AN3371
Application note

Using the STM32 hardware real-time clock (RTC)

Introduction

A real-time clock (RTC) is a computer clock that keeps track of the current time. Although RTCs are often used in personal computers, servers and embedded systems, they are also present in almost any electronic device that requires accurate time keeping. Microcontrollers supporting RTC can be used for chronometers, alarm clocks, watches, small electronic agendas, and many other devices.

This application note describes the features of the real-time clock (RTC) controller embedded in **Ultra Low Power Medium-density** and **F-2 series** devices microcontrollers, and the steps required to configure the RTC for use with the calendar, alarm, periodic wakeup unit and tamper detection.

Examples are provided with configuration information to enable you to quickly and correctly configure the RTC for calendar, alarm, periodic wakeup unit, tamper detection, time stamp and calibration applications.

Note: 1 *All examples and explanations are based on the STM32L1xx and STM32F2xx firmware libraries and reference manuals of STM32L1xx (RM0038) and STM32F2xx (RM0033).*

2 *STM32 refers to Ultra Low Power Medium-density devices and F-2 series devices microcontrollers in this document.*

3 *Ultra Low Power Medium (ULPM) density devices are STM32L151xx and STM32L152xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.*

4 *F-2 series devices are STM32F205xx, STM32F207xx, STM32F215xx and STM32F217xx microcontrollers.*

Contents

1 Overview of the STM32 advanced RTC

The RTC embedded in STM32 microcontrollers can be used to provide a full-featured calendar, alarm and periodic wakeup unit, digital calibration and advanced tamper detection.

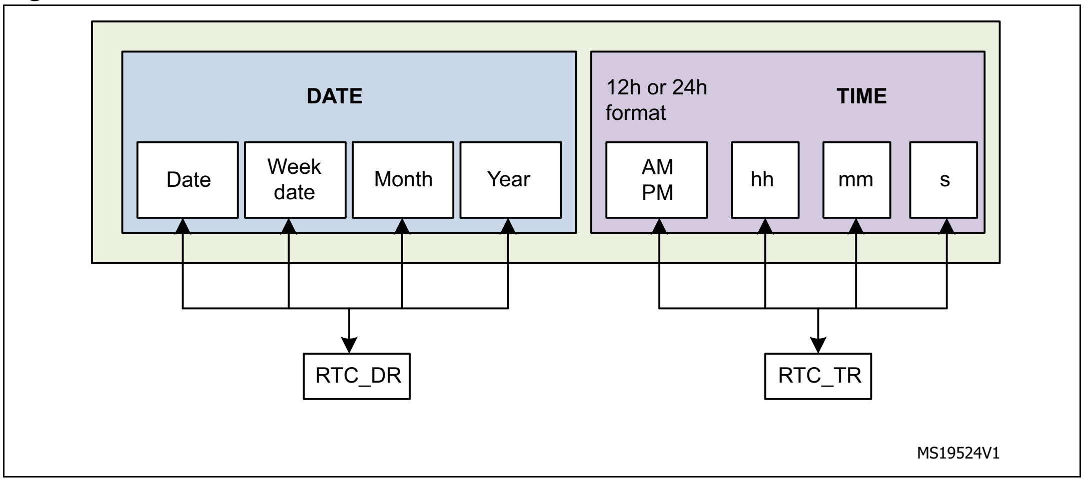
Refer to *Section 2: RTC features summary* for the complete list of features available on Ultra Low Power Medium-density and F-2 series devices.

1.1 RTC calendar

A calendar keeps track of the time (hours, minutes and seconds) and date (day, week, month, year). The STM32 RTC calendar offers several features to help you easily configure and display the calendar data fields:

- Calendar with:
  - seconds
  - minutes
  - hours in 12-hour or 24-hour format
  - day of the week (day)
  - day of the month (date)
  - month
  - year
- Calendar in binary-coded decimal (BCD) format
- Automatic management of 28-, 29- (leap year), 30-, and 31-day months
- Daylight saving time adjustment programmable by software

Figure 1. RTC calendar fields



1. RCT_DR, RTC_TR are RTC Date and Time registers.

A software calendar can be a software counter (usually 32 bits long) that represents the number of seconds. Software routines convert the counter value to hours, minutes, day of the month, day of the week, month and year. This data can be converted to BCD format and displayed on a standard LCD, which is useful in countries that use the 12-hour format with an *AM*/*PM* indicator (see *Figure 2*). Conversion routines use significant program memory space and are CPU-time consuming, which may be critical in certain real-time applications.

When using the STM32 RTC calendar, software conversion routines are no longer needed because their functions are performed by hardware.

STM32 RTC calendar is provided in BCD format. This avoids binary to BCD software conversion routines, which use significant program memory space and a CPU-load that may be critical in certain real-time applications.

Figure 2. Example of calendar display on an LCD

```
11:15:28 PM
WED NOV 19 2010
```

1.1.1 Initializing the calendar

Table 1 describes the steps required to correctly configure the calendar time and date.

Table 1. Steps to initialize the calendar

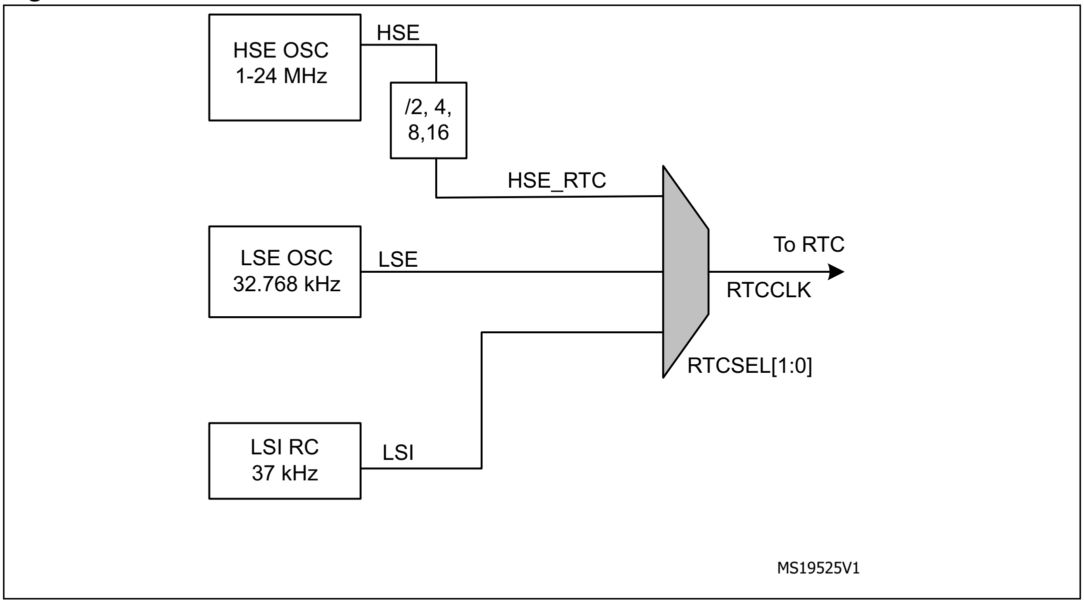
Step	What to do	How to do it	Comments
1	Disable the RTC registers Write protection	Write "0xCA" and then "0x53" into the RTC_WPR register	RTC registers can be modified
2	Enter Initialization mode	Set INIT bit to ‘1’ in RTC_ISR register	The calendar counter is stopped to allow update
3	Wait for the confirmation of Initialization mode (clock synchronization)	Poll INITF bit of in RTC_ISR until it is set	It takes approximately 2 RTCCLK clock cycles for medium density devices
4	Program the prescalers register if needed	Register RTC_PRER: Write first the synchronous value and then write the asynchronous	By default, the prescalers register RTC_PRER is initialized to provide 1Hz to the Calendar unit when RTCCLK = 32768Hz
5	Load time and date values in the shadow registers	Set RTC_TR and RTC_DR registers	
6	Configure the time format (12h or 24h)	Set FMT bit in RTC_CR register	FMT = 0: 24 hour/day format FMT = 1: AM/PM hour format
7	Exit Initialization mode	Clear the INIT bit in RTC_ISR register	The current calendar counter is automatically loaded and the counting restarts after 4 RTCCLK clock cycles
8	Enable the RTC Registers Write Protection	Write "0xFF" into the RTC_WPR register	RTC Registers can no longer be modified

1.1.2 Delivering a 1 Hz signal to the calendar using different clock sources

The RTC features several prescalers that allow delivering a 1 Hz clock to the calendar unit, regardless of the clock source.

For ULPM density devices and For F-2 series devices, the RTC clock source (RTCCLK) can be LSE, LSI or HSE (see *Figure 3* and *Figure 4*).

Figure 3. STM32L1xx RTC clock sources



Note: RTCSEL[1:0] bits are the RCC Control/status register (RCC_CSR) [17:16] bits

Figure 4. STM32F2xx RTC clock sources

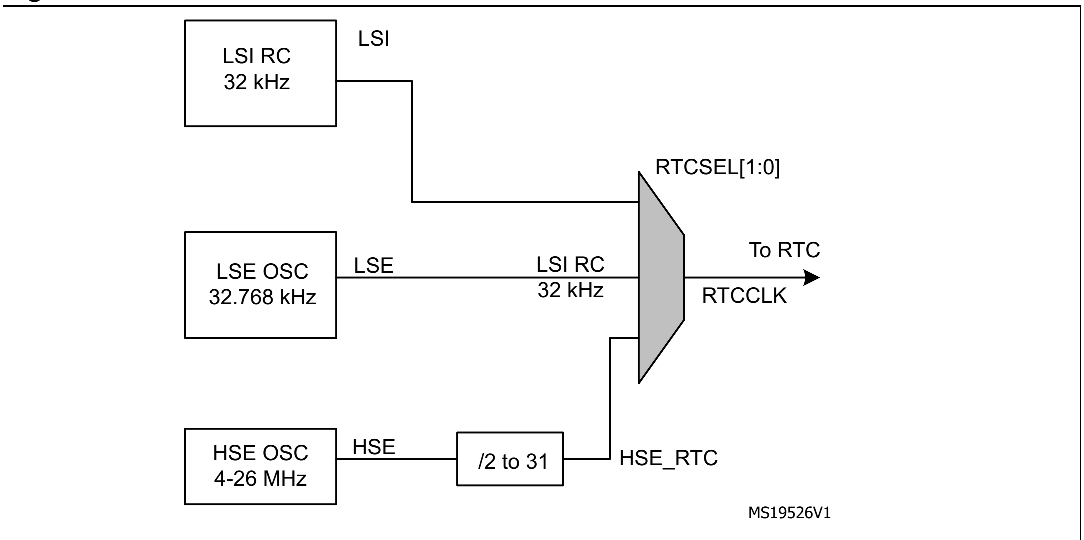
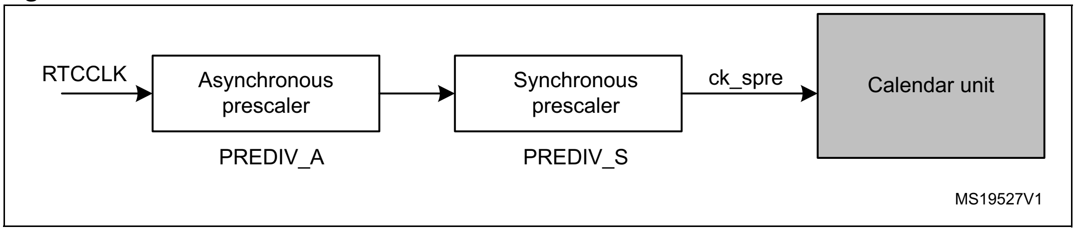


Figure 5. Prescalers from RTC clock source to calendar unit



The formula to calculate ck_spre is:

$$ck\_spre = \frac{RTCCLK}{(PREDIV\_A + 1) \times (PREDIV\_S + 1)}$$

where:

- RTCCLK can be any clock source: HSE_RTC, LSE or LSI
- PREDIV_A can be 1,2,3,..., or 127
- PREDIV_S can be 0,1,2,..., or 8191

Table 2 shows several ways to obtain the calendar clock (ck_spre) = 1 Hz.

Table 2. Calendar clock equal to 1Hz with different clock sources

RTCCLK Clock source	Prescalers		ck_spre
	PREDIV_A[6:0]	PREDIV_S[12:0]	
HSE_RTC = 1MHz	124 (div125)	7999 (div8000)	1 Hz
LSE = 32.768 kHz	127 (div128)	255 (div256)	1 Hz
LSI = 32 kHz(1)	127 (div128)	249 (div250)	1 Hz
LSI = 37 kHz(2)	124 (div125)	295 (div296)	1 Hz

1. For STM32L1xx, LSI = 37 KHz, but LSI accuracy is not suitable for calendar application.
2. For STM32F2xx, LSI = 32 KHz, but LSI accuracy is not suitable for calendar application.

1.2 RTC alarms

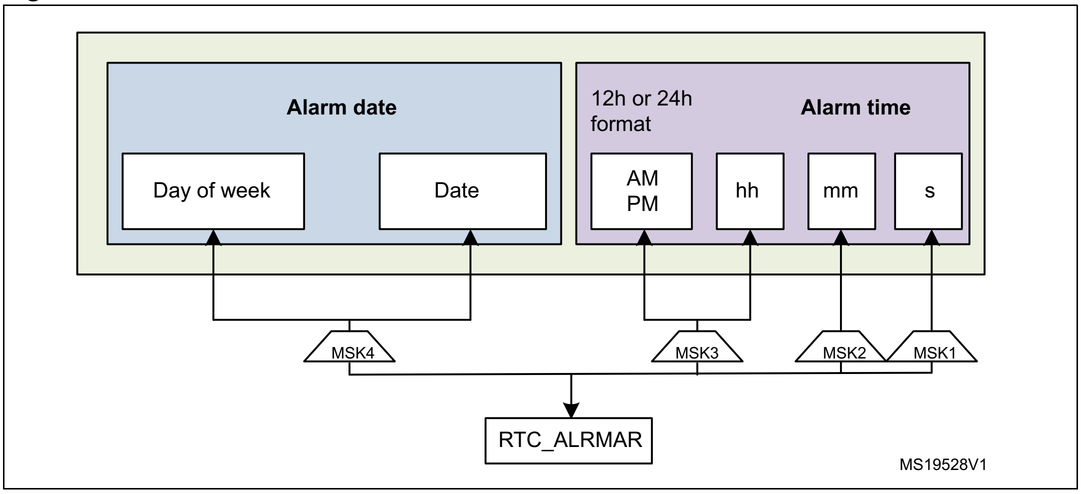
STM32 RTC embeds two alarms, Alarm A and Alarm B, which are similar. An alarm can be generated at a given time or/and date programmed by the user.

The STM32 RTC provides a rich combination of alarms settings, and offers many features to make it easy to configure and display these alarms settings.

Each alarm unit provides the following features:

- Fully programmable alarm: seconds, minutes, hours and date fields can be independently selected or masked to provide a rich combination of alarms.
- Ability to exit the device from low power modes when the alarm occurs.
- The alarm event can be routed to a specific output pin with configurable polarity.
- Dedicated alarm flags and interrupt.

Figure 6. Alarm A fields



1. RTC_ALRMAR is a RTC register. The same fields are also available for the RTC_ALRMBR register.
2. MSKx are bits in the RTC_ALARM register that enable/disable the RTC_ALARM fields used for alarm A and calendar comparison. For more details refer to *Table 4*.

An alarm consists of a register with the same length as the RTC time counter. When the RTC time counter reaches the value programmed in the alarm register, a flag is set to indicate that an alarm event occurred.

The STM32 RTC alarm can be configured by hardware to generate different types of alarms. For more details refer to *Table 4*.

1.2.1 Programming the alarm

Table 3 describes the steps required to configure alarm A.

Table 3. Steps to configure the alarm

Step	What to do	How to do it	Comments
1	Disable the RTC registers Write protection	Write "0xCA" and then "0x53" into the RTC_WPR register	RTC registers can be modified
2	Disable alarm A	Clear ALRAE(1) bit in RTC_CR register.	
3	Check that the RTC_ALRMAR register can be accessed	Poll ALRAWF(2) bit until it is set in RTC_ISR.	It takes approximately two RTCCLK clock cycles (clock synchronization).
4	Configure the alarm	Configure RTC_ALRMAR(3) register.	The alarm hour format must be the same(4) as the RTC Calendar in RTC_ALRMAR.
5	Re-enable alarm A	Set ALRAE(5) bit in RTC_CR register.	
6	Enable the RTC registers Write protection	Write "0xFF" into the RTC_WPR register	RTC registers can no longer be modified

1. Respectively ALRBE bit for Alarm B.
2. Respectively ALRBWF bit for Alarm B.
3. Respectively RTC_ALRMBR register for Alarm B.
4. As an example, if the alarm is configured to occur at 3:00:00 PM, the alarm will not occur even if the calendar time is 15:00:00, because the RTC calendar is 24-hour format and the alarm is 12-hour format.
5. Respectively ALRBE bit for Alarm B.
6. RTC Alarm registers can only be written when the corresponding RTC Alarm is disabled or during RTC Initialization mode.

1.2.2 Configuring the alarm behavior using the MSKx bits

The alarm behavior can be configured using the MSKx bits (x = 1, 2, 3, 4) of the RTC_ALRMAR register for alarm A (RTC_ALRMBR register for alarm B).

Table 4 shows all the possible alarm settings. As an example, to configure the alarm time to 23:15:07 on Monday (assuming that the WDSEL = 1), MSKx bits must be set to 0000b. When the WDSEL = 0, all cases are similar with the exception that the Alarm Mask field compares with day number and not the day of the week, and MSKx bits must be set to 0000b.

Table 4. Alarm combinations

MSK4	MSK3	MSK2	MSK1	Alarm behavior
0	0	0	0	**All fields are used in alarm comparison:** Alarm occurs at 23:15:07, each Monday.
0	0	0	1	**Seconds do not matter in alarm comparison** The alarm occurs every second of 23:15, each Monday.

Table 4. Alarm combinations (continued)

MSK4	MSK3	MSK2	MSK1	Alarm behavior
0	0	1	0	**Minutes do not matter in alarm comparison** The alarm occurs at the 7th second of every minute of 23:XX, each Monday.
0	0	1	1	**Minutes and seconds do not matter in alarm comparison**
0	1	0	0	**Hours do not matter in alarm comparison**
0	1	0	1	**Hours and seconds do not matter in alarm comparison**
0	1	1	0	**Hours and minutes do not matter in alarm comparison**
0	1	1	1	**Hours, minutes and seconds do not matter in alarm comparison** The alarm is set every second, each Monday, during the whole day.
1	0	0	0	**Week day (or date, if selected) do not matter in alarm comparison** Alarm occurs all days at 23:15:07.
1	0	0	1	**Week day and seconds do not matter in alarm comparison**
1	0	1	0	**Week day and minutes do not matter in alarm comparison**
1	0	1	1	**Week day, minutes and seconds do not matter in alarm comparison**
1	1	0	0	**Week day and Hours do not matter in alarm comparison**
1	1	0	1	**Week day, Hours and seconds do not matter in alarm comparison**
1	1	1	0	**Week day, Hours and minutes do not matter in alarm comparison**
1	1	1	1	**Alarm occurs every second**

Caution: If the seconds field is selected (MSK1 bit reset in RTC_ALRMAR or RTC_ALRMBR), the synchronous prescaler division factor PREDIV_S set in the RTC_PRER register must be at least 3 to ensure correct behavior.

1.3 RTC periodic wakeup unit

Like many STMicroelectronics microcontrollers, the STM32 provides several low power modes to reduce power consumption.

The STM32 features a periodic timebase and wakeup unit that can wake up the system when the STM32 operates in low power modes. This unit is a programmable downcounting auto-reload timer. When this counter reaches zero, a flag and an interrupt (if enabled) are generated.

The wakeup unit has the following features:

- Programmable downcounting auto-reload timer.
- Specific flag and interrupt capable of waking up the device from low power modes.
- Wakeup alternate function output which can be routed to RTC_ALARM output (unique pad for Alarm A, Alarm B or Wakeup events) with configurable polarity.
- A full set of prescalers to select the desired waiting period.

1.3.1 Programming the Auto-wakeup unit

Table 5 describes the steps required to configure the Auto-wakeup unit.

Table 5. Steps to configure the Auto wake-up unit

Step	What to do	How to do it	Comments
1	Disable the RTC registers Write protection	Write "0xCA" and then "0x53" into the RTC_WPR register	RTC Registers can be modified
2	Disable the wakeup timer.	Clear WUTE bit in RTC_CR register	
3	Ensure access to Wakeup auto-reload counter and bits WUCKSEL[2:0] is allowed.	Poll WUTWF until it is set in RTC_ISR	It takes approximately 2 RTCCLK clock cycles
4	Program the value into the wakeup timer.	Set WUT[15:0] in RTC_WUTR register	See *Section 1.3.2: Maximum and minimum RTC wakeup period*
5	Select the desired clock source.	Program WUCKSEL[2:0] bits in RTC_CR register	
6	Re-enable the wakeup timer.	Set WUTE bit in RTC_CR register	The wakeup timer restarts downcounting
7	Enable the RTC registers Write protection	Write "0xFF" into the RTC_WPR register	RTC registers can no more be modified

1.3.2 Maximum and minimum RTC wakeup period

The wakeup unit clock is configured through the WUCKSEL[2:0] bits of RTC_CR1 register. Three different configurations are possible:

- Configuration 1: WUCKSEL[2:0] = 0xxb for short wakeup periods (see *Section : Periodic timebase/wakeup configuration for clock configuration 1*)
- Configuration 2: WUCKSEL[2:0] = 10xb for medium wakeup periods (see *Section : Periodic timebase/wakeup configuration for clock configuration 2*)
- Configuration 3: WUCKSEL[2:0] = 11xb for long wakeup periods (see *Section : Periodic timebase/wakeup configuration for clock configuration 3*)

Periodic timebase/wakeup configuration for clock configuration 1

Figure 7 shows the prescalers connection to the timebase/wakeup unit and *Table 6* gives the timebase/wakeup clock resolutions corresponding to configuration 1.

The prescaler depends on the Wakeup clock selection:

- WUCKSEL[2:0] =000: RTCCLK**/16** clock is selected
- WUCKSEL[2:0] =001: RTCCLK**/8** clock is selected
- WUCKSEL[2:0] =010: RTCCLK**/4** clock is selected
- WUCKSEL[2:0] =011: RTCCLK**/2** clock is selected

Figure 7. Prescalers connected to the timebase/wakeup unit for configuration 1

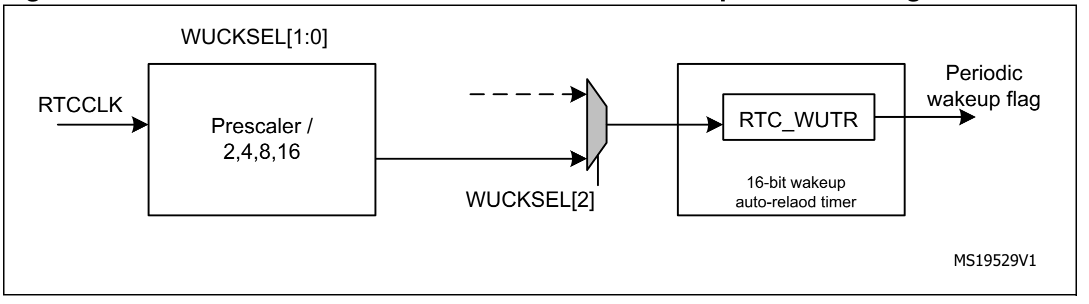


Table 6. Timebase/wakeup unit period resolution with clock configuration 1

Clock source	Wakeup period resolution	
	WUCKSEL[2:0] = 000b (div16)	WUCKSEL[2:0] = 011b (div2)
LSE = 32 768 Hz	488.28 µs	61.035 µs

When RTCCLK= 32768 Hz, the minimum timebase/wakeup resolution is 61.035 µs, and the maximum resolution 488.28µs. As a result:

- The minimum timebase/wakeup period is (0x0001 + 1) x 61.035 µs = 122.07 µs.

  The timebase/wakeup timer counter WUT[15:0] cannot be set to 0x0000 with WUCKSEL[2:0]=011b (f_{RTCCLK}/2) because this configuration is prohibited. *Refer to the STM32 reference manuals for more details*.
- The maximum timebase/wakeup period is (0xFFFF+ 1) x 488.28 µs = 2 s.

Periodic timebase/wakeup configuration for clock configuration 2

Figure 8 shows the prescaler connection to the timebase/wakeup unit and *Table 7* gives the timebase/wakeup clock resolutions corresponding to configuration 2.

Figure 8. Prescalers connected to the wake up unit for configurations 2 and 3

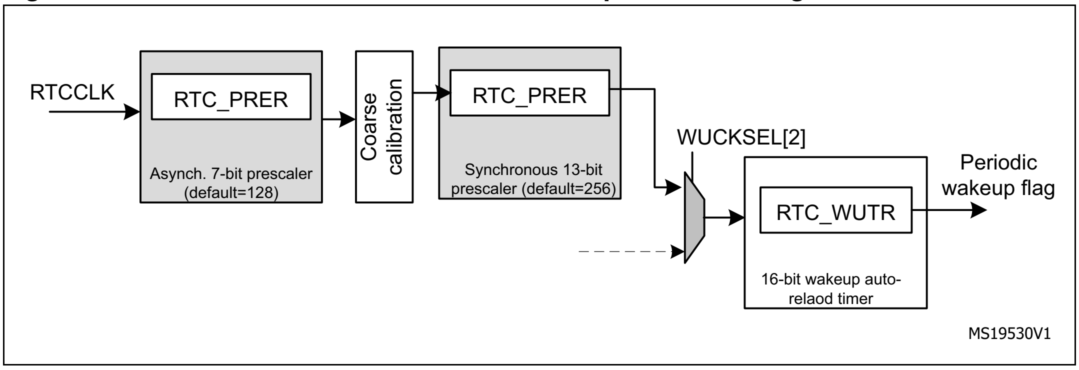


Table 7. Timebase/wakeup unit period resolution with clock configuration 2

Clock source	Wakeup period resolution	
	PREDIV_A[6:0] = div128 PREDIV_S[12:0] = div8192	PREDIV_A[6:0] = div2(1) PREDIV_S[12:0] = div1
LSE = 32 768 Hz	**32 s**	61.035 µs

1. PREDIV_A minimum value is '1' on medium density devices.

When RTCCLK= 32768 Hz, the minimum resolution for configuration 2 is 61.035 µs, and the maximum resolution 32s.

As a result:

- The minimum timebase/wakeup period is (0x0000 + 1) x 61.035 µs = 122.07 µs.
- The maximum timebase/wakeup period is (0xFFFF+ 1) x 32s = 131072 s (more than 36 hours).

Periodic timebase/wakeup configuration for clock configuration 3

For this configuration, the resolution is the same as for configuration 2. However the timebase/wakeup counter downcounts starting from **0x1FFFF** to 0x00000, instead of **0xFFFF** to 0x0000 for configuration 2.

When RTCCLK= 32768,

- The minimum timebase/wakeup period is:
  (0x10000 + 1) x 61.035 µs = 250.06 ms
- The maximum timebase/wakeup period is:
  (0x1FFFF+ 1) x 32s = 4194304s (more than 48 day).

Summary of timebase/wakeup period extrema

When RTCCLK= 32768 Hz, the minimum and maximum period values, according on the configuration, are listed in *Table 8*.

Table 8. Min. and max. timebase/wakeup period when RTCCLK= 32768

Configuration	Minimum period	Maximum period
1	122.07 µs	2s
2	122.07 µs	more than 36 hours
3	250.06 ms	more than 48 day

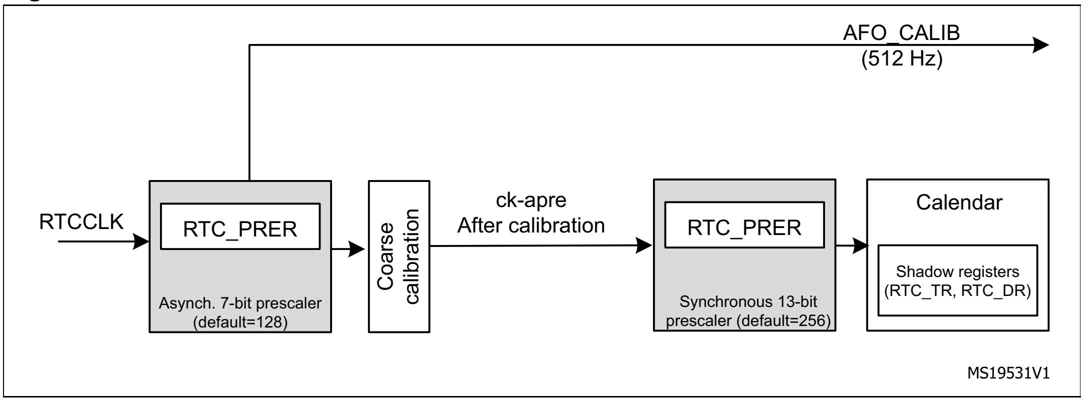
1. These values are calculated when RTCCLK = 32768 Hz

1.4 RTC coarse calibration

The digital coarse calibration can be used to compensate crystal inaccuracy by adding (positive calibration) or masking (negative calibration) clock cycles at the output of the asynchronous prescaler (ck_apre).

Negative calibration can be performed with a resolution of about 2 ppm while positive calibration can be performed with a resolution of about 4 ppm. The maximum calibration ranges from -63 ppm to 126 ppm.

Figure 9. Calibration block



You can calculate the clock deviation using AFO_CALIB, then update the calibration block. It is not possible to check the calibration result as the 512 Hz output is *before* the calibration block.

Note: 1 The calibration settings can only be changed during initialization.

The full calibration cycle lasts 64 minutes.

The calibration is done during the first minutes (from 0 to 62min depending to the configuration) of the calibration cycle.

2 We recommend the use of coarse calibration for static correction only. Due to the points listed in note 1, changing the calibration settings brings errors:

- Entering initialization mode stops the calendar and reinitialize the prescalers

- The calibration change rate must be very much smaller than the calibration window size in order to minimize the impact of the error brought by the change on the final accuracy.

Consequently, the coarse calibration is not adequate for dynamic calibration (such as compensation of the quartz variations due to external temperature changes).

3 *The reference clock calibration and the coarse calibration can not be used together.*

Caution: Digital coarse calibration may not work correctly if PREDIV_A < 6.

1.5 RTC reference clock detection

The reference clock (at 50 Hz or 60 Hz) should have a higher precision than the 32.768 kHz LSE clock. This is why the RTC provides a reference clock input (RTC_50Hz pin) that can be used to compensate the imprecision of the calendar frequency (1 Hz).

The RTC_50Hz pin should be configured in input floating mode.

This mechanism enables the calendar to be as precise as the reference clock.

The reference clock detection is enabled by setting REFCKON bit of the RTC_CR register.

When the reference clock detection is enabled, PREDIV_A and PREDIV_S must be set to their default values: PREDIV_A = 0x007F and PREVID_S = 0x00FF.

When the reference clock detection is enabled, each 1 Hz clock edge is compared to the nearest reference clock edge (if one is found within a given time window). In most cases, the two clock edges are properly aligned. When the 1 Hz clock becomes misaligned due to the imprecision of the LSE clock, the RTC shifts the 1 Hz clock a bit so that future 1 Hz clock edges are aligned. The update window is 3 ck_calib periods (ck_calib is the output of the coarse calib block).

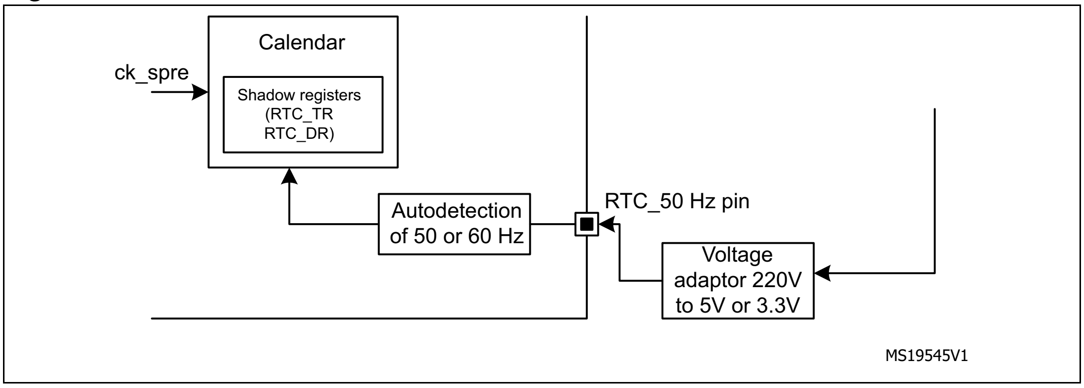
If the reference clock halts, the calendar is updated continuously based solely on the LSE clock. The RTC then waits for the reference clock using a detection window centered on the Synchronous Prescaler output clock (ck_spre) edge. The detection window is 7 ck_calib periods.

The reference clock can have a large local deviation (for instance in the range of 500ppm), but in the long term it must be much more precise than 32 kHz quartz.

The detection system is used only when the reference clock needs to be detected back after a loss. As the detection window is a bit larger than the reference clock period, this detection system brings an uncertainty of 1 ck_ref period (20 ms for a 50Hz reference clock) because we can have 2 ck_ref edges in the detection window. Then the update window is used, which brings no error as it is smaller than the reference clock period.

We assume than the ck_ref is not lost more than once a day. So the total uncertainty per month would be 20ms *1* 30 = 0.6s, which is much less than the uncertainty of a typical quartz (1.53 minutes per month for 35ppm quartz).

Figure 10. RTC reference clock detection



Note: The reference clock calibration and the coarse calibration can not be used together.

The reference clock calibration is the best (ensures a high calibrated time) if the 50Hz is always available. If the 50Hz input is lost, the LSE can be used.

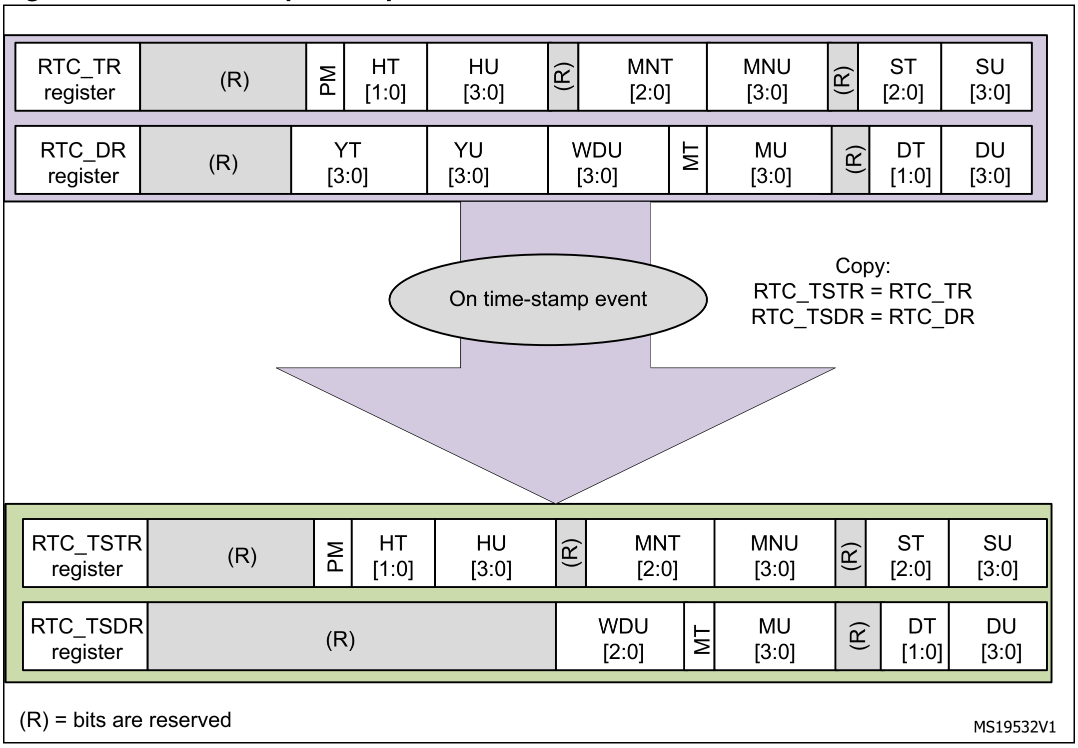
The reference clock detection can not be used in Vbat mode.

The reference clock calibration can only be used if you provide a precise 50 or 60Hz input.

1.6 Time-stamp

The Time-stamp feature provides the means to automatically save the current calendar.

Figure 11. Time-stamp event procedure



When the time-stamp is enabled, the calendar is saved in the time-stamp registers (RTC_TSTR, RTC_TSDR) when a time-stamp event is detected on the pin that the TIMESTAMP alternate function is mapped to. When a time-stamp event occurs, the time-stamp flag bit (TSF) in RTC_ISR register is set.

Table 9. Time-stamp features

What to do	How to do it	Comments
Enable Time-stamp	Setting the TSE bit of RTC_CR register to 1	
Map TIMESTAMP pin alternate function	Select with TSINSEL bit in RTC_TCR register	Only for F-2 series devices. The TIMESTAMP pin can be either PI8 or PC13.
Detect a time-stamp event by interrupt	Setting the TSIE bit in the RTC_CR register	An interrupt is generated when a time-stamp event occurs.
Detect a time-stamp event by polling	By polling on the time-stamp flag (TSF[(1)]) in the RTC_ISR register	To clear the flag, write zero on the TSF bit.[(2)]

Table 9. Time-stamp features (continued)

What to do	How to do it	Comments
Detect a Time-stamp overflow event[(3)]	By polling on the time-stamp over flow flag (TSOVF[(4)]) in the RTC_ISR register.	1- to clear the flag, write zero on the TSOVF bit. 2- time-stamp registers (RTC_TSTR and RTC_TSDR) maintain the results of the previous event. 3-If a time-stamp event occurs immediately after the TSF bit is supposed to be cleared, then both TSF and TSOVF bits are set

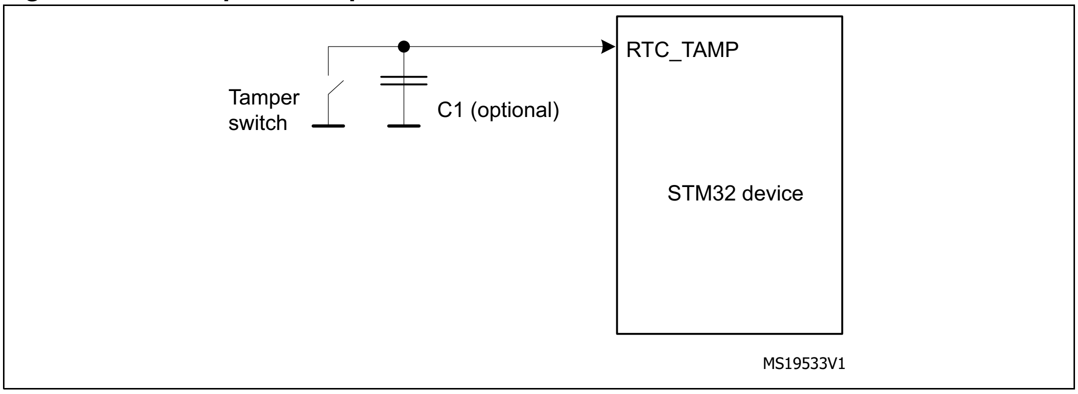
1. TSF is set 2 ck_apre cycles after the time-stamp event occurs due to synchronization process.
2. To avoid masking a time-stamp event occurring at the same moment, the application must not write ‘0’ into TSF bit unless it has already read it to‘1’.
3. Time-stamp overflow event is not connect to an interrupt.
4. There is no delay in the setting of TSOVF. This means that if two time-stamp events are close together, TSOVF can be seen as '1' while TSF is still '0'. As a consequence, it is recommended to poll TSOVF only after TSF has been set.

1.7 Back up registers and tamper detection

The RTC includes 1 tamper detection input. The tamper input active level can be configured. The tamper input has an individual flag (TAMP1F bit in RTC_ISR register).

A tamper detection event generates an interrupt when the TAMPIE bit in RTC_TAFCR register is set.

Figure 12. Example of tamper detection circuit



1. C1 is optional (filtering can be performed by software).

Table 10 shows possible features of tamper.

Table 10. Tamper features

What to do	How to do it	Comments
Enable Tamper	Set the TAMP1E bit of RTC_TAFCR register to 1	
select Tamper1 active edge detection	Select with TAMP1TRG bit in RTC_TAFCR register	The default edge is rising edge.

Table 10. Tamper features (continued)

What to do	How to do it	Comments
Map Tamper1 pin alternate function	Select with TAMP1INSEL bit in RTC_TAFCR register	For F-2 series devices, the Tamper1 pin can be either PI8 or PC13.
Detect a Tamper1event by interrupt	Set the TAMPIE bit in the RTC_TAFCR register	An interrupt is generated when tamper detection event occurs.
Detect a Tamper1 event by polling	Poll on the time-stamp flag (TAMP1F) in the RTC_ISR register	To clear the flag, write zero on the TAMP1F bit.

RTC_BKPxR, where x=0 to 19 are 20 backup registers (80 bytes) and are reset when a tamper detection event occurs. These registers are powered-on by VBAT when VDD is switched off, so that they are not reset by a system reset, and their contents remain valid when the device operates in low-power mode.

1.8 RTC and low-power modes

The RTC is designed to minimize the power consumption. The prescalers used for the calendar are divided into synchronous and asynchronous.

Increasing the value of the asynchronous prescaler reduces the power consumption.

The RTC continues working in reset mode and its registers are not reset except by a VDD or VBAT power on, if both supplies have previously been powered off or the Backup Domain is reset on STM32F2xx devices.

Registers are not reset except by a power-on reset. RTC register values are not lost after a reset and the calendar keeps the correct time and date.

After a system reset or a power-on reset, the STM32 operates in Run mode. In addition, the device supports five low power modes to achieve the best compromise between low power consumption, short startup time and available wakeup sources.

The RTC peripheral can be active in the following low power modes:

- Sleep mode
- Low power Run mode (only for ULPM density devices)
- Low power Sleep mode (only for ULPM density devices)
- Standby mode
- Stop mode

Refer to the low power modes section of the *STM32 reference manuals* for more details about low power modes.

1.9 Alternate function RTC outputs

The RTC peripheral has two outputs:

- RTC_CALIB: used to generate an external clock.
- RTC_ALARM: unique output resulting from the multiplexing of the RTC alarm and wakeup events.

1.9.1 RTC_CALIB output

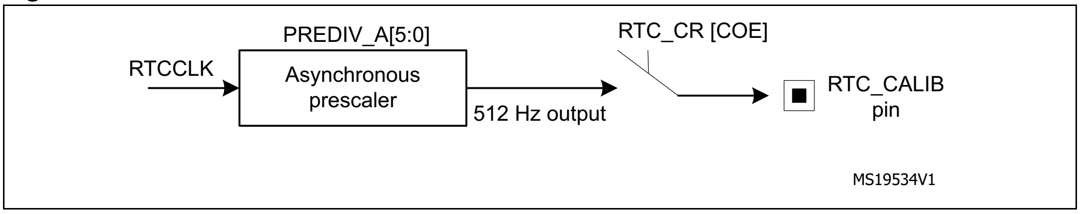
The RTC_CALIB output is used to generate a variable-frequency signal. Depending on the user application, this signal can play the role of a reference clock to calibrate an external device, or be connected to a buzzer to generate a sound.

The signal frequency is configured through the 6 LSB bits (PREDIV_A [5:0]) of the Asynchronous prescaler PREDIV_A[6:0].

RTC_CALIB is the output of the 5th stage of the 6-bit asynchronous prescaler PREDIV_A. If PREDIV_A[5]=0, no signal is output on RTC_CALIB.

When RTCCLK frequency is 32.768kHz and PREDIV_A[6:0] = 0x7F, RTC_CALIB frequency is 512Hz.

Figure 13. RTC_CALIB clock sources.



Maximum and minimum RTC_CALIB output frequency

The RTC can output the RTCCLK clock divided by a 6-bit asynchronous prescaler. The divider factor is configured through bits PREDIV_A[5:0] of the RTC_PRER register.

RTC_CALIB maximum and minimum frequencies are **31.250 kHz** and **500Hz,** respectively.

Table 11. RTC_CALIB output frequency versus clock source

RTC clock source	RTC_CALIB output frequency	
	Minimum (PREDIV_A[5:0] = 111 111b) (div64)	**Maximum (PREDIV_A[5:0] = 100 000b[1]) (div32)**
HSE_RTC=1MHz	15,625 kHz	31.250 KHz
LSE = 32768 Hz	512 Hz (default output frequency)	1.024 KHz
LSI[2] = 32 kHz	500 Hz	1 KHz
LSI[3] = 37 kHz	578.125 Hz	1156.25 Hz

1. PREDIV_A[5] must be set to ‘1’ to enable the RTC_CALIB output signal generation. If PREDIV_A[5] bit is zero, no signal is output on RTC_CALIB.
2. For STM32L1xx, LSI = 37 KHz
3. For STM32F2xx, LSI = 32 KHz

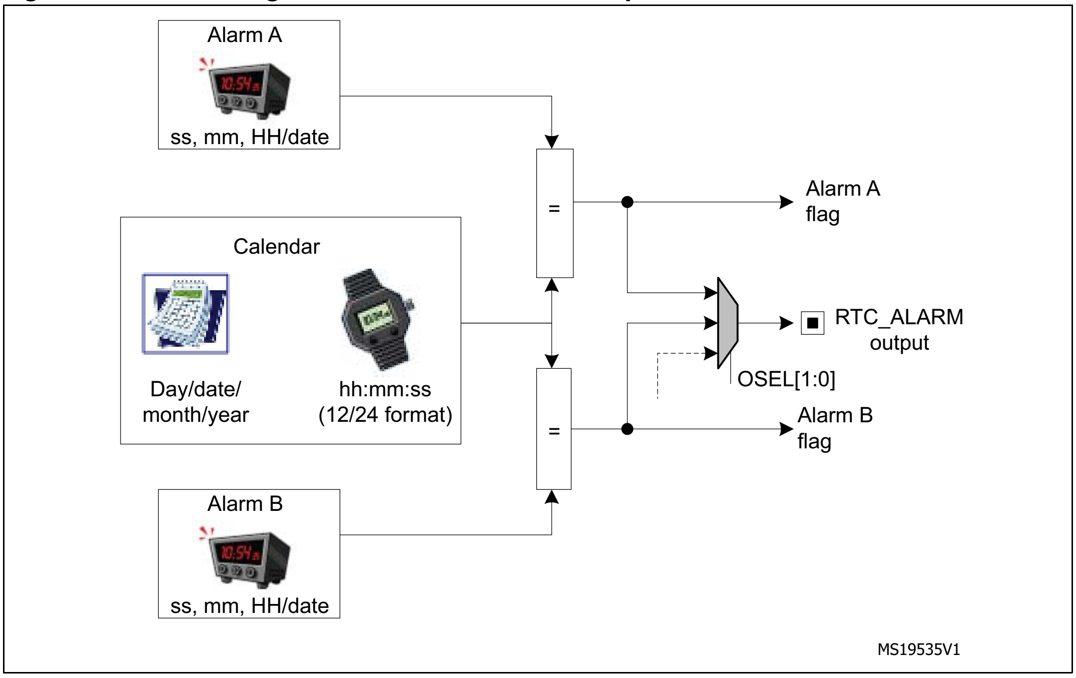
1.9.2 RTC_ALARM output

The RTC_ALARM output can be connected to the RTC alarm unit A or B to trigger an external action, or routed to the RTC wakeup unit to wake up an external device.

RTC_ALARM output connected to an RTC alarm unit

When the calendar reaches the value pre-programmed Alarm A in the RTC_ALRMAR register (TC_ALRMBR register for Alarm B), the alarm flag ALRAF bit (ALRBF bit), in RTC_ISR register, is set to ‘1’. If the alarm A or alarm B flag is routed to the RTC_ALARM output (RTC_CR_OSEL[1:0] =”01” for alarm A, and RTC_CR_OSEL[1:0] =”10” for alarm B), this pin is set to VDD or to GND, depending on the polarity selected. The output toggles when the selected alarm flag is cleared.

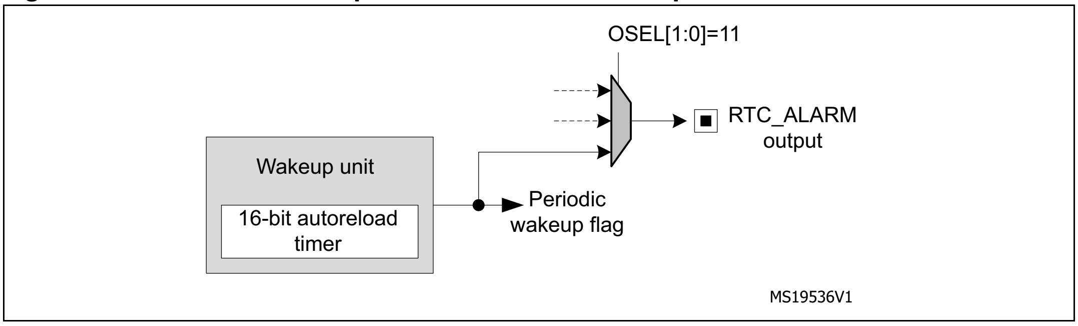
Figure 14. Alarm flag routed to RTC_ALARM output



RTC_ALARM output connected to the wakeup unit

When the wakeup downcounting timer reaches 0, the wakeup flag is set to ‘1’. If this flag is selected as source for the RTC_ALARM output (OSEL[1:0] bits set to ‘11’ in RTC_CR register), the output will be set depending to the polarity selected and will remain set as long as the flag is not cleared.

Figure 15. Periodic wake-up routed to RTC_ALARM pinout



1.10 RTC security aspects

1.10.1 RTC Register write protection

To protect RTC registers against possible parasitic write accesses after reset, the RTC registers are automatically locked. They must be unlocked to update the current calendar time and date.

Writing to the RTC registers is enabled by programming a key in the Write protection register (RTC_WPR).

The following steps are required to unlock the write protection of the RTC register:

1. Write **0xCA** into the RTC_WPR register.
2. Write **0x53** into the RTC_WPR register.

Writing an incorrect key automatically reactivates the RTC register write access protection.

1.10.2 Enter/Exit initialization mode

The RTC can operate in two modes:

- *Initialization mode:* where the counters are stopped.
- *Free-running mode*: where the counters are running.

The calendar cannot be updated while the counters are running. The RTC must consequently be switched to *Initialization mode* before updating the time and date.

When operating in this mode, the counters are stopped. They start counting from the new value when the RTC enters *Free-running mode*.

The INIT bit of the RTC_ISR register enables you to switch from one mode to another, and the INITF bit can be used to check the RTC current mode.

The RTC must be in *Initialization mode* to program the time and date registers (RTC_TR and RTC_DR) and the prescalers register (RTC_PRER). This is done by setting the INIT bit and waiting until the RTC_ISR_INITF flag is set.

To return to *Free-running mode* and restart counting, the RTC must exit *Initialization mode*: This is done by resetting the INIT bit.

Only a power-on reset can reset the calendar. A system reset does not affected it but resets the shadow registers that are read by the application. They are updated again when the RSF bit is set. After a system reset, the application can check the INITS status flag in RTC_ISR register to verify if the calendar is already initialized. This flag is reset when the calendar year field is set to 0x00 (power-on reset value), meaning that the calendar must be initialized.

1.10.3 RTC clock synchronization

When the application reads the calendar, it accesses shadow registers that contain a copy of the real calendar time and date clocked by the RTC clock (RTCCLK). The RSF bit is set in the RTC_ISR register each time the calendar time and date shadow registers are updated with the real calendar value. The copy is performed every two RTCCLK cycles, synchronized with the system clock (SYSCLK). After a system reset or after exiting initialization mode, the application must wait for RSF to be set before reading the calendar shadow registers.

When the system is woken up from low power modes (SYSCLK was off), the application must first clear the RSF bit, and then wait until it is set again before reading the calendar registers. This ensures that the value read by the application is the current calendar value, and not the value before entering the used Low power mode.

2 RTC features summary

Table 12. Summary of RTC features by product family

RTC features		F-2 series	ULPM[1] density
Prescalers	Asynchronous	7 bits	Same
	Synchronous	13 bits	Same
Calendar	Time	– 12h/24h time format – Hours – Minutes – Seconds	Same
	Date	– Weekday – Date – Month – Year	Same
	Daylight operation	Add or subtract 1 hour to compensate for daylight savings time	Same
Wake-up unit		1 - Interrupt when wakeup event occurs 2 - Three possible configurations for: – Short period wakeup – Medium period wakeup – Long period wakeup	Same
Alarm	Alarms available	– Alarm A – Alarm B	Same
	Interrupts on Alarm event	One for each alarm	Same
	Time	– 12h/24h time format – Hours – Minutes – Seconds	Same
	Date	Date or Weekday	Same
	Masks	Masks for time and date	Same
RTC outputs	RTC_ALARM pin	Depending on configuration, it can deliver: – Alarm A – Alarm B – WakeUp status	Same
	RTC_CALIB pin	It can deliver a 512-Hz signal when RTCCLK = LSE	Same
Tamper detection		– Configurable input mapping – Configurable edge detection – Interrupt when Tamper event occurs	Same but no configurable input mapping

Table 12. Summary of RTC features by product family (continued)

RTC features	F-2 series	ULPM[1] density
Time-stamp	– Copy Time and Date fields – Configurable input mapping – Interrupt when Time stamp event occurs – Detect Time stamp overflow	Same but no configurable input mapping
Digital coarse calibration	– Recalibrate the RTC clock for crystal accuracy compensation. – 4 ppm resolution	Same
RTC backup registers	– Powered-on by VBAT when VDD is switched off – Reset on a tamper detection event or when the Flash readout protection is disabled.	– Powered-off when VDD is switched off (no VBAT) – Reset on a tamper detection event or when the Flash readout protection is disabled

1. Ultra Low Power Medium-density devices.

3 RTC firmware driver API

This driver provides a set of firmware functions to manage the following functionalities of the Real-Time Clock (RTC) peripheral:

- Initialization
- Calendar (Time and Date) configuration
- Alarms (Alarm A and Alarm B) configuration
- WakeUp Timer configuration
- Daylight Saving configuration
- Output pin Configuration
- Digital Calibration configuration
- TimeStamp configuration
- Tampers configuration
- Backup Data Register configuration
- RTC Tamper and TimeStamp Pins Selection and Output Type configuration
- Interrupts and flag management

For the STM32F2xx family, the RTC driver stm32f2xx_rtc.c/.h can be found in the directory: STM32F2xx_StdPeriph_Lib_vX.Y.Z\Libraries\STM32F2xx_StdPeriph_Driver.

For the STM32L1xx family, the RTC driver stm32l1xx_rtc.c/.h can be found in the directory: STM32L1xx_StdPeriph_Lib_vX.Y.Z\Libraries\STM32L1xx_StdPeriph_Driver.

These two drivers provide a fully compatible API making it easy to move from one product to another.

3.1 Start with RTC driver

Before using the RTC features:

- Enable the RTC domain access (see following note)
- Configure the RTC prescaler (Asynchronous and Synchronous) and RTC hour format using the **RTC_Init()** function.

Note: After a reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write access. To enable access to the RTC Domain and RTC registers:

- *Enable the Power Controller (PWR) APB1 interface clock using the RCC_APB1PeriphClockCmd() function.*
- *Enable access to the RTC domain using the PWR_BackupAccessCmd() function on STM32F2xx devices or PWR_RTCAccessCmd() function on STM32L1xx devices.*
- *Select the RTC clock source using the RCC_RTCCLKConfig() function.*
- *Enable RTC Clock using the RCC_RTCCLKCmd() function.*

3.1.1 Time and Date configuration

To configure the RTC Calendar (Time and Date) use the **RTC_SetTime**() and **RTC_SetDate**() functions.

To read the RTC Calendar, use the **RTC_GetTime**() and **RTC_GetDate**() functions.

Use the **RTC_DayLightSavingConfig**() function to add or sub one hour to the RTC Calendar.

3.1.2 Alarm configuration

To configure the RTC Alarm use the **RTC_SetAlarm**() function.

Enable the selected RTC Alarm using the **RTC_AlarmCmd**() function

To read the RTC Alarm, use the **RTC_GetAlarm**() function.

3.1.3 RTC Wakeup configuration

Configure the RTC Wakeup Clock source use the **RTC_WakeUpClockConfig**() function.

Configure the RTC WakeUp Counter using the **RTC_SetWakeUpCounter**() function.

Enable the RTC WakeUp using the **RTC_WakeUpCmd**() function.

To read the RTC WakeUp Counter register, use the **RTC_GetWakeUpCounter**() function.

3.1.4 Outputs configuration

The RTC has two different outputs:

- AFO_ALARM: used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal on RTC_AF1 pin, use the **RTC_OutputConfig**() function.
- AFO_CALIB: used to manage the RTC Clock divided by 64 (512Hz) signal.To output the RTC Clock on RTC_AF1 pin, use the **RTC_CalibOutputCmd**() function.

3.1.5 Coarse calibration configuration

Configure the RTC Coarse calibration value and the corresponding sign using the **RTC_CoarseCalibConfig()** function.

Enable the RTC Coarse calibration using the **RTC_CoarseCalibCmd()** function

3.1.6 TimeStamp configuration

Configure the RTC_AF1 trigger and enable the RTC TimeStamp using the **RTC_TimeStampCmd**() function.

Read the RTC TimeStamp Time and Date register using the **RTC_GetTimeStamp**() function.

The TAMPER1 alternate function can be mapped either to RTC_AF1(PC13) or RTC_AF2 (PI8) depending on the value of TAMP1INSEL bit in RTC_TAFCR register. You can use the **RTC_TamperPinSelection**() function to select the corresponding pin.

3.1.7 Tamper configuration

Configure the RTC Tamper trigger using the **RTC_TamperConfig**() function.

Enable the RTC Tamper using the RTC_TamperCmd() function.

The TIMESTAMP alternate function can be mapped to either RTC_AF1 or RTC_AF2 depending on the value of the TSINSEL bit in the RTC_TAFCR register. You can use the **RTC_TimeStampPinSelection**() function to select the corresponding pin.

3.1.8 Backup data registers configuration

To write to the RTC backup data registers, use the **RTC_WriteBackupRegister**() function.

To read the RTC backup data registers, use the **RTC_ReadBackupRegister**() function.

3.2 Function groups and description

The STM32 RTC driver can be divided into 11 function groups related to the functions embedded in the RTC peripheral.

1. RTC Initialization
2. RTC Time and date
3. RTC Alarms
4. RTC Wakeup timer
5. RTC Daylight saving
6. RTC output pin configuration
7. RTC Digital coarse calibration
8. RTC Timestamp
9. RTC Tamper
10. RTC Backup registers
11. RTC Tamper, timestamp pin selection
12. RTC Flags and IT

Table 13. RTC function groups

Group ID	Function name	Description	ULPM(1) density	F-2 series
1	**RTC Initialization Functions**			
	RTC_DeInit	Deinitializes the RTC registers to their default reset values.	Yes	Yes
	RTC_Init	Initializes the RTC registers according to the specified parameters in RTC_InitStruct <Hour format, Asynchronous predivisor, Asynchronous predivisor>.	Yes	Yes
	RTC_StructInit	Fills each RTC_InitStruct member with its default value.	Yes	Yes
	RTC_ITConfig	Enables or disables the specified RTC interrupts.	Yes	Yes
	RTC_RefClockCmd	Enables or disables the RTC reference clock detection	Yes	Yes
	RTC_EnterInitMode	Enters the RTC Initialization mode.	Yes	Yes
	RTC_ExitInitMode	Exits the RTC Initialization mode.	Yes	Yes
	RTC_WriteProtectionCmd	Enables or disables the RTC registers write protection.	Yes	Yes
	RTC_WaitForSynchro	Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized.	Yes	Yes
	RTC_TimeStructInit	Fills each RTC_TimeStruct member with its default value (Time = 00h:00min:00sec).	Yes	Yes
	RTC_DateStructInit	Fills each RTC_DateStruct member with its default value (Monday 01 January xx00).	Yes	Yes
	RTC_AlarmStructInit	Fills each RTC_AlarmStruct member with its default value (Time = 00h:00mn:00sec / Date = 1st day of the month/Mask = all fields are masked).	Yes	Yes
2	**RTC time and date functions**			
	RTC_SetTime	Sets the RTC current time < RTC hours, RTC minutes, RTC seconds, RTC 12-hour clock period (AM/PM)>.	Yes	Yes
	RTC_SetDate	Sets the current RTC date. < Calendar weekday, Calendar Month, Calendar date, Calendar year>.	Yes	Yes
	RTC_GetTime	Gets the current RTC time.	Yes	Yes
	RTC_GetDate	Gets the current RTC date.	Yes	Yes
3	**RTC alarms functions**			
	RTC_SetAlarm	Sets the RTC specified alarm configuration: "Alarm time fields, Alarm masks, Alarm date/Weekday selection, Alarm Date/Weekday value".	Yes	Yes
	RTC_GetAlarm	Gets the RTC specified alarm configuration.	Yes	Yes
	RTC_AlarmCmd	Enables or disables the RTC specified alarm.	Yes	Yes

Table 13. RTC function groups (continued)

Group ID	Function name	Description	ULPM[1] density	F-2 series
4	**RTC wakeup timer functions**			
	RTC_WakeUpClockConfig	Configures the RTC wakeup clock source.	Yes	Yes
	RTC_SetWakeUpCounter	Sets the RTC Wakeup counter value.	Yes	Yes
	RTC_GetWakeUpCounter	Returns the RTC Wakeup timer counter value.	Yes	Yes
	RTC_WakeUpCmd	Enables or disables the RTC Wakeup timer.	Yes	Yes
5	**RTC daylight saving functions**			
	RTC_DayLightSavingConfig	Adds or subtracts one hour from the current time depending on the daylight saving parameter.	Yes	Yes
	RTC_GetStoreOperation	Returns the daylight saving stored operation.	Yes	Yes
6	***RTC output pin configuration function***			
	RTC_OutputConfig	Configures the RTC output for the output pinout (RTC_ALARM pin)	Yes	Yes
7	**RTC digital coarse calibration functions**			
	RTC_DigitalCalibConfig	Configures the coarse Calibration Settings	Yes	Yes
	RTC_DigitalCalibCmd	Enables or disables the digital calibration process	Yes	Yes
	RTC_CalibOutputCmd	Enables or disables the connection of the RTCCLK/PREDIV_A[6:0] clock to be output through the relative pinout (RTC_CALIB pin)	Yes	Yes
8	**RTC timestamp functions**			
	RTC_TimeStampCmd	Enables or Disables the RTC TimeStamp functionality with the specified time stamp pin stimulating edge	Yes	Yes
	RTC_GetTimeStamp	Get the RTC TimeStamp value and masks	Yes	Yes
9	**RTC tamper functions**			
	RTC_TamperTriggerConfig	Configures the Tamper Edge Trigger	Yes	Yes
	RTC_TamperCmd	Enables or disables the Tamper detection	Yes	Yes
10	**RTC backup registers functions**			
	RTC_WriteBackupRegister	Writes a data in a specified RTC Backup data register	Yes	Yes
	RTC_ReadBackupRegister	Reads data from the specified RTC Backup data register	Yes	Yes
11	**RTC tamper, timestamp pins selection functions**			
	RTC_OutputTypeConfig	Configures the RTC Output pin mode (OpenDrain / PushPull)	Yes	Yes
	RTC_TimeStampPinSelection	Selects the RTC TimeStamp pin	No	Yes
	RTC_TamperPinSelection	Selects the RTC Tamper pin	No	Yes

Table 13. RTC function groups (continued)

Group ID	Function name	Description	ULPM[(1)] density	F-2 series
12	**RTC flags and interrupts functions**			
	RTC_GetFlagStatus	Checks whether the specified RTC flag is set or not	Yes	Yes
	RTC_ClearFlag	Clears the RTC pending flags	Yes	Yes
	RTC_GetITStatus	Checks whether the specified RTC interrupt has occurred or not	Yes	Yes
	RTC_ClearITPendingBit	Clears the RTC interrupt pending bits	Yes	Yes

1. Ultra Low Power Medium-density devices.

4 Application examples

The RTC firmware driver is provided with a set of examples so you can quickly become familiar with the RTC peripheral.

This section provides descriptions of examples that are delivered within the STM32F2xx and STM32L1xx Standard Peripherals Libraries available from http://www.st.com/.

For the STM32F2xx family, the examples can be found in the following directory: ***STM32F2xx_StdPeriph_Lib_vX.Y.Z\Project\STM32F2xx_StdPeriph_Examples\RTC***

For the STM32L1xx family, the examples can be found in the following directory: ***STM32L1xx_StdPeriph_Lib_vX.Y.Z\Project\STM32L1xx_StdPeriph_Examples\RTC***

Table 14. Example descriptions

Example	Description	Covered features
RTC Hardware Calendar (1)	This example describes how to use the RTC peripheral calendar features: seconds, minutes, hours (12 or 24 format), day, date, month, and year. As an application example, it demonstrates how to setup the RTC peripheral, in terms of prescaler and interrupts to be used to keep time and to generate alarm interrupt.	– Hardware Calendar – Alarm (interrupt) – Prescalers – RTC Backup registers
RTC Backup domain (2)	This example demonstrates and explains how to use the peripherals available on Backup Domain. These peripherals are the RCC BDCR register containing the LSE oscillator configuration and the RTC Clock enable/disable bits. This example embeds the RTC peripheral and its associated Backup Data registers, and the Backup SRAM (4KB) with its low power regulator (which enables it to preserve its contents when the product is powered by VBAT pin). As an application example, it demonstrates how to setup the RTC hardware calendar, and read/write operations for RTC Backup Data registers and BKPSRAM (Backup SRAM).	– RTC Backup registers – Backup SRAM – Low power Regulator for Backup SRAM – Hardware Calendar – Wakeup (interrupt)
Auto calibration using LSI	This example demonstrates and explains how to use the LSI clock source auto calibration to get a precise RTC clock. The Low Speed Internal (LSI) clock is used as RTC clock source. The RTC WakeUp is configured to generate an interrupt each 1s. The WakeUp counter is clocked by the RTC CK_SPRE signal (1Hz) and its counter is set to zero.	– Prescalers – RTC Backup registers – Hardware Calendar – Wakeup (interrupt)
Tamper detection	This example shows how to write/read data to/from RTC Backup data registers and demonstrates the Tamper detection feature.It configures the RTC_AF1 pin Tamper to be falling edge, and enables the Tamper interrupt.On applying a low level on the RTC_AF1 pin, the RTC backup data registers are reset and the Tamper interrupt is generated.	– Tamper (interrupt) – RTC Backup registers
Time Stamp	This example describes how to use the RTC peripheral and the Time Stamp feature. It configures the RTC_AF1 pin TimeStamp to be falling edge and enables the TimeStamp detection.On applying a low level on the RTC_AF1 pin, the calendar is saved in the time-stamp registers thanks to the timestamp event detection.	– Time stamp (interrupt) – Prescalers – Wakeup (interrupt) – Hardware Calendar – RTC Backup registers

1. For Ultra Low Power Medium-density example, Alarm feature is not used.
2. This example is delivered only with F2 - series FW examples.

5 Revision history

Table 15. Document revision history

Date	Revision	Changes
20-May-2011	1	Initial release