculated by the following formula:

$$f_{PERIODIC} = \frac{f_{ASY}}{2^{n+3}}$$

Where

f_{ASY}

refers to the *asynchronous* clock set up in the RTC module configuration. For the RTC to operate correctly in calendar mode, this frequency must be 1KHz, while the RTC's internal prescaler should be set to divide by 1024. The **n** parameter is the event source generator index of the RTC module. If the asynchronous clock is operated at the recommended 1KHz, the formula results in the values shown in Table 3-1 RTC Event Frequencies for Each Prescaler Bit Using a 1KHz Clock on page 7.

Table 3-1. RTC Event Frequencies for Each Prescaler Bit Using a 1KHz Clock

n	Periodic event
7	1Hz
6	2Hz
5	4Hz
4	8Hz
3	16Hz
2	32Hz
1	64Hz
0	128Hz

Note: The connection of events between modules requires the use of the SAM Event System Driver (EVENTS) to route output event of one module to the input event of another. For more information on event routing, refer to the event driver documentation.

3.4. Digital Frequency Correction

The RTC module contains Digital Frequency Correction logic to compensate for inaccurate source clock frequencies which would otherwise result in skewed time measurements. The correction scheme requires that at least two bits in the RTC module prescaler are reserved by the correction logic. As a result of this implementation, frequency correction is only available when the RTC is running from a 1Hz reference clock.

The correction procedure is implemented by subtracting or adding a single cycle from the RTC prescaler every 1024 RTC Generic Clock (GCLK) cycles. The adjustment is applied the specified number of time

(maximum 127) over 976 of these periods. The corresponding correction in parts per million (PPM) will be given by:

$$Correction(PPM) = \frac{VALUE}{999424}10^6$$

The RTC clock will tick faster if provided with a positive correction value, and slower when given a negative correction value.

3.5. RTC Tamper Detect

See RTC Tamper Detect.

4. Special Considerations

4.1. Year Limit

The RTC module has a year range of 63 years from the starting year configured when the module is initialized. Dates outside the start to end year range described below will need software adjustment:

$$[YEAR_{START}, YEAR_{START} + 64]$$

4.2. Clock Setup

4.2.1. SAM D20/D21/R21/D10/D11/DA1 Clock Setup

The RTC is typically clocked by a specialized GCLK generator that has a smaller prescaler than the others. By default the RTC clock is on, selected to use the internal 32KHz Resistor/Capacitor (RC)-oscillator with a prescaler of 32, giving a resulting clock frequency of 1024Hz to the RTC. When the internal RTC prescaler is set to 1024, this yields an end-frequency of 1Hz for correct time keeping operations.

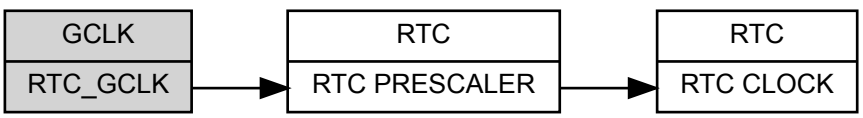
The implementer also has the option to set other end-frequencies. Table 4-1 RTC Output Frequencies from Allowable Input Clocks on page 9 lists the available RTC frequencies for each possible GCLK and RTC input prescaler options.

Table 4-1. RTC Output Frequencies from Allowable Input Clocks

End-frequency	GCLK prescaler	RTC prescaler
32KHz	1	1
1KHz	32	1
1Hz	32	1024

The overall RTC module clocking scheme is shown in Figure 4-1 SAM D20/D21/R21/D10/D11/DA1 Clock Setup on page 9.

Figure 4-1. SAM D20/D21/R21/D10/D11/DA1 Clock Setup



Note: For the calendar to operate correctly, an asynchronous clock of 1Hz should be used.

4.2.2. SAM L21/C20/C21 Clock Setup

The RTC clock can be selected from OSC32K, XOSC32K, or OSCULP32K. A 32KHz or 1KHz oscillator clock frequency is required. This clock must be configured and enabled in the 32KHz oscillator controller before using the RTC.

Table 4-2 RTC Clocks Source on page 10 lists the available RTC clock.

Table 4-2. RTC Clocks Source

RTC clock frequency	Clock source	Description
1.024kHz	ULP1K	1.024kHz from 32KHz internal ULP oscillator
32.768kHz	ULP32K	32.768kHz from 32KHz internal ULP oscillator
1.024kHz	OSC1K	1.024kHz from 32KHz internal oscillator
32.768kHz	OSC32K	32.768kHz from 3KkHz internal oscillator
1.024kHz	XOSC1K	1.024kHz from 32KHz internal oscillator
32.768kHz	XOSC32K	32.768kHz from 32KHz external crystal oscillator

Note: For the calendar to operate correctly, an asynchronous clock of 1Hz should be used.

5. Extra Information

For extra information, see Extra Information for RTC (CAL) Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

For a list of examples related to this driver, see Examples for RTC CAL Driver.

7. API Overview

7.1. Structure Definitions

7.1.1. Struct rtc_calendar_alarm_time

Alarm structure containing time of the alarm and a mask to determine when the alarm will trigger.

Table 7-1. Members

Type	Name	Description
enum rtc_calendar_alarm_mask	mask	Alarm mask to determine on what precision the alarm will match
struct rtc_calendar_time	time	Alarm time

7.1.2. Struct rtc_calendar_config

Configuration structure for the RTC instance. This structure should be initialized using the rtc_calendar_get_config_defaults() before any user configurations are set.

Table 7-2. Members

Type	Name	Description
struct rtc_calendar_alarm_time	alarm[]	Alarm values
bool	clear_on_match	If `true`, clears the clock on alarm match
bool	clock_24h	If `true`, time is represented in 24 hour mode
enum rtc_calendar_prescaler	prescaler	Input clock prescaler for the RTC module
uint16_t	year_init_value	Initial year for counter value 0

7.1.3. Struct rtc_calendar_events

Event flags for the rtc_calendar_enable_events() and rtc_calendar_disable_events().

Table 7-3. Members

Type	Name	Description
bool	generate_event_on_alarm[]	Generate an output event on an alarm channel match against the RTC count
bool	generate_event_on_overflow	Generate an output event on each overflow of the RTC count
bool	generate_event_on_periodic[]	Generate an output event periodically at a binary division of the RTC counter frequency
bool	generate_event_on_tamper	Generate an output event on every tamper input
bool	on_event_to_tamper	Tamper input event and capture the CLOCK value

7.1.4. Struct rtc_calendar_time

Time structure containing the time given by or set to the RTC calendar. The structure uses seven values to give second, minute, hour, PM/AM, day, month, and year. It should be initialized via the rtc_calendar_get_time_defaults() function before use.

Table 7-4. Members

Type	Name	Description
uint8_t	day	Day value, where day 1 is the first day of the month
uint8_t	hour	Hour value
uint8_t	minute	Minute value
uint8_t	month	Month value, where month 1 is January
bool	pm	PM/AM value, `true` for PM, or `false` for AM
uint8_t	second	Second value
uint16_t	year	Year value

7.1.5. Struct rtc_tamper_config

The configuration structure for the RTC tamper. This structure should be initialized using the rtc_tamper_get_config_defaults() before any user configurations are set.

Table 7-5. Members

Type	Name	Description
enum rtc_tamper_active_layer_freq_divider	actl_freq_div	Active layer frequency
bool	bkup_reset_on_tamper	Backup register reset on tamper enable
enum rtc_tamper_debounce_freq_divider	deb_freq_div	Debounce frequency
enum rtc_tamper_debounce_seq	deb_seq	Debounce sequential
bool	dma_tamper_enable	DMA on tamper enable
bool	gp0_enable	General Purpose 0/1 Enable
bool	gp_reset_on_tamper	GP register reset on tamper enable
struct rtc_tamper_input_config	in_cfg[]	Tamper IN configuration

7.1.6. Struct rtc_tamper_input_config

The configuration structure for tamper INn.

Table 7-6. Members

Type	Name	Description
enum rtc_tamper_input_action	action	Tamper input action
bool	debounce_enable	Debounce enable
enum rtc_tamper_level_sel	level	Tamper level select

7.2. Macro Definitions

7.2.1. Macro FEATURE_RTC_PERIODIC_INT

```
#define FEATURE_RTC_PERIODIC_INT
```

Define port features set according to different device familyRTC periodic interval interrupt.

7.2.2. Macro FEATURE_RTC_PRESCALER_OFF

```
#define FEATURE_RTC_PRESCALER_OFF
```

RTC prescaler is off.

7.2.3. Macro FEATURE_RTC_CLOCK_SELECTION

```
#define FEATURE_RTC_CLOCK_SELECTION
```

RTC clock selection.

7.2.4. Macro FEATURE_RTC_GENERAL_PURPOSE_REG

```
#define FEATURE_RTC_GENERAL_PURPOSE_REG
```

General purpose registers.

7.2.5. Macro FEATURE_RTC_TAMPER_DETECTION

```
#define FEATURE_RTC_TAMPER_DETECTION
```

RTC tamper detection.

7.2.6. Macro RTC_TAMPER_DETECT_EVT

```
#define RTC_TAMPER_DETECT_EVT
```

RTC tamper input event detection bitmask.

7.2.7. Macro RTC_TAMPER_DETECT_ID0

```
#define RTC_TAMPER_DETECT_ID0
```

RTC tamper ID0 detection bitmask.

7.2.8. Macro RTC_TAMPER_DETECT_ID1

```
#define RTC_TAMPER_DETECT_ID1
```

RTC tamper ID1 detection bitmask.

7.2.9. Macro RTC_TAMPER_DETECT_ID2

```
#define RTC_TAMPER_DETECT_ID2
```

RTC tamper ID2 detection bitmask.

7.2.10. Macro RTC_TAMPER_DETECT_ID3

```
#define RTC_TAMPER_DETECT_ID3
```

RTC tamper ID3 detection bitmask.

7.2.11. Macro RTC_TAMPER_DETECT_ID4

```
#define RTC_TAMPER_DETECT_ID4
```

RTC tamper ID4 detection bitmask.

7.3. Function Definitions

7.3.1. Configuration and Initialization

7.3.1.1. Function rtc_calendar_get_time_defaults()

Initialize a `time` structure.

```
void rtc_calendar_get_time_defaults(
        struct rtc_calendar_time *const time)
```

This will initialize a given time structure to the time 00:00:00 (hh:mm:ss) and date 2000-01-01 (YYYY-MM-DD).

Table 7-7. Parameters

Data direction	Parameter name	Description
[out]	time	Time structure to initialize

7.3.1.2. Function rtc_calendar_get_config_defaults()

Gets the RTC default settings.

```
void rtc_calendar_get_config_defaults(
        struct rtc_calendar_config *const config)
```

Initializes the configuration structure to the known default values. This function should be called at the start of any RTC initiation.

The default configuration is as follows:

- Input clock divided by a factor of 1024

- Clear on alarm match off
- Continuously sync clock off
- 12 hour calendar
- Start year 2000 (Year 0 in the counter will be year 2000)
- Events off
- Alarms set to January 1. 2000, 00:00:00
- Alarm will match on second, minute, hour, day, month, and year
- Clock read synchronization is enabled for SAM L22

Table 7-8. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to be initialized to default values

7.3.1.3. Function rtc_calendar_reset()

Resets the RTC module.

```
void rtc_calendar_reset(
        struct rtc_module *const module)
```

Resets the RTC module to hardware defaults.

Table 7-9. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct

7.3.1.4. Function rtc_calendar_enable()

Enables the RTC module.

```
void rtc_calendar_enable(
        struct rtc_module *const module)
```

Enables the RTC module once it has been configured, ready for use. Most module configuration parameters cannot be altered while the module is enabled.

Table 7-10. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct

7.3.1.5. Function rtc_calendar_disable()

Disables the RTC module.

```
void rtc_calendar_disable(
        struct rtc_module *const module)
```

Disables the RTC module.

Table 7-11. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct

7.3.1.6. Function rtc_calendar_init()

Initializes the RTC module with given configurations.

```
void rtc_calendar_init(
        struct rtc_module *const module,
        Rtc *const hw,
        const struct rtc_calendar_config *const config)
```

Initializes the module, setting up all given configurations to provide the desired functionality of the RTC.

Table 7-12. Parameters

Data direction	Parameter name	Description
[out]	module	Pointer to the software instance struct
[in]	hw	Pointer to hardware instance
[in]	config	Pointer to the configuration structure

7.3.1.7. Function rtc_calendar_swap_time_mode()

Swaps between 12h and 24h clock mode.

```
void rtc_calendar_swap_time_mode(
        struct rtc_module *const module)
```

Swaps the current RTC time mode:

- If currently in 12h mode, it will swap to 24h
- If currently in 24h mode, it will swap to 12h

Note: This will not change setting in user's configuration structure.

Table 7-13. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct

7.3.1.8. Function rtc_calendar_frequency_correction()

Calibrate for too-slow or too-fast oscillator.

```
enum status_code rtc_calendar_frequency_correction(
        struct rtc_module *const module,
        const int8_t value)
```

When used, the RTC will compensate for an inaccurate oscillator. The RTC module will add or subtract cycles from the RTC prescaler to adjust the frequency in approximately 1 PPM steps. The provided correction value should be between -127 and 127, allowing for a maximum 127 PPM correction in either direction.

If no correction is needed, set value to zero.

Note: Can only be used when the RTC is operated at 1Hz.

Table 7-14. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	value	Between -127 and 127 used for the correction

Returns
Status of the calibration procedure.

Table 7-15. Return Values

Return value	Description
STATUS_OK	If calibration was done correctly
STATUS_ERR_INVALID_ARG	If invalid argument(s) were provided

7.3.2. Time and Alarm Management

7.3.2.1. Function rtc_calendar_time_to_register_value()

Convert time structure to register_value. Retrieves register_value convert by the time structure.

```
uint32_t rtc_calendar_time_to_register_value(
        struct rtc_module *const module,
        const struct rtc_calendar_time *const time)
```

Table 7-16. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	time	Pointer to the time structure

Returns
32-bit value.

7.3.2.2. Function rtc_calendar_register_value_to_time()

Convert register_value to time structure. Retrieves the time structure convert by register_value.

```
void rtc_calendar_register_value_to_time(
        struct rtc_module *const module,
        const uint32_t register_value,
        struct rtc_calendar_time *const time)
```

Table 7-17. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	register_value	The value stored in register
[out]	time	Pointer to the time structure

7.3.2.3. Function rtc_calendar_set_time()

Set the current calendar time to desired time.

```
void rtc_calendar_set_time(
        struct rtc_module *const module,
        const struct rtc_calendar_time *const time)
```

Sets the time provided to the calendar.

Table 7-18. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	time	The time to set in the calendar

7.3.2.4. Function rtc_calendar_get_time()

Get the current calendar value.

```
void rtc_calendar_get_time(
        struct rtc_module *const module,
        struct rtc_calendar_time *const time)
```

Retrieves the current time of the calendar.

Table 7-19. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[out]	time	Pointer to value that will be filled with current time

7.3.2.5. Function rtc_calendar_set_alarm()

Set the alarm time for the specified alarm.

```
enum status_code rtc_calendar_set_alarm(
        struct rtc_module *const module,
        const struct rtc_calendar_alarm_time *const alarm,
        const enum rtc_calendar_alarm alarm_index)
```

Sets the time and mask specified to the requested alarm.

Table 7-20. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	alarm	The alarm struct to set the alarm with
[in]	alarm_index	The index of the alarm to set

Returns
Status of setting alarm.

Table 7-21. Return Values

Return value	Description
STATUS_OK	If alarm was set correctly
STATUS_ERR_INVALID_ARG	If invalid argument(s) were provided

7.3.2.6. Function rtc_calendar_get_alarm()

Get the current alarm time of specified alarm.

```
enum status_code rtc_calendar_get_alarm(
        struct rtc_module *const module,
        struct rtc_calendar_alarm_time *const alarm,
        const enum rtc_calendar_alarm alarm_index)
```

Retrieves the current alarm time for the alarm specified alarm.

Table 7-22. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[out]	alarm	Pointer to the struct that will be filled with alarm time and mask of the specified alarm
[in]	alarm_index	Index of alarm to get alarm time from

Returns
Status of getting alarm.

Table 7-23. Return Values

Return value	Description
STATUS_OK	If alarm was read correctly
STATUS_ERR_INVALID_ARG	If invalid argument(s) were provided

7.3.3. Status Flag Management

7.3.3.1. Function rtc_calendar_is_overflow()

Check if an RTC overflow has occurred.

```
bool rtc_calendar_is_overflow(
        struct rtc_module *const module)
```

Checks the overflow flag in the RTC. The flag is set when there is an overflow in the clock.

Table 7-24. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct

Returns
Overflow state of the RTC module.

Table 7-25. Return Values

Return value	Description
true	If the RTC count value has overflowed
false	If the RTC count value has not overflowed

7.3.3.2. Function rtc_calendar_clear_overflow()

Clears the RTC overflow flag.

```
void rtc_calendar_clear_overflow(
        struct rtc_module *const module)
```

Table 7-26. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct

Clears the RTC module counter overflow flag, so that new overflow conditions can be detected.

7.3.3.3. Function rtc_calendar_is_periodic_interval()

Check if an RTC periodic interval interrupt has occurred.

```
bool rtc_calendar_is_periodic_interval(
        struct rtc_module *const module,
        enum rtc_calendar_periodic_interval n)
```

Checks the periodic interval flag in the RTC.

Table 7-27. Parameters

Data direction	Parameter name	Description
[in, out]	module	RTC hardware module
[in]	n	RTC periodic interval interrupt

Returns
Periodic interval interrupt state of the RTC module.

Table 7-28. Return Values

Return value	Description
true	RTC periodic interval interrupt occur
false	RTC periodic interval interrupt doesn't occur

7.3.3.4. Function rtc_calendar_clear_periodic_interval()

Clears the RTC periodic interval flag.

```
void rtc_calendar_clear_periodic_interval(
        struct rtc_module *const module,
        enum rtc_calendar_periodic_interval n)
```

Clears the RTC module counter periodic interval flag, so that new periodic interval conditions can be detected.

Table 7-29. Parameters

Data direction	Parameter name	Description
[in, out]	module	RTC hardware module
[in]	n	RTC periodic interval interrupt

7.3.3.5. Function rtc_calendar_is_alarm_match()

Check the RTC alarm flag.

```
bool rtc_calendar_is_alarm_match(
        struct rtc_module *const module,
        const enum rtc_calendar_alarm alarm_index)
```

Check if the specified alarm flag is set. The flag is set when there is a compare match between the alarm value and the clock.

Table 7-30. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	alarm_index	Index of the alarm to check

Returns
Match status of the specified alarm.

Table 7-31. Return Values

Return value	Description
true	If the specified alarm has matched the current time
false	If the specified alarm has not matched the current time

7.3.3.6. Function rtc_calendar_clear_alarm_match()

Clears the RTC alarm match flag.

```
enum status_code rtc_calendar_clear_alarm_match(
        struct rtc_module *const module,
        const enum rtc_calendar_alarm alarm_index)
```

Clear the requested alarm match flag, so that future alarm matches can be determined.

Table 7-32. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	alarm_index	The index of the alarm match to clear

Returns
Status of the alarm match clear operation.

Table 7-33. Return Values

Return value	Description
STATUS_OK	If flag was cleared correctly
STATUS_ERR_INVALID_ARG	If invalid argument(s) were provided

7.3.4. Event Management

7.3.4.1. Function rtc_calendar_enable_events()

Enables an RTC event output.

```
void rtc_calendar_enable_events(
        struct rtc_module *const module,
        struct rtc_calendar_events *const events)
```

Enables one or more output events from the RTC module. See rtc_calendar_events for a list of events this module supports.

Note: Events cannot be altered while the module is enabled.

Table 7-34. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	events	Struct containing flags of events to enable

7.3.4.2. Function rtc_calendar_disable_events()

Disables an RTC event output.

```
void rtc_calendar_disable_events(
        struct rtc_module *const module,
        struct rtc_calendar_events *const events)
```

Disabled one or more output events from the RTC module. See rtc_calendar_events for a list of events this module supports.

Note: Events cannot be altered while the module is enabled.

Table 7-35. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	events	Struct containing flags of events to disable

7.3.5. RTC General Purpose Registers

7.3.5.1. Function rtc_write_general_purpose_reg()

Write a value into general purpose register.

```
void rtc_write_general_purpose_reg(
        struct rtc_module *const module,
        const uint8_t index,
        uint32_t value)
```

Table 7-36. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to the software instance struct
[in]	n	General purpose type
[in]	index	General purpose register index (0..3)

7.3.5.2. Function rtc_read_general_purpose_reg()

Read the value from general purpose register.

```
uint32_t rtc_read_general_purpose_reg(
        struct rtc_module *const module,
        const uint8_t index)
```

Table 7-37. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to the software instance struct
[in]	index	General purpose register index (0..3)

Returns
Value of general purpose register.

7.3.6. Callbacks

7.3.6.1. Function rtc_calendar_register_callback()

Registers callback for the specified callback type.

```
enum status_code rtc_calendar_register_callback(
        struct rtc_module *const module,
        rtc_calendar_callback_t callback,
        enum rtc_calendar_callback callback_type)
```

Associates the given callback function with the specified callback type. To enable the callback, the rtc_calendar_enable_callback function must be used.

Table 7-38. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	callback	Pointer to the function desired for the specified callback
[in]	callback_type	Callback type to register

Returns
Status of registering callback.

Table 7-39. Return Values

Return value	Description
STATUS_OK	Registering was done successfully
STATUS_ERR_INVALID_ARG	If trying to register, a callback is not available

7.3.6.2. Function rtc_calendar_unregister_callback()

Unregisters callback for the specified callback type.

```
enum status_code rtc_calendar_unregister_callback(
        struct rtc_module *const module,
        enum rtc_calendar_callback callback_type)
```

When called, the currently registered callback for the given callback type will be removed.

Table 7-40. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	callback_type	Specifies the callback type to unregister

Returns
Status of unregistering callback.

Table 7-41. Return Values

Return value	Description
STATUS_OK	Unregistering was done successfully
STATUS_ERR_INVALID_ARG	If trying to unregister, a callback is not available

7.3.6.3. Function rtc_calendar_enable_callback()

Enables callback.

```
void rtc_calendar_enable_callback(
        struct rtc_module *const module,
        enum rtc_calendar_callback callback_type)
```

Enables the callback specified by the callback_type.

Table 7-42. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	callback_type	Callback type to enable

7.3.6.4. Function rtc_calendar_disable_callback()

Disables callback.

```
void rtc_calendar_disable_callback(
        struct rtc_module *const module,
        enum rtc_calendar_callback callback_type)
```

Disables the callback specified by the callback_type.

Table 7-43. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	callback_type	Callback type to disable

7.3.7. RTC Tamper Detection

7.3.7.1. Function rtc_tamper_get_config_defaults()

Gets the RTC tamper default configurations.

```
void rtc_tamper_get_config_defaults(
        struct rtc_tamper_config *const config)
```

Initializes the configuration structure to default values.

The default configuration is as follows:

- Disable backup register reset on tamper
- Disable GP register reset on tamper
- Active layer clock divided by a factor of 8

- Debounce clock divided by a factor of 8
- Detect edge on INn with synchronous stability debouncing
- Disable DMA on tamper
- Enable GP register
- Disable debouce, detect on falling edge and no action on INn

Table 7-44. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to be initialized to default values.

7.3.7.2. Function rtc_tamper_set_config()

```
enum status_code rtc_tamper_set_config(
        struct rtc_module *const module,
        struct rtc_tamper_config *const tamper_cfg)
```

7.3.7.3. Function rtc_tamper_get_detect_flag()

Retrieves the RTC tamper detection status.

```
uint32_t rtc_tamper_get_detect_flag(
        struct rtc_module *const module)
```

Retrieves the detection status of each input pin and the input event.

Table 7-45. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to the RTC software instance struct

Returns
Bitmask of detection flags.

Table 7-46. Return Values

Return value	Description
RTC_TAMPER_DETECT_ID0	Tamper condition on IN0 has been detected
RTC_TAMPER_DETECT_ID1	Tamper condition on IN1 has been detected
RTC_TAMPER_DETECT_ID2	Tamper condition on IN2 has been detected
RTC_TAMPER_DETECT_ID3	Tamper condition on IN3 has been detected
RTC_TAMPER_DETECT_ID4	Tamper condition on IN4 has been detected
RTC_TAMPER_DETECT_EVT	Tamper input event has been detected

7.3.7.4. Function rtc_tamper_clear_detect_flag()

Clears RTC tamper detection flag.

```
void rtc_tamper_clear_detect_flag(
        struct rtc_module *const module,
        const uint32_t detect_flags)
```

Clears the given detection flag of the module.

Table 7-47. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to the TC software instance struct
[in]	detect_flags	Bitmask of detection flags

7.3.8. Function rtc_tamper_get_stamp()

Get the tamper stamp value.

```
void rtc_tamper_get_stamp(
        struct rtc_module *const module,
        struct rtc_calendar_time *const time)
```

Table 7-48. Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[out]	time	Pointer to value that filled with tamper stamp time

7.4. Enumeration Definitions

7.4.1. Enum rtc_calendar_alarm

Available alarm channels.

Note: Not all alarm channels are available on all devices.

Table 7-49. Members

Enum value	Description
RTC_CALENDAR_ALARM_0	Alarm channel 0
RTC_CALENDAR_ALARM_1	Alarm channel 1
RTC_CALENDAR_ALARM_2	Alarm channel 2
RTC_CALENDAR_ALARM_3	Alarm channel 3

7.4.2. Enum rtc_calendar_alarm_mask

Available mask options for alarms.

Table 7-50. Members

Enum value	Description
RTC_CALENDAR_ALARM_MASK_DISABLED	Alarm disabled
RTC_CALENDAR_ALARM_MASK_SEC	Alarm match on second
RTC_CALENDAR_ALARM_MASK_MIN	Alarm match on second and minute
RTC_CALENDAR_ALARM_MASK_HOUR	Alarm match on second, minute, and hour
RTC_CALENDAR_ALARM_MASK_DAY	Alarm match on second, minute, hour, and day
RTC_CALENDAR_ALARM_MASK_MONTH	Alarm match on second, minute, hour, day, and month
RTC_CALENDAR_ALARM_MASK_YEAR	Alarm match on second, minute, hour, day, month, and year

7.4.3. Enum rtc_calendar_callback

The available callback types for the RTC calendar module.

Table 7-51. Members

Enum value	Description
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_0	Callback for Periodic Interval 0 Interrupt
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_1	Callback for Periodic Interval 1 Interrupt
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_2	Callback for Periodic Interval 2 Interrupt
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_3	Callback for Periodic Interval 3 Interrupt
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_4	Callback for Periodic Interval 4 Interrupt
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_5	Callback for Periodic Interval 5 Interrupt
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_6	Callback for Periodic Interval 6 Interrupt
RTC_CALENDAR_CALLBACK_PERIODIC_INTERVAL_7	Callback for Periodic Interval 7 Interrupt
RTC_CALENDAR_CALLBACK_ALARM_0	Callback for alarm 0
RTC_CALENDAR_CALLBACK_ALARM_1	Callback for alarm 1
RTC_CALENDAR_CALLBACK_ALARM_2	Callback for alarm 2
RTC_CALENDAR_CALLBACK_ALARM_3	Callback for alarm 3
RTC_CALENDAR_CALLBACK_TAMPER	Callback for tamper
RTC_CALENDAR_CALLBACK_OVERFLOW	Callback for overflow

7.4.4. Enum rtc_calendar_periodic_interval

Table 7-52. Members

Enum value	Description
RTC_CALENDAR_PERIODIC_INTERVAL_0	Periodic interval 0
RTC_CALENDAR_PERIODIC_INTERVAL_1	Periodic interval 1
RTC_CALENDAR_PERIODIC_INTERVAL_2	Periodic interval 2
RTC_CALENDAR_PERIODIC_INTERVAL_3	Periodic interval 3
RTC_CALENDAR_PERIODIC_INTERVAL_4	Periodic interval 4
RTC_CALENDAR_PERIODIC_INTERVAL_5	Periodic interval 5
RTC_CALENDAR_PERIODIC_INTERVAL_6	Periodic interval 6
RTC_CALENDAR_PERIODIC_INTERVAL_7	Periodic interval 7

7.4.5. Enum rtc_calendar_prescaler

The available input clock prescaler values for the RTC calendar module.

Table 7-53. Members

Enum value	Description
RTC_CALENDAR_PRESCALER_OFF	RTC prescaler is off, and the input clock frequency is prescaled by a factor of 1
RTC_CALENDAR_PRESCALER_DIV_1	RTC input clock frequency is prescaled by a factor of 1
RTC_CALENDAR_PRESCALER_DIV_2	RTC input clock frequency is prescaled by a factor of 2
RTC_CALENDAR_PRESCALER_DIV_4	RTC input clock frequency is prescaled by a factor of 4
RTC_CALENDAR_PRESCALER_DIV_8	RTC input clock frequency is prescaled by a factor of 8
RTC_CALENDAR_PRESCALER_DIV_16	RTC input clock frequency is prescaled by a factor of 16
RTC_CALENDAR_PRESCALER_DIV_32	RTC input clock frequency is prescaled by a factor of 32
RTC_CALENDAR_PRESCALER_DIV_64	RTC input clock frequency is prescaled by a factor of 64
RTC_CALENDAR_PRESCALER_DIV_128	RTC input clock frequency is prescaled by a factor of 128
RTC_CALENDAR_PRESCALER_DIV_256	RTC input clock frequency is prescaled by a factor of 256
RTC_CALENDAR_PRESCALER_DIV_512	RTC input clock frequency is prescaled by a factor of 512
RTC_CALENDAR_PRESCALER_DIV_1024	RTC input clock frequency is prescaled by a factor of 1024

7.4.6. Enum rtc_clock_sel

Table 7-54. Members

Enum value	Description
RTC_CLOCK_SELECTION_ULP1K	1.024kHz from 32KHz internal ULP oscillator
RTC_CLOCK_SELECTION_ULP32K	32.768kHz from 32KHz internal ULP oscillator
RTC_CLOCK_SELECTION_OSC1K	1.024kHz from 32KHz internal oscillator
RTC_CLOCK_SELECTION_OSC32K	32.768kHz from 32KHz internal oscillator
RTC_CLOCK_SELECTION_XOSC1K	1.024kHz from 32KHz internal oscillator
RTC_CLOCK_SELECTION_XOSC32K	32.768kHz from 32.768kHz external crystal oscillator

7.4.7. Enum rtc_tamper_active_layer_freq_divider

The available prescaler factor for the RTC clock output used during active layer protection.

Table 7-55. Members

Enum value	Description
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_2	RTC active layer frequency is prescaled by a factor of 2
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_4	RTC active layer frequency is prescaled by a factor of 4
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_8	RTC active layer frequency is prescaled by a factor of 8
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_16	RTC active layer frequency is prescaled by a factor of 16
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_32	RTC active layer frequency is prescaled by a factor of 32
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_64	RTC active layer frequency is prescaled by a factor of 64
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_128	RTC active layer frequency is prescaled by a factor of 128
RTC_TAMPER_ACTIVE_LAYER_FREQ_DIV_256	RTC active layer frequency is prescaled by a factor of 256

7.4.8. Enum rtc_tamper_debounce_freq_divider

The available prescaler factor for the input debouncers.

Table 7-56. Members

Enum value	Description
RTC_TAMPER_DEBOUNCE_FREQ_DIV_2	RTC debounce frequency is prescaled by a factor of 2
RTC_TAMPER_DEBOUNCE_FREQ_DIV_4	RTC debounce frequency is prescaled by a factor of 4
RTC_TAMPER_DEBOUNCE_FREQ_DIV_8	RTC debounce frequency is prescaled by a factor of 8
RTC_TAMPER_DEBOUNCE_FREQ_DIV_16	RTC debounce frequency is prescaled by a factor of 16
RTC_TAMPER_DEBOUNCE_FREQ_DIV_32	RTC debounce frequency is prescaled by a factor of 32
RTC_TAMPER_DEBOUNCE_FREQ_DIV_64	RTC debounce frequency is prescaled by a factor of 64
RTC_TAMPER_DEBOUNCE_FREQ_DIV_128	RTC debounce frequency is prescaled by a factor of 128
RTC_TAMPER_DEBOUNCE_FREQ_DIV_256	RTC debounce frequency is prescaled by a factor of 256

7.4.9. Enum rtc_tamper_debounce_seq

The available sequential for tamper debounce.

Table 7-57. Members

Enum value	Description
RTC_TAMPER_DEBOUNCE_SYNC	Tamper input detect edge with synchronous stability debounce
RTC_TAMPER_DEBOUNCE_ASYNC	Tamper input detect edge with asynchronous stability debounce
RTC_TAMPER_DEBOUNCE_MAJORITY	Tamper input detect edge with majority debounce

7.4.10. Enum rtc_tamper_input_action

The available action taken by the tamper input.

Table 7-58. Members

Enum value	Description
RTC_TAMPER_INPUT_ACTION_OFF	RTC tamper input action is disabled
RTC_TAMPER_INPUT_ACTION_WAKE	RTC tamper input action is wake and set tamper flag
RTC_TAMPER_INPUT_ACTION_CAPTURE	RTC tamper input action is capture timestamp and set tamper flag
RTC_TAMPER_INPUT_ACTION_ACTL	RTC tamper input action is compare IN to OUT, when a mismatch occurs, capture timestamp and set tamper flag

7.4.11. Enum rtc_tamper_level_sel

The available edge condition for tamper INn level select.

Table 7-59. Members

Enum value	Description
RTC_TAMPER_LEVEL_FALLING	A falling edge condition will be detected on Tamper input
RTC_TAMPER_LEVEL_RISING	A rising edge condition will be detected on Tamper input

8. RTC Tamper Detect

The RTC provides several selectable polarity external inputs (INn) that can be used for tamper detection. When detect, tamper inputs support the four actions:

- Off
- Wake
- Capture
- Active layer protection

Note: The Active Layer Protection is a means of detecting broken traces on the PCB provided by RTC. In this mode an RTC output signal is routed over critical components on the board and fed back to one of the RTC inputs. The input and output signals are compared and a tamper condition is detected when they do not match.

Separate debouncers are embedded for each external input. The detection time depends on whether the debouncer operates synchronously or asynchronously, and whether majority detection is enabled or not. For details, refer to the section "Tamper Detection" of datasheet.

9. Extra Information for RTC (CAL) Driver

9.1. Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Description
RTC	Real Time Counter
PPM	Part Per Million
RC	Resistor/Capacitor

9.2. Dependencies

This driver has the following dependencies:

- None

9.3. Errata

There are no errata related to this driver.

9.4. Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Added support for RTC tamper feature
Added driver instance parameter to all API function calls, except get_config_defaults
Updated initialization function to also enable the digital interface clock to the module if it is disabled
Initial release

10. Examples for RTC CAL Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM RTC Calendar (RTC CAL) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for RTC (CAL) - Basic
- Quick Start Guide for RTC (CAL) - Callback

10.1. Quick Start Guide for RTC (CAL) - Basic

In this use case, the RTC is set up in calendar mode. The time is set and also an alarm is set to show a general use of the RTC in calendar mode. Also the clock is swapped from 24h to 12h mode after initialization. The board LED will be toggled once the current time matches the set time.

10.1.1. Prerequisites

The Generic Clock Generator for the RTC should be configured and enabled; if you are using the System Clock driver, this may be done via `conf_clocks.h`.

10.1.1.1. Clocks and Oscillators

The `conf_clock.h` file needs to be changed with the following values to configure the clocks and oscillators for the module.

The following oscillator settings are needed:

```
/* SYSTEM_CLOCK_SOURCE_OSC32K configuration - Internal 32KHz oscillator */
#  define CONF_CLOCK_OSC32K_ENABLE                 true
#  define CONF_CLOCK_OSC32K_STARTUP_TIME           SYSTEM_OSC32K_STARTUP_130
#  define CONF_CLOCK_OSC32K_ENABLE_1KHZ_OUTPUT     true
#  define CONF_CLOCK_OSC32K_ENABLE_32KHZ_OUTPUT    true
#  define CONF_CLOCK_OSC32K_ON_DEMAND              true
#  define CONF_CLOCK_OSC32K_RUN_IN_STANDBY         false
```

The following generic clock settings are needed:

```
/* Configure GCLK generator 2 (RTC) */
#  define CONF_CLOCK_GCLK_2_ENABLE                 true
#  define CONF_CLOCK_GCLK_2_RUN_IN_STANDBY         false
#  define CONF_CLOCK_GCLK_2_CLOCK_SOURCE
SYSTEM_CLOCK_SOURCE_OSC32K
#  define CONF_CLOCK_GCLK_2_PRESCALER              32
#  define CONF_CLOCK_GCLK_2_OUTPUT_ENABLE          false
```

10.1.2. Setup

10.1.2.1. Initialization Code

Create an rtc_module struct and add to the main application source file, outside of any functions:

```
struct rtc_module rtc_instance;
```

Copy-paste the following setup code to your application:

```
void configure_rtc_calendar(void)
{
```

```
    /* Initialize RTC in calendar mode. */
    struct rtc_calendar_config config_rtc_calendar;
    rtc_calendar_get_config_defaults(&config_rtc_calendar);

    struct rtc_calendar_time alarm;
    rtc_calendar_get_time_defaults(&alarm);
    alarm.year   = 2013;
    alarm.month  = 1;
    alarm.day    = 1;
    alarm.hour   = 0;
    alarm.minute = 0;
    alarm.second = 4;

    config_rtc_calendar.clock_24h     = true;
    config_rtc_calendar.alarm[0].time = alarm;
    config_rtc_calendar.alarm[0].mask = RTC_CALENDAR_ALARM_MASK_YEAR;

    rtc_calendar_init(&rtc_instance, RTC, &config_rtc_calendar);

    rtc_calendar_enable(&rtc_instance);
}
```

10.1.2.2. Add to Main

Add the following to `main()`.

```
system_init();

struct rtc_calendar_time time;
time.year   = 2012;
time.month  = 12;
time.day    = 31;
time.hour   = 23;
time.minute = 59;
time.second = 59;

configure_rtc_calendar();

/* Set current time. */
rtc_calendar_set_time(&rtc_instance, &time);

rtc_calendar_swap_time_mode(&rtc_instance);
```

10.1.2.3. Workflow

1. Make configuration structure.

   ```
   struct rtc_calendar_config config_rtc_calendar;
   ```

2. Fill the configuration structure with the default driver configuration.

   ```
   rtc_calendar_get_config_defaults(&config_rtc_calendar);
   ```

   Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

3. Make time structure for alarm and set with default and desired values.

   ```
   struct rtc_calendar_time alarm;
   rtc_calendar_get_time_defaults(&alarm);
   alarm.year   = 2013;
   alarm.month  = 1;
   alarm.day    = 1;
   alarm.hour   = 0;
   ```

```
alarm.minute = 0;
alarm.second = 4;
```

4. Change configurations as desired.

```
config_rtc_calendar.clock_24h     = true;
config_rtc_calendar.alarm[0].time = alarm;
config_rtc_calendar.alarm[0].mask = RTC_CALENDAR_ALARM_MASK_YEAR;
```

5. Initialize module.

```
rtc_calendar_init(&rtc_instance, RTC, &config_rtc_calendar);
```

6. Enable module.

```
rtc_calendar_enable(&rtc_instance);
```

10.1.3. Implementation

Add the following to `main()`.

```
while (true) {
    if (rtc_calendar_is_alarm_match(&rtc_instance, RTC_CALENDAR_ALARM_0)) {
        /* Do something on RTC alarm match here */
        port_pin_toggle_output_level(LED_0_PIN);

        rtc_calendar_clear_alarm_match(&rtc_instance,
RTC_CALENDAR_ALARM_0);
    }
}
```

10.1.3.1. Workflow

1. Start an infinite loop, to continuously poll for a RTC alarm match.

```
while (true) {
```

2. Check to see if a RTC alarm match has occurred.

```
if (rtc_calendar_is_alarm_match(&rtc_instance, RTC_CALENDAR_ALARM_0)) {
```

3. Once an alarm match occurs, perform the desired user action.

```
/* Do something on RTC alarm match here */
port_pin_toggle_output_level(LED_0_PIN);
```

4. Clear the alarm match, so that future alarms may occur.

```
rtc_calendar_clear_alarm_match(&rtc_instance, RTC_CALENDAR_ALARM_0);
```

10.2. Quick Start Guide for RTC (CAL) - Callback

In this use case, the RTC is set up in calendar mode. The time is set and an alarm is enabled, as well as a callback for when the alarm time is hit. Each time the callback fires, the alarm time is reset to five seconds in the future and the board LED toggled.

10.2.1. Prerequisites

The Generic Clock Generator for the RTC should be configured and enabled; if you are using the System Clock driver, this may be done via `conf_clocks.h`.

10.2.1.1. Clocks and Oscillators

The `conf_clock.h` file needs to be changed with the following values to configure the clocks and oscillators for the module.

The following oscillator settings are needed:

```
/* SYSTEM_CLOCK_SOURCE_OSC32K configuration - Internal 32KHz oscillator */
#  define CONF_CLOCK_OSC32K_ENABLE                  true
#  define CONF_CLOCK_OSC32K_STARTUP_TIME            SYSTEM_OSC32K_STARTUP_130
#  define CONF_CLOCK_OSC32K_ENABLE_1KHZ_OUTPUT      true
#  define CONF_CLOCK_OSC32K_ENABLE_32KHZ_OUTPUT     true
#  define CONF_CLOCK_OSC32K_ON_DEMAND               true
#  define CONF_CLOCK_OSC32K_RUN_IN_STANDBY          false
```

The following generic clock settings are needed:

```
/* Configure GCLK generator 2 (RTC) */
#  define CONF_CLOCK_GCLK_2_ENABLE                  true
#  define CONF_CLOCK_GCLK_2_RUN_IN_STANDBY          false
#  define CONF_CLOCK_GCLK_2_CLOCK_SOURCE
SYSTEM_CLOCK_SOURCE_OSC32K
#  define CONF_CLOCK_GCLK_2_PRESCALER               32
#  define CONF_CLOCK_GCLK_2_OUTPUT_ENABLE           false
```

10.2.2. Setup

10.2.2.1. Code

Create an rtc_module struct and add to the main application source file, outside of any functions:

```
struct rtc_module rtc_instance;
```

The following must be added to the user application:

Function for setting up the module:

```
void configure_rtc_calendar(void)
{
    /* Initialize RTC in calendar mode. */
    struct rtc_calendar_config config_rtc_calendar;
    rtc_calendar_get_config_defaults(&config_rtc_calendar);

    alarm.time.year      = 2013;
    alarm.time.month     = 1;
    alarm.time.day       = 1;
    alarm.time.hour      = 0;
    alarm.time.minute    = 0;
    alarm.time.second    = 4;

    config_rtc_calendar.clock_24h = true;
    config_rtc_calendar.alarm[0].time = alarm.time;
    config_rtc_calendar.alarm[0].mask = RTC_CALENDAR_ALARM_MASK_YEAR;

    rtc_calendar_init(&rtc_instance, RTC, &config_rtc_calendar);

    rtc_calendar_enable(&rtc_instance);
}
```

Callback function:

```
void rtc_match_callback(void)
{
    /* Do something on RTC alarm match here */
```

```
    port_pin_toggle_output_level(LED_0_PIN);

    /* Set new alarm in 5 seconds */
    alarm.mask = RTC_CALENDAR_ALARM_MASK_SEC;

    alarm.time.second += 5;
    alarm.time.second = alarm.time.second % 60;

    rtc_calendar_set_alarm(&rtc_instance, &alarm, RTC_CALENDAR_ALARM_0);
}
```

Function for setting up the callback functionality of the driver:

```
void configure_rtc_callbacks(void)
{
    rtc_calendar_register_callback(
            &rtc_instance, rtc_match_callback,
RTC_CALENDAR_CALLBACK_ALARM_0);
    rtc_calendar_enable_callback(&rtc_instance,
RTC_CALENDAR_CALLBACK_ALARM_0);
}
```

Add to user application `main()`:

```
system_init();

struct rtc_calendar_time time;
rtc_calendar_get_time_defaults(&time);
time.year   = 2012;
time.month  = 12;
time.day    = 31;
time.hour   = 23;
time.minute = 59;
time.second = 59;

/* Configure and enable RTC */
configure_rtc_calendar();

/* Configure and enable callback */
configure_rtc_callbacks();

/* Set current time. */
rtc_calendar_set_time(&rtc_instance, &time);
```

10.2.2.2. Workflow

1. Initialize system.

   ```
   system_init();
   ```

2. Create and initialize a time structure.

   ```
   struct rtc_calendar_time time;
   rtc_calendar_get_time_defaults(&time);
   time.year   = 2012;
   time.month  = 12;
   time.day    = 31;
   time.hour   = 23;
   time.minute = 59;
   time.second = 59;
   ```

3. Configure and enable module.

```
configure_rtc_calendar();
```

   1. Create an RTC configuration structure to hold the desired RTC driver settings and fill it with the default driver configuration values.

```
struct rtc_calendar_config config_rtc_calendar;
rtc_calendar_get_config_defaults(&config_rtc_calendar);
```

      Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

   2. Create and initialize an alarm.

```
alarm.time.year      = 2013;
alarm.time.month     = 1;
alarm.time.day       = 1;
alarm.time.hour      = 0;
alarm.time.minute    = 0;
alarm.time.second    = 4;
```

   3. Change settings in the configuration and set alarm.

```
config_rtc_calendar.clock_24h = true;
config_rtc_calendar.alarm[0].time = alarm.time;
config_rtc_calendar.alarm[0].mask = RTC_CALENDAR_ALARM_MASK_YEAR;
```

   4. Initialize the module with the set configurations.

```
rtc_calendar_init(&rtc_instance, RTC, &config_rtc_calendar);
```

   5. Enable the module.

```
rtc_calendar_enable(&rtc_instance);
```

4. Configure callback functionality.

```
configure_rtc_callbacks();
```

   1. Register overflow callback.

```
rtc_calendar_register_callback(
        &rtc_instance, rtc_match_callback,
RTC_CALENDAR_CALLBACK_ALARM_0);
```

   2. Enable overflow callback.

```
rtc_calendar_enable_callback(&rtc_instance,
RTC_CALENDAR_CALLBACK_ALARM_0);
```

5. Set time of the RTC calendar.

```
rtc_calendar_set_time(&rtc_instance, &time);
```

10.2.3. Implementation

10.2.3.1. Code

Add to user application main:

```
while (true) {
    /* Infinite loop */
}
```

10.2.3.2. Workflow

1. Infinite while loop while waiting for callbacks.

```
while (true) {
```

10.2.4. Callback

Each time the RTC time matches the configured alarm, the callback function will be called.

10.2.4.1. Workflow

1. Create alarm struct and initialize the time with current time.

```
struct rtc_calendar_alarm_time alarm;
```

2. Set alarm to trigger on seconds only.

```
alarm.mask = RTC_CALENDAR_ALARM_MASK_SEC;
```

3. Add one second to the current time and set new alarm.

```
alarm.time.second += 5;
alarm.time.second = alarm.time.second % 60;

rtc_calendar_set_alarm(&rtc_instance, &alarm, RTC_CALENDAR_ALARM_0);
```

11. Document Revision History

Doc. Rev.	Date	Comments
42126E	12/2015	Added support for SAM L21/L22, SAM C21, SAM D09, and SAM DA1
42126D	12/2014	Added support for SAM R21 and SAM D10/D11
42126C	01/2014	Added support for SAM D21
42126B	06/2013	Added additional documentation on the event system. Corrected documentation typos
42126A	06/2013	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**