AN4759
Application note

Using the hardware real-time clock (RTC) and the tamper management unit (TAMP) with STM32 microcontrollers

Introduction

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

In this document, the STM32 microcontroller terminology applies to the products listed in *Table 1*.

This application note describes the features of the RTC and how to configure it to implement several use cases: calendar, alarm, wakeup, timestamp, tamper detection, calibration.

Table 4 and *Table 5* specify the RTC features available depending on the STM32 microcontroller used and its RTC type according to *Table 2* and *Table 3*.

Depending on its RTC type, the product documentation may refer to an independent tamper management unit (TAMP) that is also detailed in this application note in the sections concerning the tamper detection.

Software examples dedicated to this application note are then detailed to show how to use the RTC in the low-power modes and how to ensure the tampering detection and timestamp while the main supply is switched off and the MCU is supplied by an alternate battery. Four other examples are also presented to illustrate the following features: smooth calibration, synchronization, reference clock detection and internal tamper.

These dedicated softwares are provided through the X-CUBE-RTC embedded software package delivered with this application note. It contains the source code of these examples and all the embedded software modules required to run the examples.

Table 1. Applicable products

Type	Product series and part number
Microcontrollers	STM32F0 Series, STM32F2 Series, STM32F3 Series, STM32F4 Series, STM32F7 Series, STM32L0 Series, STM32L1 Series, STM32L4 Series, STM32L4+ Series, STM32L5 Series, STM32G0 Series, STM32G4 Series, STM32H7 Series, STM32WB Series, STM32WL Series, X-CUBE-RTC

Contents

List of tables

List of figures

1 Overview of the STM32 MCUs advanced RTC

The real-time clock (RTC) is embedded in STM32 Arm®(a)-based MCUs.

The RTC is used to provide a full-featured calendar, alarm, periodic wakeup unit, digital calibration, synchronization, timestamp, and an advanced tamper detection.

The RTC features and their implementation can be significantly different (regarding the registers map for example) depending on the product used.

Thus, two RTC types are distinguished and characterized in *Table 2*.

If the RTC type affects the information presented in this document, the two cases are treated and referred as concerning the RTC2 type and RTC3 type. If no particular precision is mentioned, the information is correct whatever the type.

Table 2. RTC/TAMP types versus features

RTC/TAMP features			RTC2	RTC3
RTC clock source (LSE, LSI, HSE with prescaler)			X	X
Binary mode			-	X
Mixed mode (BCD and binary)			-	X
Prescalers	Asynchronous		X	X
	Synchronous		X	X
Calendar	Time	12/24 format	X	X
		hour, minutes,	X	X
		Sub-seconds	X	X
	Date		X	X
	Daylight operation		X	X
	Bypass the shadow registers		X	X
	Power optimization mode		-	X
Alarm	Alarms available	Alarm A	X	X
		Alarm B	X	X
	Time	12/24 format	X	X
		hour, minutes,	X	X
		Sub-seconds	X	X
	Date or week day		X	X
	Binary mode alarm		-	X

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Table 2. RTC/TAMP types versus features (continued)

RTC/TAMP features			RTC2	RTC3
Tamper detection	Tamper effects	Backup registers erasing (depending on devices, other resources can also be erased)	X	X
		Interruption	X	X
		Trigger for low-power timer	X	X
	Configurable edge detection		X	X
	Configurable level detection (filtering, sampling and precharge configuration, internal pull-up),		X	X
	Internal tamper events		-	X
	External tamper inputs		X	X
	VBAT mode pins		X	X
	NOERASE mode		X	X
	Active tamper		-	X
	Monotonic counter		-	X
	Secure protection modes		-	X
	Privilege protection mode		-	X
Timestamp	Configurable input mapping		X	X
	Time	Hours, minutes	X	X
		Sub-seconds	X	X
	Date (day, month)		X	X
	Timestamp on tamper detection event		X	X
	Timestamp on switch to VBAT mode		X	X
Wakeup unit	Clock source available		X	X
	Hardware automatic flag clearance		X	X
RTC outputs	TAMPALRM	Alarm event	X	X
		Wakeup event	X	X
		Tamper event	-	X
	CALIB	512 Hz	X	X
		1 Hz	X	X
Smooth digital calibration	Smooth calibration		X	X
	Power optimization mode		-	X

Table 2. RTC/TAMP types versus features (continued)

RTC/TAMP features		RTC2	RTC3
Synchronizing the RTC		X	X
Reference clock detection		X	X
Backup registers	Reset on a tamper detection	X	X
	Reset when Flash readout protection is disabled	X	X
RTC secure protection mode		-	X
RTC privilege protection mode		-	X

The RTC acts as an independent binary-coded decimal (BCD) timer/counter. For RTC3 a binary and a mixed (both binary and BCD) mode are available.

Note: A feature can be available for a RTC type but not implemented on every products. Refer to *Table 4* and *Table 5 for the complete list of features available on each product.*

Table 3 specifies the two RTC types implemented for each product.

Table 3. RTC type versus STM32 products

RTC type	STM32 product series and part number
RTC2	STM32F0 Series, STM32F2 Series, STM32F3 Series, STM32F4 Series, STM32F7 Series, STM32L0 Series, STM32L1 Series, STM32WB Series, STM32L4R5/Q5 line, STM32L4R7/S7 line, STM32L4R9/S9 line, STM32L43xxx/L44xxx/L45xxx/46xxx/47xxx/48xxx, STM32H74xxx/H75xxx
RTC3	STM32L5 Series, STM32G0 Series, STM32G4 Series, STM32WL Series, STM32L4P5/Q5 line, STM32H7A3/7B3 line, STM32L41xxx/L42xxx

2 Advanced RTC features

Table 4 and *Table 5* summarize the RTC features available on each product.

Table 4. Advanced RTC2 features

RTC features			STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series except STM32 L41/42xxx / STM32L4+ Series except STM32L4P 5/Q5 line	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series	STM32WB Series	STM32 H74xxx/ H75xxx
RTC clock source (LSE, LSI, HSE with prescaler)			X	X	X	X	X	X	X	X	X	X	X
Prescalers	Asynchronous		X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)
	Synchronous		X (15 bits)	X (15 bits)	X (13 bits)	X (1 5 bits)	X (15 bits)	X (13 bits)	X (15 bits)	X (15 bits)	X (15 bits)	X (1 5 bits)	X (15 bits)
Calendar	Time	12/24 format	X	X	X	X	X	X	X	X	X	X	X
		Hour, minutes, seconds	X	X	X	X	X	X	X	X	X	X	X
		Sub-second	X	X	Not available	X	X	Not available	X	X	X	X	X
	Date		X	X	X	X	X	X	X	X	X	X	X
	Daylight operation		X	X	X	X	X	X	X	X	X	X	X
	Bypass the shadow registers		X	X	Not available	X	X	Not available	X	X	X	X	X
	VBAT mode		Not available	Not available	Not available	X	X	X	X	X	X	X	X

Table 4. Advanced RTC2 features (continued)

RTC features			STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series except STM32 L41/42xxx / STM32L4+ Series except STM32L4P 5/Q5 line	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series	STM32WB Series	STM32 H74xxx/ H75xxx
Alarm	Alarms available	Alarm A	X	X	X	X	X	X	X	X	X	X	X
		Alarm B	X	X	X	X	Not available	X	X	X	X	X	X
	Time	12/24 format	X	X	X	X	X	X	X	X	X	X	X
		Hour, minutes and seconds	X	X	X	X	X	X	X	X	X	X	X
		Sub-second	X	X	Not available	X	X	Not available	X	X	X	X	X
	Date or week day		X	X	X	X	X	X	X	X	X	X	X

Table 4. Advanced RTC2 features (continued)

RTC features		STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series except STM32 L41/42xxx / STM32L4+ Series except STM32L4P 5/Q5 line	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series	STM32WB Series	STM32 H74xxx/ H75xxx
Tamper detection	Configurable input mapping	X	Not available	Not available	X	X	X	X	X	X	X	X
	Configurable edge detection	X	X	X	X	X	X	X	X	X	X	X
	Configurable Level detection (filtering, sampling and precharge configuration on tamper input)	X	X	Not available	X	X	Not available	X	X	X	X	X
	Number of tamper inputs	3 inputs / 3 events	3 inputs / 3 events	1 input / 1 event	3 inputs / 3 events	2 inputs 3 inputs (for STM32F07x and STM32F09x)	2 inputs / 1 event	2 inputs / 2 events	2 inputs / 2 events	3 inputs/ 3 events	3 inputs / 3 events	3 inputs / 3 events
	VBAT mode pins	Not available	Not available	Not available	3 inputs	1 input	1 input	1 input	2 inputs	2 inputs	3 inputs	3 inputs

Table 4. Advanced RTC2 features (continued)

RTC features			STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series except STM32 L41/42xxx / STM32L4+ Series except STM32L4P 5/Q5 line	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series	STM32WB Series	STM32 H74xxx/ H75xxx
Timestamp	Configurable input mapping		X(1)	Not available	Not available	X(1)	X	X	X	X	X	X(1)	X
	Time	Hours, minutes and seconds	X	X	X	X	X	X	X	X	X	X	X
		Sub-seconds	X	X	Not available	X	X	Not available	X	X	X	X	X
	Date (day, month)		X	X	X	X	X	X	X	X	X	X	X
	Timestamp on tamper detection event		X	X	X	X	X	X	X	X	X	X	X
	Timestamp on switch to VBAT mode		Not available	Not available	Not available	X	Not available	Not available	Not available	Not available	X	X	X
RTC outputs	RTC_ ALARM	Alarm event	X	X	X	X	X	X	X	X	X	X	X
		Wakeup event	X	X	X	X	X	X	X	X	X	X	X
	RTC_ CALIB	512 Hz	X	X	X	X	X	X	X	X	X	X	X
		1 Hz	X	X	Not available	X	X	Not available	X	X	X	X	X
RTC calibration	Coarse calibration		Not available (2)	X	X	Not available (2)	Not available (2)	X	X	X	Not available (2)	Not available (2)	Not available (2)
	Smooth calibration		X	X	Not available	X	X	Not available	X	X	X	X	X
Synchronizing the RTC			X	X	Not available	X	X	Not available	X	X	X	X	X

Table 4. Advanced RTC2 features (continued)

RTC features		STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series except STM32 L41/42xxx / STM32L4+ Series except STM32L4P 5/Q5 line	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series	STM32WB Series	STM32 H74xxx/ H75xxx
Reference clock detection		X	X	X	X	X	X	X	X	X	X	X
Backup registers	Powered-on VBAT	Not available	Not available	Not available	X	X	X	X	X	X	X	X
	Reset on a tamper detection	X	X	X	X	X	X	X	X	X	X	X
	Reset when Flash readout protection is disabled	X	X	X	X	X	Not available	X	Not available	X	X	X
	Number of backup registers	5	32(3)	20(4)	20 32 (for STM32L4Rx/Sx lines)	5	20	16	20	32	20	32

1. Thanks to timestamp on tamper event.
2. Obsolete, replaced by smooth calibration.
3. Cat. 3/4/5/6.
4. Cat. 1 and Cat. 2.

Table 5. Advanced RTC3 features

RTC features			STM32G0x0 Value line	STM32G0x1 line	STM32G4 Series	STM32L41xxx/ L42xxx STM32L4P5/Q5 line	STM32L5 Series	STM32WL Series	STM32H7A3/ 7B3 line
RTC clock source (LSE, LSI, HSE with prescaler)			X	X	X	X	X	X	X
Binary mode			Not available	Not available	Not available	Not available	Not available	X	Not available
Mixed mode (BCD and binary)			Not available	Not available	Not available	Not available	Not available	X	Not available
Prescalers	Asynchronous		X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)
	Synchronous		X (15 bits)	X (15 bits)	X (15 bits)	X (15 bits)	X (15 bits)	X (15 bits)	X (15 bits)
Calendar	Time	12/24 format	X	X	X	X	X	X	X
		hour, minutes, seconds	X	X	X	X	X	X	X
		Sub-seconds	X	X	X	X	X	X	X
	Date		X	X	X	X	X	X	X
	Daylight operation		X	X	X	X	X	X	X
	Bypass the shadow registers		X	X	X	X	X	X	X
	Power optimization mode		Not available	Not available	Not available	X	X	X	Not available
Alarm	Alarms available	Alarm A	X	X	X	X	X	X	X
		Alarm B	X	X	X	X	X	X	X
	Time	12/24 format	X	X	X	X	X	X	X
		hour, minutes, seconds	X	X	X	X	X	X	X
		Sub-seconds	X	X	X	X	X	X	X
	Date or week day		X	X	X	X	X	X	X
	Binary mode alarm		Not available	Not available	Not available	Not available	Not available	X	Not available

Table 5. Advanced RTC3 features (continued)

RTC features			STM32G0x0 Value line	STM32G0x1 line	STM32G4 Series	STM32L41xxx/ L42xxx STM32L4P5/Q5 line	STM32L5 Series	STM32WL Series	STM32H7A3/ 7B3 line
Tamper detection	Tamper reactions	Backup registers erasing	X	X	X	X	X (erasing part of SRAM is also possible)	X (erasing part of SRAM is also possible)	X (erasing part of SRAM is also possible)
		Interruption	X	X	X	X	X	X	X
		Trigger for low-power timer	X	X	X	X	X	X	X
	Configurable edge detection		X	X	X	X	X	X	X
	Configurable level detection (filtering, sampling and precharge configuration, internal pull-up)		X	X	X	X	X	X	X
	Number of internal tamper events		4	4	4	Not available	5	4	7
	Number of external tamper inputs		2	2	3	2; 3 (for STM32L4P5 /Q5 line)	8	3	3
	VBAT mode pins		Not available	Not available	Not available	RTC_TAMP1/2/3, RTC_TS and RTC_OUT (for STM32L41xxx/ L42xxx); RTC_TS and RTC_OUT1 (for STM32L4P5/LQ5)	TAMP_IN1,2,3 and TAMP_OUT2	RTC_TS and RTC_OUT1	RTC_TS and RTC_OUT1/2
	NOERASE mode		Not available	Not available	Not available	X	Not available	Not available	Not available

Table 5. Advanced RTC3 features (continued)

RTC features			STM32G0x0 Value line	STM32G0x1 line	STM32G4 Series	STM32L41xxx/ L42xxx STM32L4P5/Q5 line	STM32L5 Series	STM32WL Series	STM32H7A3/ 7B3 line
Tamper detection (continued)	Active tamper		Not available	Not available	Not available	Not available	X	Not available	X
	Monotonic counter		Not available	Not available	Not available	Not available	X	X	X
	Low-power modes' effect on tamper unit	Sleep	No effect	No effect	No effect	No effect	No effect	No effect	No effect
		Stop	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI
		Standby	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI
		Shutdown	No effect except for filtered level detection and active tamper if clock source is not LSE	No effect except for filtered level detection and active tamper if clock source is not LSE	No effect except for filtered level detection and active tamper if clock source is not LSE	No effect except for filtered level detection and active tamper if clock source is not LSE	No effect except for filtered level detection and active tamper if clock source is not LSE	No effect except for filtered level detection and active tamper if clock source is not LSE	No effect except for filtered level detection and active tamper if clock source is not LSE
	Secure protection modes		Not available	Not available	Not available	Not available	X	Not available	Not available
	Privilege protection mode		Not available	Not available	Not available	Not available	X	Not available	Not available

Table 5. Advanced RTC3 features (continued)

RTC features			STM32G0x0 Value line	STM32G0x1 line	STM32G4 Series	STM32L41xxx/ L42xxx STM32L4P5/Q5 line	STM32L5 Series	STM32WL Series	STM32H7A3/ 7B3 line
Time stamp	Configurable input mapping		X	X	X	X	X	X	X
	Time	Hours, minutes and seconds	X	X	X	X	X	X	X
		Sub-seconds	X	X	X	X	X	X	X
	Date (day, month)		X	X	X	X	X	X	X
	Time-stamp on tamper detection event		X	X	X	X	X	X	X
	Time-stamp on switch to VBAT mode		X	X	X	X	X	X	X
Wakeup unit	Clock source available		RTC clock divided by 2,4,8,16 or RTC synchronous prescaler output	RTC clock divided by 2,4,8,16 or RTC synchronous prescaler output	RTC clock divided by 2,4,8,16 or RTC synchronous prescaler output	RTC clock divided by 2,4,8,16 or RTC synchronous prescaler output	RTC clock divided by 2,4,8,16 or RTC synchronous prescaler output	RTC clock divided by 2,4,8,16 or RTC synchronous prescaler output	RTC clock divided by 2,4,8,16 or RTC synchronous prescaler output
	Hardware automatic flag clearance		Not available	Not available	Not available	X	X	X	Not available
RTC outputs	Number of outputs		2	2	2	2	2	2	2
	TAMPALRM	Alarm event	X	X	X	X	X	X	X
		Wakeup event	X	X	X	X	X	X	X
		Tamper event	X	X	X	X	X	X	X
	CALIB	512 Hz	X	X	X	X	X	X	X
		1 Hz	X	X	X	X	X	X	X

Table 5. Advanced RTC3 features (continued)

RTC features		STM32G0x0 Value line	STM32G0x1 line	STM32G4 Series	STM32L41xxx/ L42xxx STM32L4P5/Q5 line	STM32L5 Series	STM32WL Series	STM32H7A3/ 7B3 line
Smooth digital calibration	Smooth calibration	X	X	X	X	X	X	X
	Ultra-low-power mode	Not available	Not available	Not available	X	X	X	Not available
Synchronizing the RTC		X	X	X	X	X	X	X
Reference clock detection		X	X	X	X	X	X	X
Backup registers	Powered on VBAT	X	X	X	X	X	X	X
	Reset on a tamper detection	X	X	X	X	X	X	X
	Reset when Flash readout protection is disabled	X	X	X	X	X	X	X
	Number of backup registers	5 32-bit	5 32-bit	32 or 16 32-bit (depends on the device's category)	32 32-bit	32 32-bit	20 32-bit	32 32-bit
Low-power modes' effect on RTC	Sleep	No effect	No effect	No effect	No effect	No effect	No effect	No effect
	Stop	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI
	Standby	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI	Active if RTC clocked by LSE or LSI
	Shutdown	Active if RTC clocked by LSE	Active if RTC clocked by LSE	Active if RTC clocked by LSE	Active if RTC clocked by LSE	Active if RTC clocked by LSE	Active if RTC clocked by LSE	Active if RTC clocked by LSE
RTC secure protection mode		Not available	Not available	Not available	Not available	X	Not available	Not available
RTC privilege protection mode		Not available	Not available	Not available	Not available	X	Not available	Not available

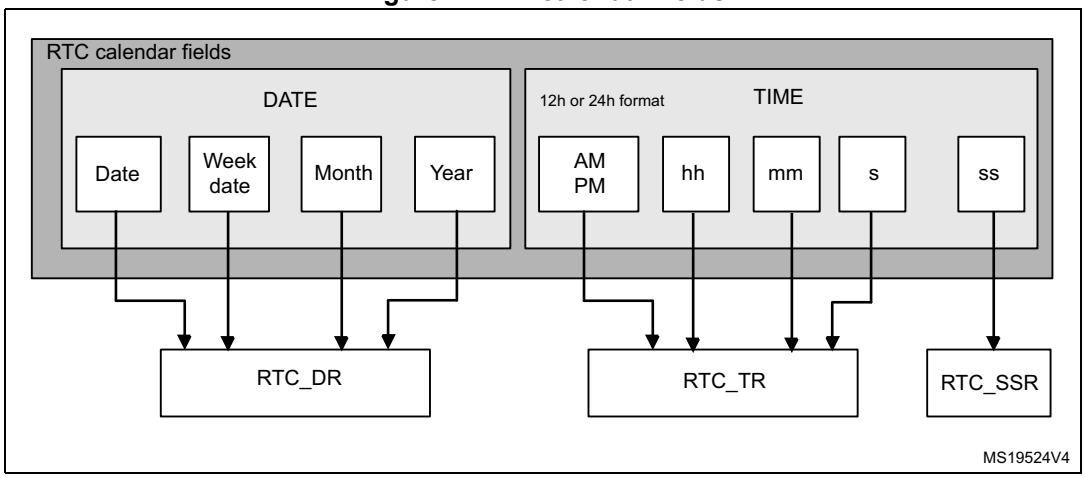
2.1 RTC calendar

A calendar keeps track of the time (hours, minutes and seconds) and date (day, week, month, year).

The RTC calendar offers the following features to easily configure and display the calendar data fields:

- Calendar with:
  - Sub-seconds (not programmable)
  - 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. RTC_DR, RTC_TR are RTC date and time registers.
2. The sub-second field (RTC_SSR) is the value of the synchronous prescaler counter. This field is not writable.

2.1.1 Software calendar

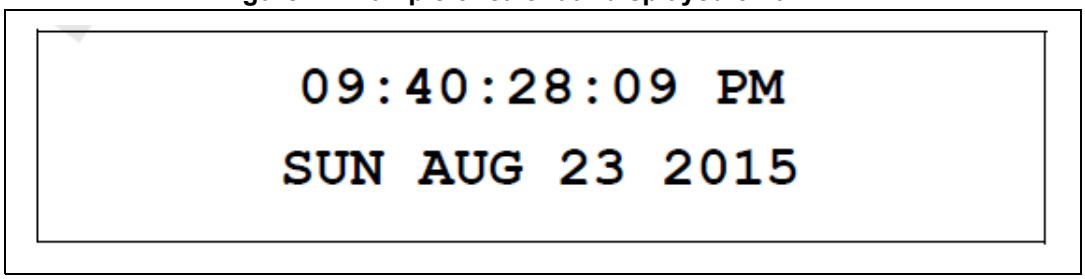
A software calendar is a software counter (usually 32-bit long) that represents the number of seconds. The software routines convert the counter value to hours, minutes, day of the month, day of the week, month and year. This data can be converted to BCD format and displayed on a standard LCD. Conversion routines use a significant program memory space and are CPU-time consuming, which may be critical in certain real-time applications.

2.1.2 RTC hardware calendar

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

The STM32 RTC calendar is provided in BCD format. This avoids binary to BCD software conversion routines, that save system resources.

Figure 2. Example of calendar displayed on an LCD



2.1.3 Initializing the calendar

Table 6 describes the steps required to correctly configure the calendar time and date.

Table 6. 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 (RTC2)/RTC_ICSR (RTC3) register	The calendar counter is stopped to allow its update
3	Wait for the confirmation of Initialization mode (clock synchronization)	Poll INITF bit of in RTC_ISR (RTC2)/RTC_ICSR (RTC3) until it is set	For RTC2: It takes around 2 RTCCLK clock cycles due to clock synchronization. For RTC3: If LPCAL=0 INITF is set around 2 RTCLK cycles after INIT bit is set, if LPCAL=1 INITF is set up to 2 ck_apre cycles after INIT bit is set.
4	Program the prescaler values if needed	RTC_PRER register: Write first the synchronous value and then write the asynchronous value. For RTC3, program also BIN and BCDU in the RTC_ICSR register if in binary or mixed mode.	By default (in BCD mode for RTC3), the RTC_PRER prescalers register are initialized to provide 1 Hz to the Calendar unit when RTCCLK = 32768 Hz
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

Table 6. Steps to initialize the calendar (continued)

Step	What to do	How to do it	Comments
7	Exit Initialization mode	Clear the INIT bit in RTC_ISR (RTC2)/RTC_ICSR (RTC3) register	For RTC2: The current calendar counter is automatically loaded and the counting restarts after 4 RTCCLK clock cycles. For RTC3: If LPCAL=0 the counting restarts after 4 RTCCLK clock cycles, if LPCAL=1 the counting restarts after up to 2 RTCCLK + 1 ck_apre.
8	Enable the RTC Registers Write Protection	Write 0xFF into the RTC_WPR register	RTC Registers can no longer be modified

2.1.4 RTC clock configuration

RTC clock source

The RTC calendar can be driven by one of three possible clock sources LSE, LSI or HSE. If HSE is chosen, a prescaler must be selected. The user can refer to the product reference manual to know its possible values and configurations.

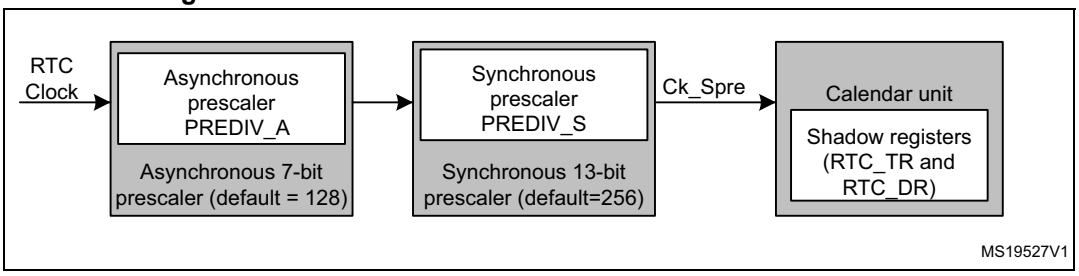
Moreover, the choice of the RTC clock source is done thanks to the RTCSEL[1:0] bits stored in a RCC register. This register depends on the product and this information can also be found in the product's reference manual.

How to adjust the RTC calendar clock

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

In case of the RTC3, the BCD or mixed mode is considered. In binary mode (available only on RTC3) the calendar is not functional.

Figure 3. Prescalers from RTC clock source to calendar unit



Note: It is important to know that the calendar unit is linked to ck_spre but that it belongs to the ck_apre (asynchronous prescaler) clock domain. Thus, the choice of ck_apre can help to optimize power consumption.

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 32767

Table 7 shows several ways to obtain the calendar clock (ck_spre) = 1 Hz.

Table 7. Calendar clock equal to 1 Hz with different clock sources(1)

RTCCLK Clock source	Prescalers		ck_spre
	PREDIV_A[6:0]	PREDIV_S[14:0]	
HSE_RTC = 1 MHz	124 (div 125)	7999 (div 8000)	1 Hz
LSE = 32.768 kHz	127 (div 128)	255 (div 256)	1 Hz
LSI = 32 kHz	127 (div 128)	249 (div 250)	1 Hz
LSI = 37 kHz	124 (div 125)	295 (div 296)	1 Hz
LSI = 40 kHz	127 (div 128)	311 (div 312)	1 Hz

1. Other PREDIV_A[6:0]/PREDIV_S[14:0] values are possible. The user must always prefer the combination where PREDIV_A[6:0] is the highest for the needed accuracy and for lower consumption.

2.1.5 Calendar firmware examples

The RTC peripheral comes with a set of example projects so that the user can quickly become familiar with the RTC. Refer to the STM32Cube MCU Package provided with the product for a complete projects list.

For example, the user can find the following projects concerning calendar:

- For the NUCLEO-L412RB-P equipped with an RTC3:
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_Calendar
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples_LL\RTC\RTC_Calendar
- For the P-NUCLEO-WB55 equipped with an RTC3:
  - STM32Cube_FW_WB_Vx.y.z\Projects\P-NUCLEO-WB55.Nucleo\Examples_LL\RTC\RTC_Calendar_Init

The user can check if these examples are available in his product STM32Cube MCU Package or adapt them for his product.

2.2 Binary and mixed modes (RTC3 only)

A particularity for the RTC3 is the availability of a binary mode. In this mode the time and date BCD calendar is disable but the sub second register (SSR) of the RTC is extended to 32-bit instead of 16-bit in normal mode and is used as a binary down counter.

This feature allows to have a 32-bit counter in low-power mode (the modes' compatibilities are the same than for the other features of the RTC) and to avoid the BCD to binary conversion in case it is required by an application.

The binary mode implementation is simple. After initialization (INIT bit set to 1 and wait for INITF to be set, plus choice of the asynchronous prescaler value to clock the binary counter), the user sets the BIN bits in the RTC_ICSR register to 0b01 then the RTC_SSR register is initialized to 0xFFFF FFFF. After exiting initialization (INIT bit cleared), the counter start to count down from RTC_SSR initial value.

Also, the counter is clocked by the RTC asynchronous prescaler ouput. When it reaches 0, it is reloaded with 0xFFFF FFFF.

The mixed mode also offered by RTC3 allows both the 32-bit binary down-counter and the BCD calendar. In this mode, the user can choose when the calendar is incremented by 1 second with the BCDU[2:0] bits in RTC_ICSR. These bits code how many least significant bits from SSR (sub second segister) need to be at 0 for the calendar to be incremented by 1 second.

2.3 RTC alarms

2.3.1 RTC alarm configuration

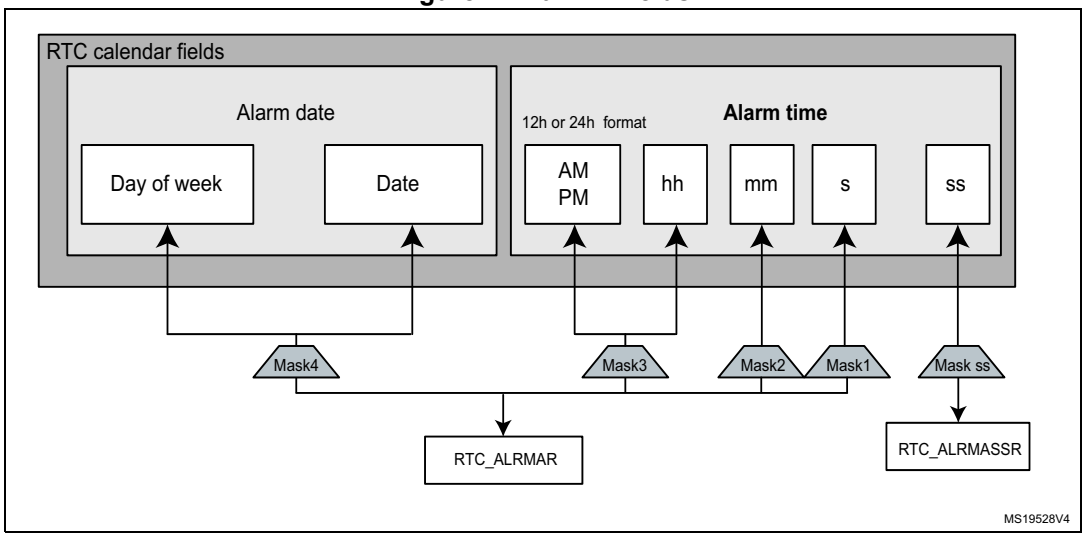
The RTC embeds two alarms, alarm A and alarm B, which are similar. An alarm can be generated at a given time or/and date programmed by the user.

The 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: sub-second (this is discussed later), seconds, minutes, hours and date fields can be independently selected or masked to provide a rich combination of alarms.
- Ability to wake up 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 4. Alarm A fields



1. RTC_ALRMAR is an RTC register. The same fields are also available for the RTC_ALRMBR register.
2. RTC_ALRMASSR is an RTC register. The same field is also available for the RTC_ALRMBSSR register.
3. Maskx are the bits in the RTC_ALRMAR register that enable/disable the RTC_ALARM fields used for alarm A. For more details, refer to *Table 9*.
4. Mask ss are the bits in the RTC_ALRMASSR register.

The time configuration bit-fields of the alarm configuration register are mapped into the same bit offsets as those on the RTC_TR time register for easier software manipulation. When the RTC time counter reaches the value programmed in the alarm register, a flag is set to indicate that an alarm event occurred (ALRAF or ALRBF in RTC_ISR).

The RTC alarm can be configured by hardware to generate different types of alarms. For more details, refer to *Table 9*.

Programming the alarm

Table 8 describes the steps required to configure alarm A.

Table 8. 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(2).	-
3(3)	Check that the RTC_ALRMAR register can be accessed	Poll ALRAWF(4) bit until it is set in RTC_ISR.	It takes around 2 RTCCLK clock cycles due to clock synchronization.
4	Configure the alarm	Configure RTC_ALRMAR(5) register.	The alarm hour format must be the same(6) as the RTC Calendar in RTC_ALRMAR.
5	Re-enable alarm A	Set ALRAE(7) 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. RTC alarm registers can only be written when the corresponding RTC alarm is disabled or during RTC Initialization mode.
3. Only required for RTC2.
4. Respectively ALRBWF bit for alarm B. ALRAWF and ALRBWF does not exist on RTC3.
5. Respectively RTC_ALRMBR register for alarm B.
6. As an example, if the alarm is configured to occur at 3:00:00 PM, the alarm does 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.
7. Respectively ALRBE bit for alarm B.

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 9 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), configure the RTC_ALRMAR register (resp. RTC_ALRMBR register) with the MSKx bits set to 0b0000. When the WDSEL = 0, all the cases are similar, except that the Alarm Mask field compares with the day number and not the day of the week, and MSKx bits must be set to 0b0000.

Table 9. Alarm combinations

MSK3	MSK2	MSK1	MSK0	Alarm behavior
0	0	0	0	**All the fields are used in the alarm comparison:** The alarm occurs at 23:15:07, each Monday.
0	0	0	1	**The seconds do not matter in the alarm comparison** The alarm occurs every second of 23:15, each Monday.
0	0	1	0	**The minutes do not matter in the alarm comparison** The alarm occurs at the 7th second of every minute of 23:XX, each Monday.
0	0	1	1	**The minutes and seconds do not matter in the alarm comparison**
0	1	0	0	**The hours do not matter in the alarm comparison**
0	1	0	1	**The hours and seconds do not matter in the alarm comparison**
0	1	1	0	**The hours and minutes do not matter in the alarm comparison**
0	1	1	1	**The hours, minutes and seconds do not matter in the alarm comparison** The alarm is set every second, each Monday, during the whole day.
1	0	0	0	**The week day (or date, if selected) do not matter in the alarm comparison** The alarm occurs all the days at 23:15:07.
1	0	0	1	**The week day and seconds do not matter in the alarm comparison**
1	0	1	0	**The week day and minutes do not matter in the alarm comparison**
1	0	1	1	**The week day, minutes and seconds do not matter in the alarm comparison**
1	1	0	0	**The week day and hours do not matter in the alarm comparison**
1	1	0	1	**The week day, hours and seconds do not matter in the alarm comparison**
1	1	1	0	**The week day, hours and minutes do not matter in the alarm comparison**
1	1	1	1	**The alarm occurs every second**

Caution: If the second 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 a correct behavior.

2.3.2 Alarm sub-second configuration

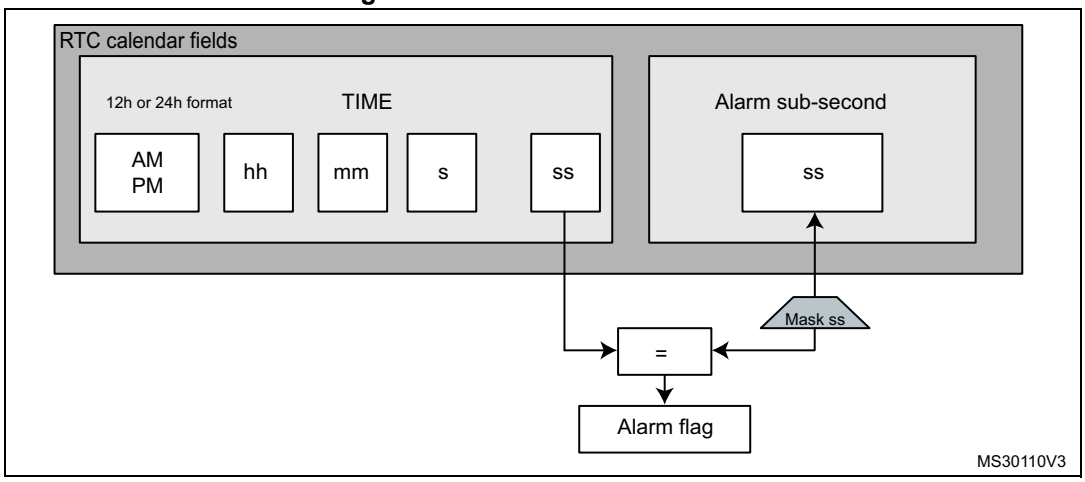
In this section: for RTC3, the BCD or mixed mode is considered.

The RTC unit provides programmable alarms, sub-second A and B, which are similar. They generate alarms with a high resolution (for the second division).

The value programmed in the alarm sub-second register is compared to the content of the sub-second field in the calendar unit.

The sub-second field counter counts down from the value configured in the synchronous prescaler to zero, and then reloads a value from the RTC_SPRE register.

Figure 5. Alarm sub-second field



Note: Mask ss is the most significant bit in the sub-second alarm. This bit is compared to the synchronous prescaler register. Mask ss is 4-bit length (from 0 to 15) for RTC2 and can be up to 6-bit length (from 0 to 31) for RTC3. This is due to the fact that the sub-second register RTC_SSR is always 16-bit length for RTC2 and can be expanded to 32 bits on some RTC3 products.

The alarm sub-second can be configured using the mask ss bit in the alarm sub-second register. *Table 10* shows the configuration possibilities for the mask register and provides an example with the following settings:

- Select LSE as the RTC clock source (for example LSE = 32768 Hz).
- Make sure that the asynchronous prescaler is set at 127.
- Make sure that the synchronous prescaler is set at 255 (the calendar clock is equal to 1Hz).
- Set the alarm A sub-second to 255 (put 255 in the SS[14:0] field).

Table 10. Alarm sub-second mask combinations

MASK SS	Alarm A sub-second behavior	Example result
0	There is no comparison on sub-second for alarm. The alarm is activated when the second unit is incremented.	Alarm activated every 1 s
1	Only the AlarmA_SS[0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1/128 s
2	Only the AlarmA_SS[1:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1/64 s
3	Only the AlarmA_SS[2:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1/32 s
4	Only the AlarmA_SS[3:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1/16 s
5	Only the AlarmA_SS[4:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 125 ms
6	Only the AlarmA_SS[5:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 250 ms

Table 10. Alarm sub-second mask combinations (continued)

MASK SS	Alarm A sub-second behavior	Example result
7	Only the AlarmA_SS[6:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 500 ms
8	Only the AlarmA_SS[7:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s
9	Only the AlarmA_SS[8:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s
10	Only the AlarmA_SS[9:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s
11	Only the AlarmA_SS[10:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s
12	Only the AlarmA_SS[11:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s
13	Only the AlarmA_SS[12:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s
14	Only the AlarmA_SS[13:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s
15	Only the AlarmA_SS[14:0] bit is compared to the RTC sub-second register RTC_SSR	Alarm activated every 1 s

Note: For RTC3 this table can be completed up to mask ss equal to 31.

The overflow bit in the sub-second register (bit 15 for RTC2 and 31 for RTC3) is never compared.

2.3.3 Alarms firmware examples

The RTC peripheral comes with a set of example projects so that the user can quickly becomes familiar with the RTC. Refer to the STM32Cube MCU Package provided with the product for a complete projects list.

For example, the user can find the following projects concerning alarms:

- For the NUCLEO-L412RB-P equipped with an RTC3:
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_Alarm
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples_LL\RTC\RTC_Alarm
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples_LL\RTC\RTC_Alarm_Init

The user can check if these examples are available in his product STM32Cube MCU Package or adapt them for his product.

2.4 RTC periodic wakeup unit

The STM32 MCUs provide several low-power modes to reduce the power consumption. The RTC features a periodic timebase and the wakeup unit is able to wake up the system from low-power modes. This unit is a programmable down-counting 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 down-counting 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 for RTC2 and TAMPALRM for RTC3 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.

2.4.1 Programming the auto-wakeup unit

Table 11 describes the steps required to configure the auto-wakeup unit.

Table 11. Steps to configure the auto-wakeup 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 (RTC2) /RTC_ICSR (RTC3)	For RTC2: It takes around 2 RTCCLK clock cycles due to clock synchronization. For RTC3(1): If WUCKSEL[2] = 0: WUTWF is set, it takes around 1 ck_wut + 1 RTCCLK cycles after WUTE bit is cleared. If WUCKSEL[2] = 1: WUTWF is set, it takes up to 1 ck_apre + 1 RTCCLK cycles after WUTE bit is cleared.
4	Program the value into the wakeup timer.	Set WUT[15:0] in RTC_WUTR register For RTC3 the user must also program WUTOCLR bits(2)	See *Section 2.4.2: 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 counting down.
7	Enable the RTC registers Write protection	Write 0xFF into the RTC_WPR register	RTC registers can no more be modified.

1. ck_wut is the wakeup timer clock input, and ck_spre the clock output by the RTC synchronous prescaler (usually 1Hz).
2. WUTOCLR bits of RTC_WUTR register is only available on RTC3. It allows to choose if the WUTF flag is cleared by software (WUTOCLR=0x0000) or automatically cleared by hardware when the auto-reload down counter reaches WUTOCLR value (0x0000<WUTOCLR<=WUT). For RTC2, WUTF must always be cleared by software.

2.4.2 Maximum and minimum RTC wakeup period

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

- Configuration 1: WUCKSEL[2:0] = 0b0xx for short wakeup periods (see *Periodic timebase/wakeup configuration for clock configuration 1*)
- Configuration 2: WUCKSEL[2:0] = 0b10x for medium wakeup periods (see *Periodic timebase/wakeup configuration for clock configuration 2*)
- Configuration 3: WUCKSEL[2:0] = 0b11x for long wakeup periods (see *Periodic timebase/wakeup configuration for clock configuration 3*)

Periodic timebase/wakeup configuration for clock configuration 1

Figure 6 shows the prescaler connection to the timebase/wakeup unit and *Table 12* 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 6. Prescalers connected to the timebase/wakeup unit for configuration 1

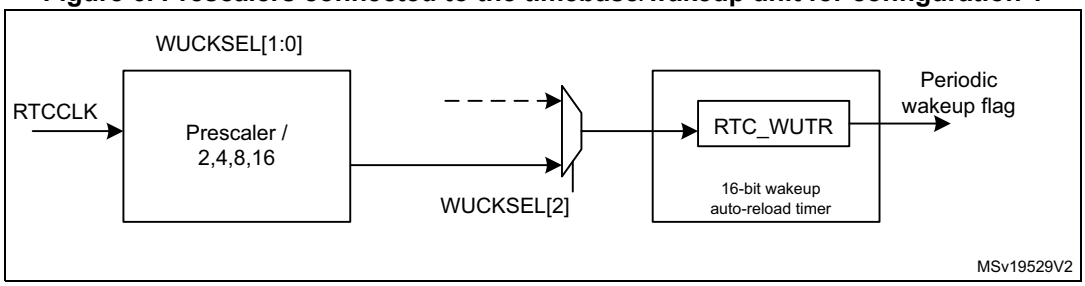


Table 12. Timebase/wakeup unit period resolution with clock configuration 1

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 is 488.28 µs. As a result:

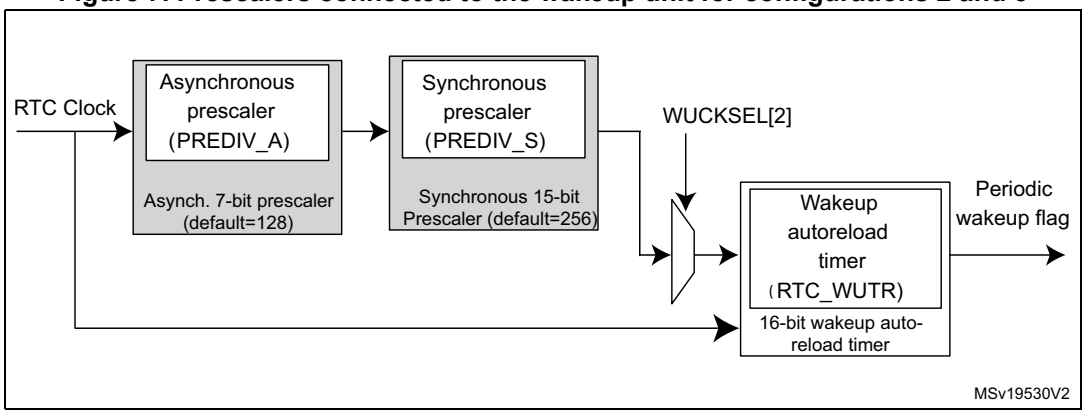
- The minimum timebase/wakeup period is (0x0001 + 1) x 61.035 µs = 122.07 µs.

  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 = 32 s.

Periodic timebase/wakeup configuration for clock configuration 2

Figure 7 shows the prescaler connection to the timebase/wakeup unit corresponding to configuration 2 and 3.

Figure 7. Prescalers connected to the wakeup unit for configurations 2 and 3



If ck_spre (synchronous prescaler output clock) is adjusted to 1 Hz, then:

- The minimum timebase/wakeup period is (0x0000 + 1) x 1 s = 1 s.
- The maximum timebase/wakeup period is (0xFFFF+ 1) x 1 s = 65536 s (18 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.

If ck_spre is adjusted to 1 Hz, then:

- The minimum timebase/wakeup period is:
  (0x10000 + 1) x 1 s = 65537 s (18 hours + 1 s)
- The maximum timebase/wakeup period is:
  (0x1FFFF+ 1) x 1 s = 131072 s (36 hours).

Summary of timebase/wakeup period extrema

When RTCCLK= 32768 Hz and ck_spre (Synchronous prescaler output clock) is adjusted to 1 Hz, the minimum and maximum period values are listed in *Table 13*.

Table 13. Min. and max. timebase/wakeup period when RTCCLK= 32768

Configuration	Minimum period	Maximum period
1	122.07 µs	32 s
2	1 s	18 hours
3	18 hours +1 s	36 hours

2.4.3 Wakeup firmware examples

The RTC peripheral comes with a set of example projects so that the user can quickly become familiar with the RTC. Refer to the STM32Cube MCU Package provided with the product for a complete projects list.

For example, the user can find the following projects concerning wakeup:

- For the NUCLEO-L412RB-P equipped with an RTC3:
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples_LL\RTC\RTC_ExitStandbyWithWakeUpTimer
- For the NUCLEO-G071RB equipped with an RTC3:
  - STM32Cube_FW_G0_Vx.y.z\Projects\NUCLEO-G071RB\Examples_LL\RTC\RTC_ExitStandbyWithWakeUpTimer_Init
- For the NUCLEO-G431RB equipped with an RTC3:
  - STM32Cube_FW_G4_Vx.y.z\Projects\NUCLEO-G431RB\Examples_LL\RTC\RTC_ProgrammingTheWakeUpTimer

The user can check if these examples are available in his product STM32Cube MCU Package or adapt them for his product.

2.5 Digital smooth calibration

2.5.1 RTC calibration basics

The term “quartz-accurate” has become a familiar phrase used to describe the accuracy of many time keeping functions. The quartz oscillators provide an accuracy far superior to that of other conventional oscillator designs, but they are not perfect. The quartz crystals are sensitive to temperature variations. *Figure 8* shows the relationship between accuracy (acc), temperature (T) and curvature (K) for a typical 32.768 kHz crystal. The curve follows the general formula given below:

acc = K x (T-To)², where:

- To = 25 °C ± 5 °C
- K = –0.032 ppm/°C^2

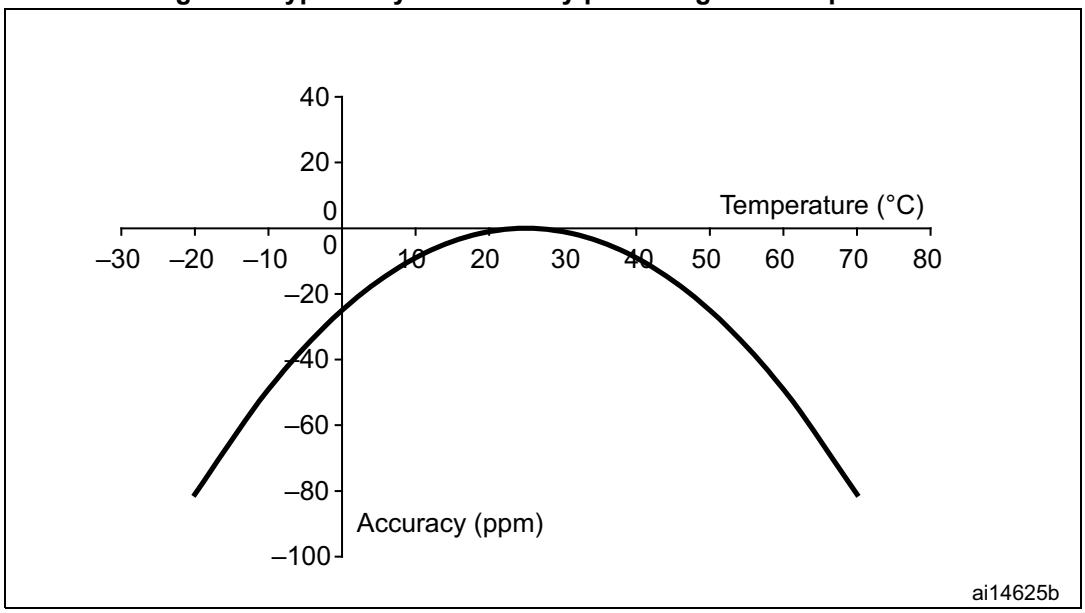
Note: The variable K is crystal-dependent, the value indicated here is for the crystal mounted on the STM32L476RG-Nucleo board. Refer to the crystal manufacturer for more details on this parameter.

The clocks used in most applications require a high degree of accuracy, and there are several factors involved in achieving this accuracy.

Two major approaches exist to compensate for an oscillator's oscillation frequency deviation when the generated signal is used to clock a time-keeping logic:

- The analog approach where the oscillator is built with embedded capacitor banks on both of the oscillator's input and output. Thus, to adjust the oscillation frequency, the software needs to configure the oscillator to switch on or off some of the embedded load capacitors to adjust the overall load capacitance seen by the crystal resonator. It must be noted that the two external load capacitors are always needed even with this kind of oscillators. The two external load capacitors make the dominant part of the crystal resonator's load capacitance. The oscillator's embedded capacitor banks are only intended to compensate for the capacitance value dispersion of the two external load capacitors or to compensate for the temperature variation. This approach suffers from most of drawbacks that generally analog circuits suffer from, like relatively important value dispersion from one chip to another, etc.
- The digital approach, where the deviation of the oscillation frequency is not compensated for at the oscillator level. Instead of that, the compensation is made at the time-keeping logic/function level. The time-keeping logic either adds or removes few clock cycles to/from the deviating clock signal that is feeding its counter. The chief advantage of this approach is the deterministic compensation effect from one device to another. This approach is adopted to compensate for the LSE oscillation frequency deviation when it is used to clock the RTC peripheral. The RTC peripheral is built with a dedicated register to configure the number of clock cycles to add or remove from the feeding clock signal.

Figure 8. Typical crystal accuracy plotted against temperature



2.5.2 Methodology

The RTC clock frequency can be corrected using a series of small adjustments by adding or subtracting individual ck_cal (smooth calibration clock) pulses.

For RTC2 ck_cal is always RTCCLK.

For RTC3, if the LPCAL bit of the RTC_CALR register is 0 then ck_cal is RTCCLK. If LPCAL is 1 then ck_cal is ck_apre (clock output by the RTC asynchronous prescaler). Moreover, when setting LPCAL to 1 and choosing ck_apre, the calibration consumption decrease, its accuracy remains the same and the calibration window changes to about 2^{20} x PREDIV_A x RTCCLK pulses instead of 2^{20} RTCCLK pulses when LPCAL=0.

The RTC clock can be calibrated with a resolution of about 0.954 ppm with a range from -487.1 ppm to +488.5 ppm.

This digital smooth calibration is designed to compensate for the inaccuracy of crystal oscillators due to the temperature, crystal aging.

Figure 9. Smooth calibration block for RTC2

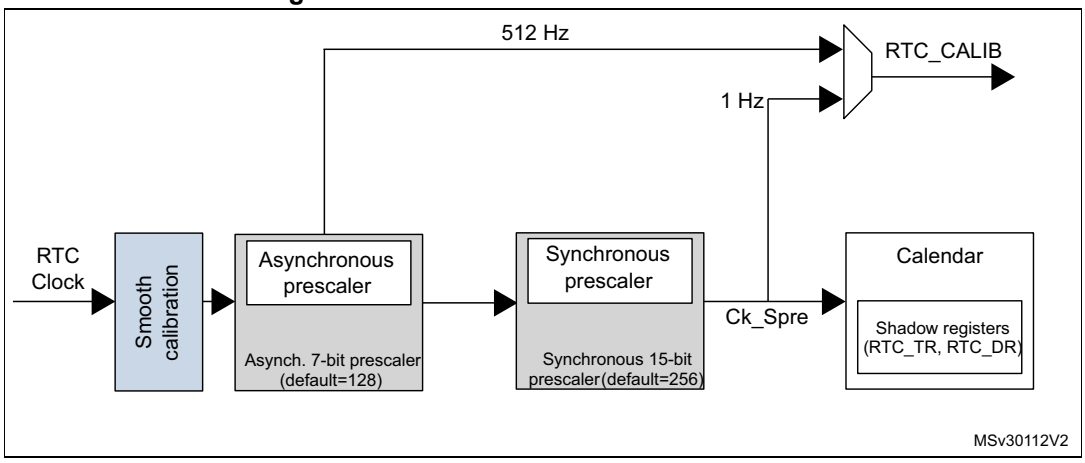
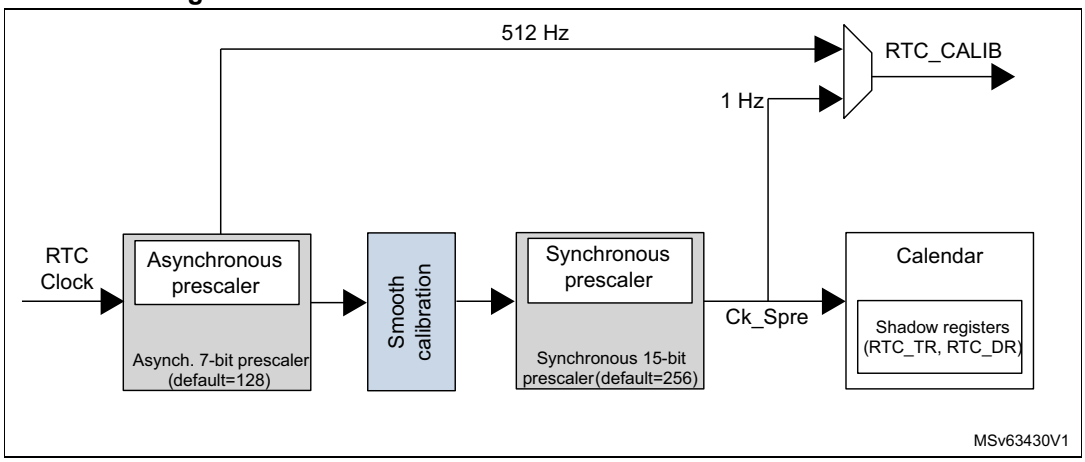


Figure 10. Smooth calibration block for RTC3 with LPCAL=1



The user can compute the clock deviation using the RTC_CALIB signal, then update the calibration block. It is also possible to input an external 1 Hz reference and make the adequate processing inside the MCU to correct the RTC clock. The user finds such example in *Section 4: STM32L4 API and digital smooth calibration application example* (RTC2) and in *Section 6: STM32L5 API and digital smooth calibration application example* (RTC3).

It is possible to check the calibration result using the calibration output 512 Hz or 1 Hz for the RTC_CALIB signal. Refer to *Table 4: Advanced RTC2 features* and *Table 5: Advanced RTC3 features*.

A smooth calibration consists of masking and adding N (configurable) 32 kHz-frequency pulses that are well distributed in a configurable window (8 s, 16 s or 32 s).

The number of masked or added pulses is defined using CALP and CALM in the RTC_CALR register.

By default, when the input frequency is 32768 Hz, the calibration window duration is 32 seconds (considering LPCAL=0 for RTC3). It can be reduced to 8 or 16 seconds by setting the CALW8 bit or the CALW16 bit in the RTC_CALR register:

Note: The 0.954 PPM accuracy of the smooth calibration is only achievable by 32 s calibration window, for 16 s the best accuracy is (0.954 x 2) and for 8 s is (0.954 x 4).

Example 1: setting CALM[0] to 1, CALP=0 and using 32 seconds as a calibration window results in exactly one pulse being masked for 32 seconds.

Example 2: setting CALM[2] to 1, CALP=0 and using 32 seconds as a calibration window results in exactly 4 pulses being masked for 32 seconds.

Note: Both the CALM and CALP can be used and, in this case, an offset ranging from -511 to +512 pulses can be added for 32 seconds (calibration window).

When the asynchronous prescaler is less than 3, CALP cannot be set to 1.

The formula to calculate the effective calibrated frequency (FCAL), given the input frequency (FRTCCLK), is:

$F_{CAL} = F_{RTCCLK} \times [1 + (CALP \times 512 - CALM) / (2^{20} + CALM - CALP \times 512)]$.

A smooth calibration can be performed on the fly so that it is changed when the temperature changes or if other factors are detected.

Checking the smooth calibration

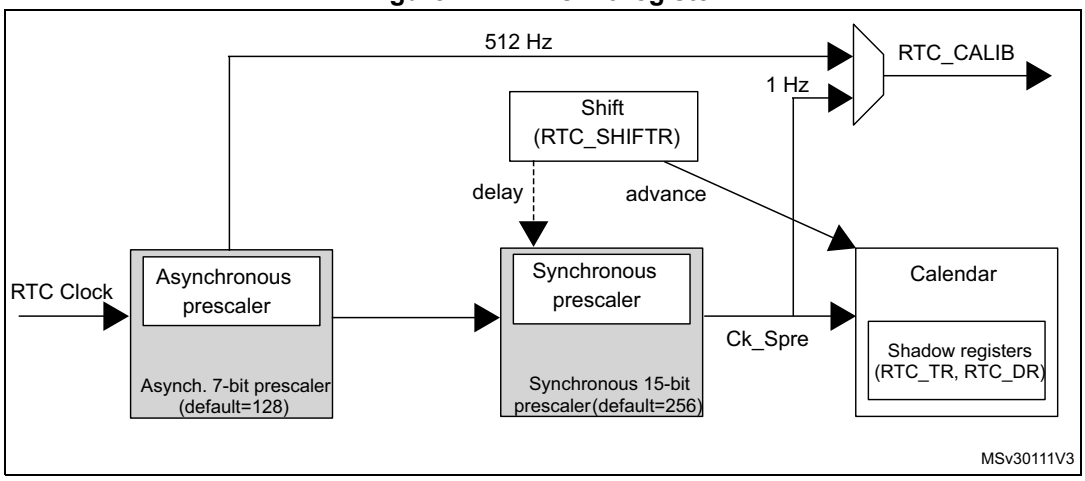
The smooth calibration effect on the calendar clock (RTC Clock) can be checked by:

- Calibration using the RTC_CALIB output (1 Hz).
- Calibration using the sub-second alarms.
- Calibration using the wakeup timer.

2.6 Synchronizing the RTC

The RTC calendar can be synchronized to a more precise clock, “remote clock”, using the RTC shift feature. After reading the RTC sub-second field, a calculation of the precise offset between the time being maintained by the remote clock and the RTC is made. The RTC can be adjusted by removing this offset with a fine adjustment using the shift register control.

Figure 11. RTC shift register



It is not possible to check the “Synchronization” shift function using the RTC_CALIB output since the shift operation has no impact on the RTC clock, other than adding or subtracting a few fractions from the calendar counter.

Correcting the RTC calendar time

If the RTC clock is advanced compared to the remote clock by n fractions of seconds, the offset value must be written in SUBFS, that is added to the synchronous prescaler’s counter. As this counter counts down, this operation effectively subtracts from (delays) the clock by:

Delay (seconds) = SUBFS / (PREDIV_S + 1)

If the RTC is delayed compared to the remote clock by n fractions of seconds, the offset value can effectively be added to the clock (advancing the clock) when the ADD1S function is used in conjunction with SUBFS, effectively advancing the clock by:

Advance (seconds) = (1 - (SUBFS / (PREDIV_S + 1))).

Caution: For RTC3, ADD1S has no effect in binary mode.

An example for this feature is provided in the X-CUBE-RTC Expansion Package associated to this application note and detailed in *Section 7: STM32L5 API and synchronization application example*.

2.7 RTC reference clock detection

For RTC3, this feature is available only in BCD mode (BIN=00).

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

The RTC_REFIN pin must 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 the 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 PREDIV_S = 0x00FF.

Note: This feature is only valid with the RTC clock from LSE oscillator oscillating at 32.768 kHz due to this requirement.

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_apre periods.

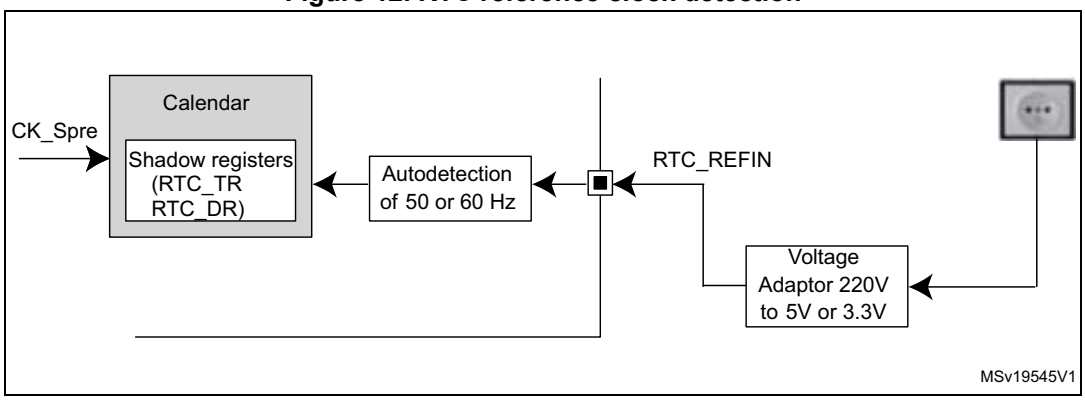
If the reference clock halts, the calendar is updated continuously based solely on the LSE clock. The RTC then waits for the reference clock using a detection window centered on the Synchronous Prescaler output clock (ck_spre) edge. The detection window is 7 ck_apre periods.

The reference clock can have a large local deviation (for instance in the range of 500 ppm), 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 50 Hz reference clock) because it is possible to 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.

Assuming that ck_ref is not lost more than once a day. So the total uncertainty per month would be 20 ms x 1x 30 = 0.6 s, which is much less than the uncertainty of a typical quartz (53 s/month for +/-20 ppm quartz).

Figure 12. RTC reference clock detection



Note: The reference clock calibration and the RTC synchronization (shift feature) cannot be used together.

The reference clock calibration is the best (ensures a high calibrated time) if the 50 Hz is always available. If the 50 Hz input is lost, the RTC accuracy is provided by the LSE crystal.

The reference clock detection cannot be used in Vbat mode.

The reference clock calibration can only be used if the user provides a precise 50 or 60 Hz input.

An example for this feature is provided in the X-CUBE-RTC Expansion Package associated to this application note and detailed in *Section 8: STM32L5 API and reference clock detection application example*.

2.8 RTC prescaler adjustment with LSI measurements

When LSI is selected as RTC clock source, there is a possibility to use a timer to measure its frequency and adjust the RTC synchronous prescaler depending on the result got. The goal is to improve the observed LSI accuracy.

The principle consists in connecting the LSI clock signal to the input capture of one of the STM32 timers (see the timers connected to LSI in the product reference manual). For each period of LSI, the timer must count the number of system core clock periods elapsed between two LSI rising edges associated. Then, the software exploit the number of periods counted by comparing it with the amount expected and adjust the RTC synchronous prescaler value in order to improve the RTC accuracy.

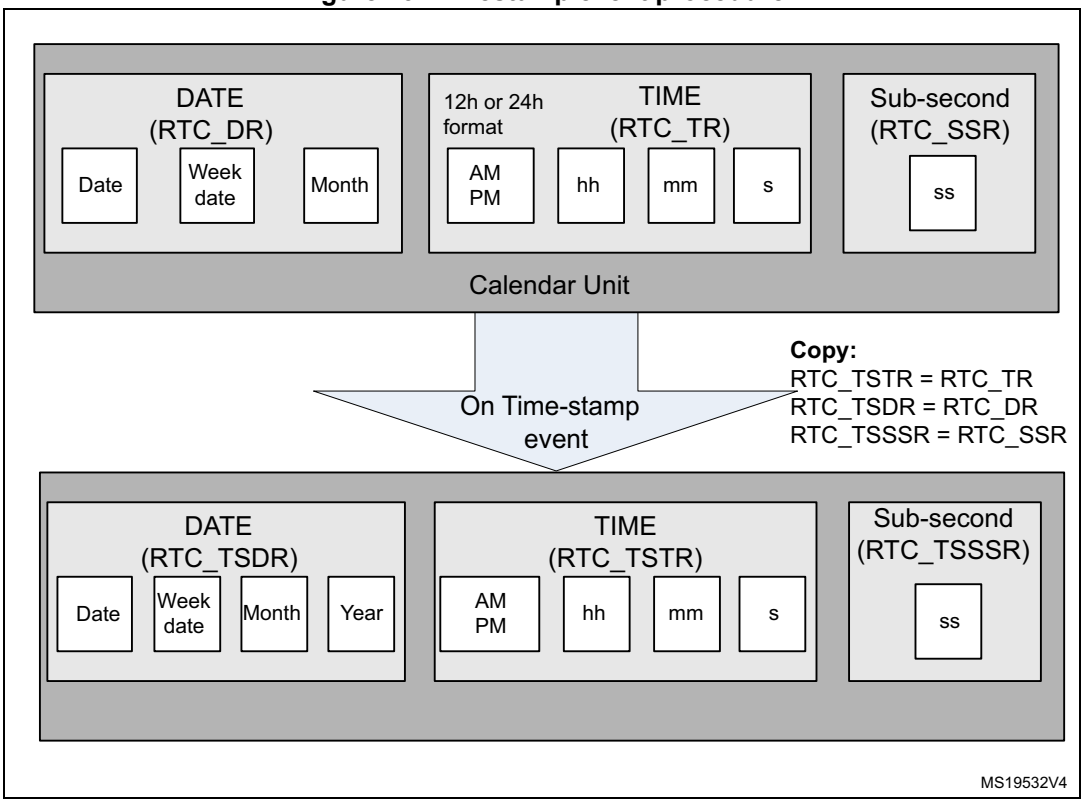
A firmware example is provided to illustrate this principle. It is available in the L4 STM32CubeL4 MCU Package for NUCLEO-L412RB-P (RTC3 product):
STM32Cube_FW_L4_V1.14.0\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_LSI

The user can check if this example is available in his product STM32Cube MCU Package or adapt them for his product.

2.9 Timestamp function

The timestamp feature provides the means to automatically save the current calendar when some specific events occur.

Figure 13. Timestamp event procedure



Provided that the timestamp function is enabled, the calendar is saved in the timestamp registers (RTC_TSTR, RTC_TSDR, RTC_TSSSR) when an internal or external timestamp event is detected. When a timestamp event occurs, the timestamp flag bit (TSF) in the RTC_ISR (RTC2)/RTC_SR (RTC3) register is set.

The events that can generate a timestamp are:

- An edge detection on the RTC_TS I/O
- A tamper event detection (from all RTC_TAMP I/Os)
- A switch to VBAT when the main supply if powered off (For example, it is available on STM32L4xxxx devices. Refer to the product reference manual and datasheet to verify availability for other products).

Table 14. Timestamp features

What to do	How to do it	Comments
Enable timestamp	Setting the TSE or ITSE or TAMPTS bit of RTC_CR register to 1	TAMPTS is only available on RTC3 to allow tamper events to trigger timestamp.
Detect a timestamp event by interrupt	Setting the TSIE bit in the RTC_CR register	An interrupt is generated when a timestamp event occurs.
Detect a timestamp event by polling	By polling on the timestamp flags (TSF or ITSF(1)) in the RTC_ISR (RC2)/RTC_SR (RTC3) register	To clear the flag, write zero on the TSF bit or ITSF.(2)
Detect a timestamp overflow event(3)	By checking on the timestamp over flow flag (TSOVF(4)) in the RTC_ISR (RC2)/RTC_SR (RTC3) register.	– To clear the flag, write zero in the TSOVF bit. – timestamp registers (RTC_TSTR and RTC_TSDR, RTC_TSSSR) maintain the results of the previous event. – If a timestamp 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 timestamp event occurs due to the synchronization process.
2. To avoid masking a timestamp event occurring at the same moment, the application must not write “0” into TSF bit unless it has already read it to “1”.
3. The timestamp overflow event is not connected to an interrupt.
4. There is no delay in the setting of TSOVF. This means that if two timestamp events are close to each other, 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.

2.9.1 Timestamp firmware examples

The RTC peripheral comes with a set of example projects so that the user can quickly become familiar with the RTC. Refer to the STM32Cube MCU Package provided with the product for a complete projects list.

For example, the user can find the following projects concerning timestamp:

- For the NUCLEO-L412RB-P equipped with an RTC3:
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_TimeStamp
- For the NUCLEO-G071RB equipped with an RTC3:
  - STM32Cube_FW_G0_Vx.y.z\Projects\NUCLEO-G071RB\Examples_LL\RTC\RTC_TimeStamp_Init

The user can check if these examples are available in his product STM32Cube MCU Package or adapt them for his product.

2.10 RTC tamper detection function

The RTC includes several tamper detection inputs. The tamper inputs can be configured to detect different types of tamper events (detailed below) and each one has an individual flag (TAMPxF bit in RTC_ISR (RTC2)/TAMP_SR (RTC3) register).

A tamper detection event generates an interrupt when the TAMPIE or TAMPxIE bit in RTC_TAMPCR (RTC2)/TAMP_IER (RTC3) register is set.

The configuration of the tamper filter, “TAMPFLT bits”, defines whether the tamper detection is activated on edge (set TAMPFLT to 00), or on level (TAMPFLT must be different from 00).

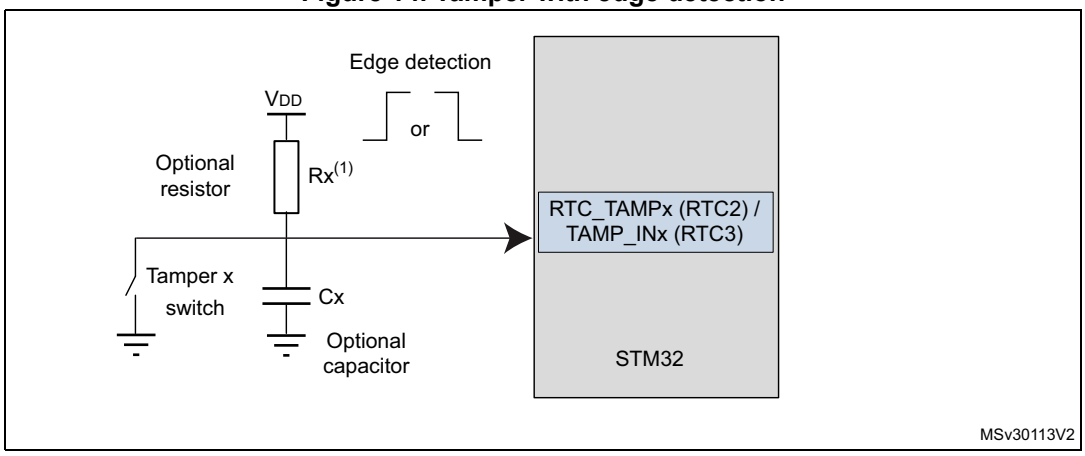
Note: The number of tamper inputs depends on product packages. Each input has a “TAMPxF” individual flag in the RTC_ISR (RTC2)/TAMP_SR (RTC3) register.

The DBP bit, for which the register associated depends on the product, must be set to allow write access to any TAMP registers after a system reset.

2.10.1 Edge detection on tamper input

When the TAMPFLT bits are set to zero, the tamper input detection triggers when a rising edge or a falling edge (depending on the TAMPxTRG bit) is observed on the corresponding RTC_TAMPx (RTC2)/TAMP_INx (RTC3) input pin.

Figure 14. Tamper with edge detection



1. If the power consumption is not a concern, the internal 40 kΩ pull-up resistor can be used. TAMPPUDIS set to 1 allows the bypass of the internal pull-up resistor.

Note: With the edge detection, the sampling and precharge features are deactivated.

Table 15. Tamper features (edge detection)

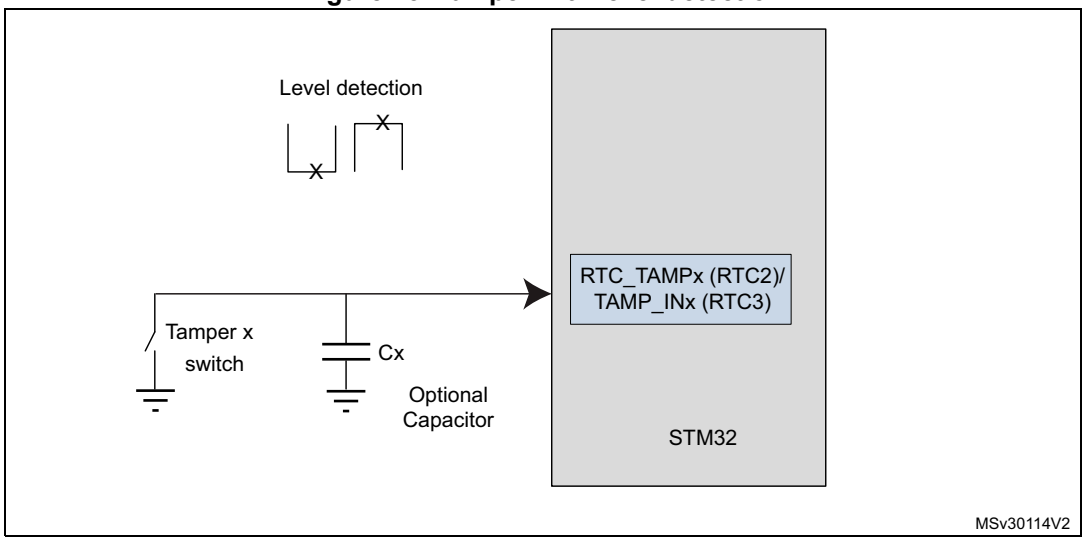
What to do	How to do it	Comments
Enable tamper	Set the TAMPxE bit of RTC_TAMPCR (RTC2) /TAMP_CR1 (RTC3) register to 1	-
Select tamper active edge detection	Select with TAMPxTRG bit in RTC_TAMPCR (RTC2)/ TAMP_CR2 (RTC3) register	The default edge is rising edge.
Detect a tamper event by interrupt	Set the TAMPIE or TAMPxIE bit in the RTC_TAMPCR (RTC2) /TAMP_IER (RTC3) register	An interrupt is generated when a tamper detection event occurs.
Detect a tamper event by polling	Poll the tamper flag (TAMPxF) in the RTC_ISR (RTC2)/ TAMP_SR (RTC3) register	RTC2: To clear the flag, write zero in the TAMPxF bit. RTC3: Write 1 in the bit CTAMPxF in the TAMP_SCR clears the TAMPxF bit in the TAMP_SR register.

2.10.2 Level detection on tamper input

Setting the tamper filter “TAMPFLT” to a value other than zero means that the tamper input triggers when a selected level (high or low) is observed on the corresponding RTC_TAMPx (RTC2)/TAMP_INx (RTC3) input pin.

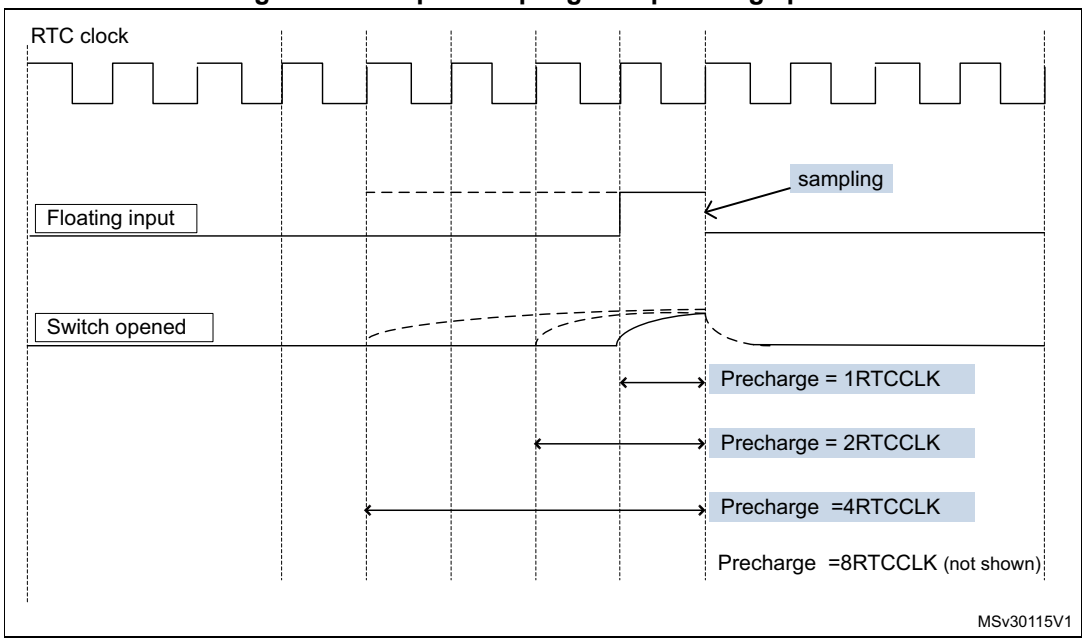
A tamper detection event is generated when either 2, 4 or 8 (depending on the TAMPFLT value) consecutive samples are observed at the selected level.

Figure 15. Tamper with level detection



Using the level detection (tamper filter set to a non-zero value), the tamper input pin can be precharged by resetting TAMPPUDIS through an internal resistance before sampling its state. In order to support the different capacitance values, the length of the pulse during which the internal pull-up is applied can be 1, 2, 4 or 8 RTCCLK cycles (see TAMPPRCH bits).

Figure 16. Tamper sampling with precharge pulse



Note: When the internal pull-up is not applied, the I/Os Schmitt triggers are disabled in order to avoid an extra consumption if the tamper switch is open.

The trade-off between the tamper detection latency and the power consumption through the weak pull-up or external pull-down can be reduced by using a tamper sampling frequency feature. The tamper sampling frequency is determined by configuring the TAMPFREQ bits in the RTC_TAMPCR (RTC2)/TAMP_FLTCR (RTC3) register. When using the LSE at 32768 Hz as the RTC clock source, the sampling frequency can be 1, 2, 4, 8, 16, 32, 64, or 128 Hz.

Table 16. Tamper features (level detection)

What to do	How to do it	Comments
Enable Tamper	Set the TAMPxE bit of RTC_TAMPCR (RTC2)/TAMP_CR1(RTC3) register to 1	-
Configure Tamper filter count	Configure TAMPFLT bits in RTC_TAMPCR (RTC2)/TAMP_FLTCR (RTC3) register	-
Configure Tamper sampling frequency	Configure TAMPFREQ bits in RTC_TAMPCR (RTC2)/TAMP_FLTCR (RTC3) register	Default value is 1Hz
Configure tamper internal pull-up and precharge duration	Configure TAMPPUDIS and TAMPPPRCH bits in RTC_TAMPCR (RTC2)/TAMP_FLTCR (RTC3) register	-
Select Tamper active edge/Level detection	Select with RTC_TAMPCR (RTC2)/TAMP_CR2(RTC3) register	Edge or Level is depending on tamper filter configuration.
Detect a Tamper event by interrupt	Set the TAMPIE or TAMPxIE bit in the RTC_TAMPCR (RTC2)/TAMP_IER (RTC3) register	An interrupt is generated when tamper detection event occurs.
Detect a Tamper event by polling	Poll the timestamp flag (TAMPxF) in the RTC_ISR (RTC2)/TAMP_SR (RTC3) register	To clear the flag, write zero in the TAMPxF bit.

2.10.3 Timestamp on tamper detection event

By setting the TAMPTS bit to 1, any tamper event (with edge or level detection) causes a timestamp to occur. Consequently, the timestamp flag and timestamp overflow flag are set when the tamper flag is set and work in the same manner as when a normal timestamp event occurs.

Note: It is not necessary to enable or disable the timestamp function when using this feature.

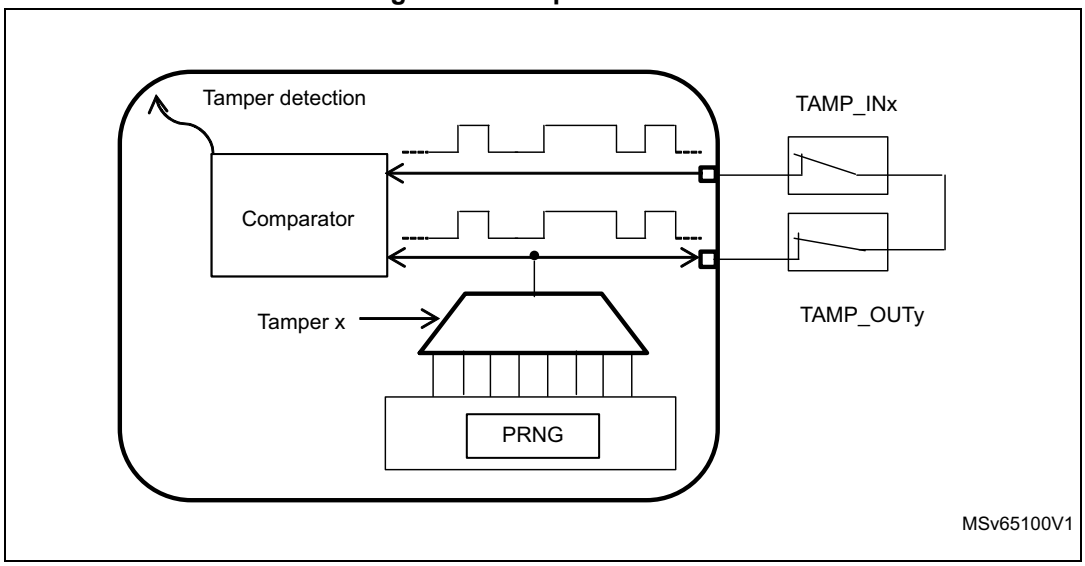
2.10.4 Active tamper detection (RTC3 only)

Principle:

Tamper detection can be made robust against tamper pin being externally opened or shorted. That is done thanks to active tamper.

The principle is to set TAMP_OUTy output pins to deliver a random message, and TAMP_INx input pins to receive it. If the message sent is not the one received, it is considered that a tamper event has occurred: TAMPxF is set in TAMP_SR register for the TAMP_INx pin detecting the error. Depending on the tamper pins available, several organizations can be configured. For example, for 8 tamper pins, the user can have 4 outputs and 4 inputs with 4 different messages, or 7 inputs with 1 output and 1 message exchanged.

Figure 17. Tamper detection



Active pins IN/OUT association:

The user configures TAMP_INx as active input pins by setting the TAMPxAM bit to 1 in TAMP_ATCR1 register.

By default, when TAMP_INx is activated as active tamper pins, the TAMP_OUTx pin is associated to it to implement the active tamper detection mechanism.

If other TAMP_INy pins need to be associated to TAMP_OUTx pin (the one mentioned above) for the application, the ATOSHARE (Active Tamper Output Share) bit can be used to make all the TAMP_OUT pins, including TAMP_OUTx, sharable.

When ATOSHARE is set to 1, ATOSELi (active tamper output selection) allows the choice of the new TAMP_OUT pin associated to TAMP_INi.

Considering there is at least 2 active tamper pins on the product, The following example can be given: TAMP_OUT1 can be used for comparison with TAMP_IN1 and TAMP_IN2 by configuring and enabling both TAMP_IN1 and TAMP_2 in active mode,
with ATOSHARE = 1, ATOSEL1 = 00 and ATOSEL2 = 00.

Note: ATOSHARE and ATOSELi bits are respectively in TAMP_ATCR1 and TAMP_ATCR2 registers.

Filtering:

As for passive tamper detections (not active ones), the input can be filtered by setting FLTEN to 1. With this action, a tamper event is triggered when 2 comparisons among 4 are false (received message different from the sent one).

Message randomization:

Concerning the randomization of the message sent by the active output pins, a pseudo-random number generator (PRNG value in TAMP_ATOR register) is used and need to be fed periodically with a new seed. To ensure that, the user needs to maintain four 32-bit random values in the code. These last one are not necessarily generated by a random generator, the user can choose random values and update it with an atypical mathematic formula.

By writing consecutively these four values in TAMP_ATSEEDR register, the user feeds the PRNG and allows active output pins to renew the randomization of their message.

The PRNG takes several APB clock cycles (refer to the product documentation) to take into account the seed given, during this time the SEEDF bit in the TAMP_ATOR register is set and the TAMP_APB clock must not be switched off. Active tamper output pins are only activated after the PRNG has taken its seed into account. Then, the user must wait SEEDF to be cleared before, for example, entering low-power modes.

If the user doesn't want to enter low-power modes, the update of the seed can be done during tamper active activity. The update rate for the seeds depends on the number of tamper active outputs used, the product reference manual gives advisable time references for this action.

Initialization:

If INITS is not set in TAMP_ATOR register, the active tamper initialization must be done:

- Configure the active tamper clock prescaler with ATCKSEL in TAMP_ATCR1. Set the filter and associate TAMP in and out pins as needed (explained above).
- Enable tamper pins used in TAMP_CR1 register (all in the same write access).
- Feed the PRNG and wait for SEEDF to be cleared. Then backup registers are protected by active tamper.

If INITS is set, the tamper active feature is already initialized. The user only must update the PRNG seed for a more robust randomization.

2.10.5 Internal tamper detection (RTC3 only)

The internal tamper detection feature allows the detection of transient and environmental perturbations. An internal tamper event is enabled with the ITAMPxE bit in TAMP_CR1 register. The interrupt is enabled by setting ITAMPxIE in TAMP_IER register. Then, if the event occurs, the ITAMPxF flag is set in TAMP_SR register, and the ITAMPxMF flag is set in TAMP_MISR if the interrupt has been enabled.

To clear both flags (polling and interrupt mode one), the corresponding bit in TAMP_SCR register must be set.

The STM32 devices may implement some of the internal tampers listed below (refer to device reference manual):

ITAMP1: Supply voltage monitoring

ITAMP2: Temperature monitoring

ITAMP3: LSE monitoring

ITAMP4: HSE monitoring

ITAMP5: RTC calendar overflow

ITAMP6: JTAG/SWD access when RDP>0 or ST manufacturer readout (depends on device)

ITAMP7: Not used yet for a product

ITAMP8: Monotonic counter 1 overflow

ITAMP9: Not used yet for a product

ITAMP10: Not used yet for a product

ITAMP11: Not used yet for a product

From ITAMP12 to ITAMP16: Not used yet for a product

Moreover, the electrical characteristics for each event depend on the device. Refer to its datasheet to get precise values and conditions on event triggering (example: voltage value threshold for voltage monitoring).

As for every tamper detection features, internal tamper events allow attacks to be detected on STM32. For instance, some attacks consist in going back in time in the RTC calendar to execute specific code. Since the user can only increment the RTC physically, going back in time can be done by making it overflow. This justifies the existence of the ITAMP5 signal.

A firmware example exploiting the ITAMP5 event is provided in the X-CUBE-RTC Expansion Package associated to this application note and detailed in *Section 9*.

The RTC3 tamper detection unit embeds also a monotonic counter implemented in the TAMP_COUNT1R register (see *Table 4* and *Table 5*) check the availability in the product and associated to ITAMP8. This register can be read and is incremented each time the user writes it whatever the written value. It cannot roll-over and is frozen when it has reached its maximum. It can be used in the application as a low-power counter robust to system reset. For example, it can count the number of time a sensitive part of code is executed to verify it does not exceed a limit.

2.10.6 Tamper detection firmware examples

The RTC peripheral comes with a set of example projects so that the user can quickly become familiar with the RTC. Refer to the STM32Cube MCU Package provided with the product for a complete projects list.

For example, the user can find the following projects concerning tamper detection:

- For the NUCLEO-L412RB-P equipped with an RTC3:
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_Tamper
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples_LL\RTC\RTC_Tamper
- For the NUCLEO-G071RB equipped with an RTC3:
  - STM32Cube_FW_G0_Vx.y.z\Projects\NUCLEO-G071RB\Examples_LL\RTC\RTC_Tamper_Init
- For the STM32L552E evaluation board equipped with an RTC3:
  - STM32Cube_FW_L5_Vx.y.z\Projects\STM32L552E-EV\Examples\RTC\RTC_ActiveTamper

The user can check if these examples are available in his product STM32Cube MCU Package or adapt them for his product.

2.11 Backup registers

The 32-bit backup registers (RTC_BKPxR) are reset when a tamper detection event occurs. These registers are powered-on by VBAT when VDD is switched off (Some products do not implement this feature. For example it is not the case for STM32L0 Series and STM32L1 Series. Refer to *Table 4* and *Table 5*, they are not reset by a system reset, and their contents remain valid when the device operates in low-power mode.

For RTC2:

For STM32L0 Series, STM32L4 Series and STM32F7 Series, the 32-bit backup registers (RTC_BKPxR) are not reset if the TAMPxNOERASE bit is set, or if TAMPxMF (Tamper Mask Flag) is set in the RTC_TAMPCR register.

For RTC3:

The 32-bit backup registers (TAMP_BKPxR) are not reset if the TAMPxNOER bit is set, or if TAMPxMSK is set in the TAMP_CR3 register.

For RTC2 and RTC3, disabling the backup register reset feature allows a LPTIM event to be triggered from the tamper pin without erasing the backup registers. This also allows to take benefit of the tamper input digital filtering, either to generate triggers or to generate interrupts.

Note: The VBAT availability, the LPTIM availability and the number of backup registers depend on the product. Refer to Table 4: Advanced RTC2 features and Table 5: Advanced RTC3 features.

The RTC3 backup registers can be protected against non-secure or unprivileged accesses (see *Section 2.14*).

For this goal, BKPRWDPROT[7:0] and BKPWDPROT[7:0] in TAMP_SMCR register, are used to program the backup registers offset, thus delimiting three protection zones for the backup registers:

- Zone 1 is read and write secure
- Zone 2 is read non-secure and write secure
- Zone 3 is read and write non-secure

2.12 Alternate function RTC outputs

The RTC peripheral has two outputs:

- RTC_CALIB (RTC2)/CALIB (RTC3), used to generate an external clock.
- RTC_ALARM (RTC2)/TAMPALARM (RTC3), a unique output resulting from the multiplexing of the RTC alarm and wakeup events (and also tamper detection events for RTC3).

For RTC3 these outputs can be associated to two STM32 pins against one pin for RTC2. See OUT2EN bit in RTC_CR register of RTC3 to choose which function is output on which pin.

2.12.1 RTC_CALIB output

The RTC_CALIB output can be used to generate a 1 Hz or 512 Hz signal and used to measure the deviation of the RTC clock when compared to a more precise clock.

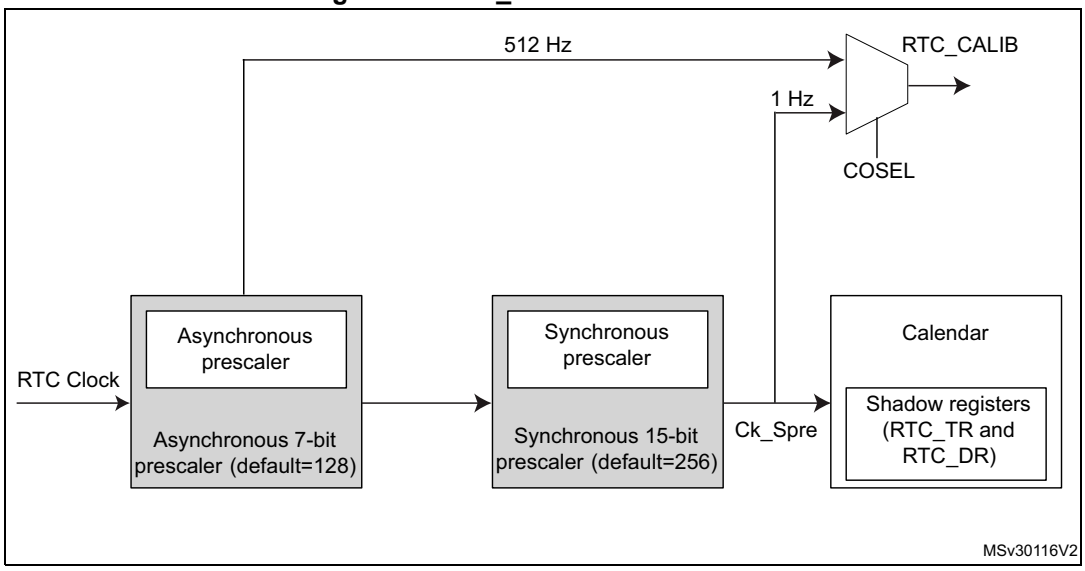
Setting 512 Hz as the output signal

1. Select LSE at 32768 Hz as RTC clock source.
2. Set the asynchronous prescaler to the default value 128.
3. Enable the output calibration by setting “COE” to 1.
4. Select 512 Hz as the calibration output by setting COSEL to 0.

Setting 1 Hz as the output signal

1. Select LSE at 32768 Hz as the RTC clock source.
2. Set the asynchronous prescaler to the default value 128.
3. Set the synchronous prescaler to the default value 256.
4. Enable the output calibration by setting “COE” to 1.
5. Select 1 Hz as the calibration output by setting COSEL to 1.

Figure 18. RTC_CALIB clock sources



Maximum and minimum RTC_CALIB 512 Hz output frequency

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

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

Table 17. RTC_CALIB output frequency versus clock source

RTC clock source	RTC_CALIB output frequency	
	Minimum (PREDIV_A[5:0] = 0b111 111) (div64)	**Maximum (PREDIV_A[5:0] = 0b100 000(1)) (div32)**
HSE_RTC = 1 MHz	15,625 kHz	30.303 kHz
LSE = 32768 Hz	512 Hz (default output frequency)	993 Hz
LSI = 32 kHz	500 Hz	969 Hz
LSI = 37 kHz	578 Hz	1.121 kHz
LSI = 40 kHz	625 Hz	1.212 kHz

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.

2.12.2 RTC_ALARM (RTC2)/TAMPALRM (RTC3) output

It is possible to root the alarm, wake-up and tamper flags to the RTC ouputs (1 output for RTC2, 2 for RTC3) thanks to the OSEL bits in the RTC_CR register.

For RTC2:

OSEL=0b01 roots alarm A to the RTC_ALARM output.

OSEL=0b10 roots alarm B to the RTC_ALARM output.

OSEL=0b11 roots wake up flag to the RTC_ALARM output.

Once OSEL fixed, the output then reflects the selected flag stored in RTC_ISR

For RTC3:

The OSEL selection follows the same principle than for RTC2 but when TAMPOE bit in RTC_CR is set, tamper flags are ORed with the flag already chosen (alarm A, alarm B or wake-up) before the OR result to be rooted to the TAMPALRM output.

If TAMPOE=1 and OSEL=0b00, TAMPALRM output reflects only the tamper flags.

In both cases RTC2 and RTC3, output pin polarity is chosen with POL bit in RTC_CR: when POL=1 the opposite state of the flag concerned is output on RTC_ALARM (RTC2)/ TAMPALRM (RTC3). The pin can also be configured as open drain or push-pull with the ALARMOUTTYPE (RTC2)/TAMPALRM_TYPE (RTC3) bit of the same register: 0 is push-pull, 1 is open-drain.

For RTC3, an internal pull-up can be added to the output pin by setting TAMPALRM_PU in RTC_CR.

2.13 RTC safety aspects

2.13.1 RTC register write protection

To protect the RTC registers against possible unintentional write accesses after reset, the RTC registers are initially, after a backup domain reset, locked. They must be unlocked to update the current calendar time and date.

The write-access to the RTC registers is enabled by writing a key in the write protection register (RTC_WPR).

The following sequence is 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.

Any write access to the RTC_WPR register different from the above described write sequence activates the write-protection mechanism for the RTC registers.

2.13.2 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 the *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 the *Free-running mode*.

The INIT bit of the RTC_ISR (RTC2)/RTC_ICSR (RTC3) register enables the user 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 (RTC2)/RTC_ICSR (RTC3) INITF flag flag is set.

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

Only a power-on reset can reset the calendar on microcontrollers without the VBAT pin. A system reset does not affect it but resets the shadow registers (the APB bus interface registers clocked by the APB bus clock) 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 the RTC_ISR (RTC2)/RTC_ICSR (RTC3) 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. The calendar can also be reseted by setting the BDRST bit in the RCC block, even when VBAT is present.

2.13.3 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 (RTC2)/RTC_ICSR (RTC3) 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 APB bus clock (the APB bus by which the RTC

peripheral is accessed). After a system reset or after exiting the initialization mode, the application must wait for the RSF bit to be set before reading the calendar shadow registers.

When the system is woken up from low-power modes (SYSCLK was off, consequently, the APB clock was off too), 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 low-power mode.

By setting the “BYPSHAD” bit to 1 in the RTC_CR register, the calendar values are taken directly from the calendar counters instead of reading the shadow register. In this case, it is not mandatory to wait for the synchronization time, but the calendar registers consistency must be checked by the software. The user must read the required calendar field values. The read operation must then be performed again. The results of the two read sequences are then compared. If the results match, the read result is correct. If they do not match, the fields must be read one more time, and the third read result is valid.

Note: After resetting the BYPSHAD *bit, the shadow registers may be incorrect until the next synchronization. In this case, the software must clear the RSF bit then wait for the synchronization (RSF must be set) and finally read the shadow registers.*

2.14 Reducing power consumption

The RTC is designed to minimize the power consumption. The way to configure it can optimize further its consumption.

2.14.1 Using the right power reduction mode

The RTC peripheral can be active in the following low-power modes (listed from the biggest current consumer to the smallest):

- Sleep mode
- Stop mode if the RTC clock is provided by LSE or LSI
- Standby mode if the RTC clock is provided by LSE or LSI
- Shutdown mode if the RTC clock is provided by LSE

The precise list of modes compatible with the product and their consumption are listed in its reference manual and datasheet.

Thanks to the RTC wake-up unit the user can wake the STM32 up from this mode periodically. Depending on what the user wants to do during the low-power phases and the current consumption target, the user must choose the right low-power mode and wake-up frequency.

2.14.2 Use tamper pin internal pull-up resistor

The internal pull-up resistor in the tamper input pad is only applied for a short period of time (1, 2, 4 or 8 RTCCLK cycles). So using the RTC internal pull-up resistor instead of standard IO pull-up/pull-down or external pull-up/pull-down ensures the lowest power consumption.

The trade-off between the tamper detection latency and the power consumption by the RTC internal pull-up can be optimized using TAMPFREQ field. It determines the frequency of the tamper sampling from 128 Hz to 1 Hz when RTCCLK equals 32768 Hz.

Note: Refer to the microcontroller datasheet and reference manual for the availability and electrical characteristics of the pull-up resistors.

2.14.3 Setting the RTC prescalers

The prescalers used for the calendar are divided into an asynchronous and a synchronous prescalers. Increasing the PREDIV_A value of the asynchronous prescaler while reducing the synchronous prescaler accordingly to keep the 1 Hz output, reduces the RTC power consumption.

On RTC3, the RTC dynamic consumption is optimized for LPCAL = 1 and PREDIV_A+1 being a power of 2. Note that for this consumption optimization, the RTC is considered as the only peripheral to use LSE.

2.14.4 External optimization factors

The prescaler linked to LSI clock can also optimize the consumption. Thus, to make profit of it the user needs to select LSI as RTC clock source. This prescaler divides LSI by 128 to get a rough 250 Hz clock instead of the former 32 kHz and is activated using LSIPRE bit in the RCC_CSR register.

Concerning LSE, the user can set its oscillator driving strength at the lowest level to reduce its consumption down to a typical 250 nA against 630 nA for a high drive capability. This is done by setting LSEDRV to 0b00 in RCC_BDCR register.

2.14.5 Low-power management firmware examples

The RTC peripheral comes with a set of example projects so that the user can quickly become familiar with the RTC. Refer to the STM32Cube MCU Package provided with the product for a complete projects list.

For example, the user can find the following projects concerning low-power management:

- For the NUCLEO-L412RB-P equipped with an RTC3:
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_LowPower_STANDBY
- For the NUCLEO-L412KB equipped with an RTC3:
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412KB\Examples\PWR\PWR_STANDBY_RTC
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412KB\Examples\PWR\PWR_STOP1_RTC
  - STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412KB\Examples\PWR\PWR_STOP2_RTC

The user can check if these examples are available in his product STM32Cube MCU Package or adapt them for his product.

2.15 RTC3 secure and privileged protection modes

On some STM32, the choice to run in secure or non-secure modes can be available. These two mode have their own environment so that a trustful zone, not accessible in non-secure mode, can be maintained in secure mode (TrustZone feature).

Moreover, part of their environment can be associated to a privileged or a non-privileged mode allowing a second level of restriction and security.

For more information on privileged and secure modes, refer to security dedicated STM32 application notes and to Arm® cortex®-M documentation.

On some products (see *Table 4* and *Table 5*), the programming of registers can allow the user to restrict the read and write operations on RTC/TAMP registers to secure or privileged modes. Some configuration bits are even able to establish the restriction only for some RTC registers. These registers/bits to program can differ depending on the product used. That is why they are not mentionned here and the user must refer to the product reference manual.

Note: The backup registers and the monotonic counter have their own protection mode settings.

The most obvious goal for establishing such a protection is to make sure the RTC registers are not modified by hazardous code or malicious attack.

3 STM32L4 API and tampering detection application example

3.1 STM32Cube firmware libraries

The RTC peripheral comes with:

- a firmware driver API abstracting the RTC features for the end-user. Refer to stm32l4xx_hal_rtc.c and stm32l4xx_hal_rtc_ex.c files in \STM32Cube_FW_L4_Vx.y.z\Drivers\STM32L4xx_HAL_Driver\
- a set of example projects so that the user can quickly become familiar with the RTC. Refer to the examples in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Eval\Examples\RTC

3.2 STM32Cube expansion firmware

This application note comes with the X-CUBE-RTC Expansion Package upon the STM32Cube firmware libraries.

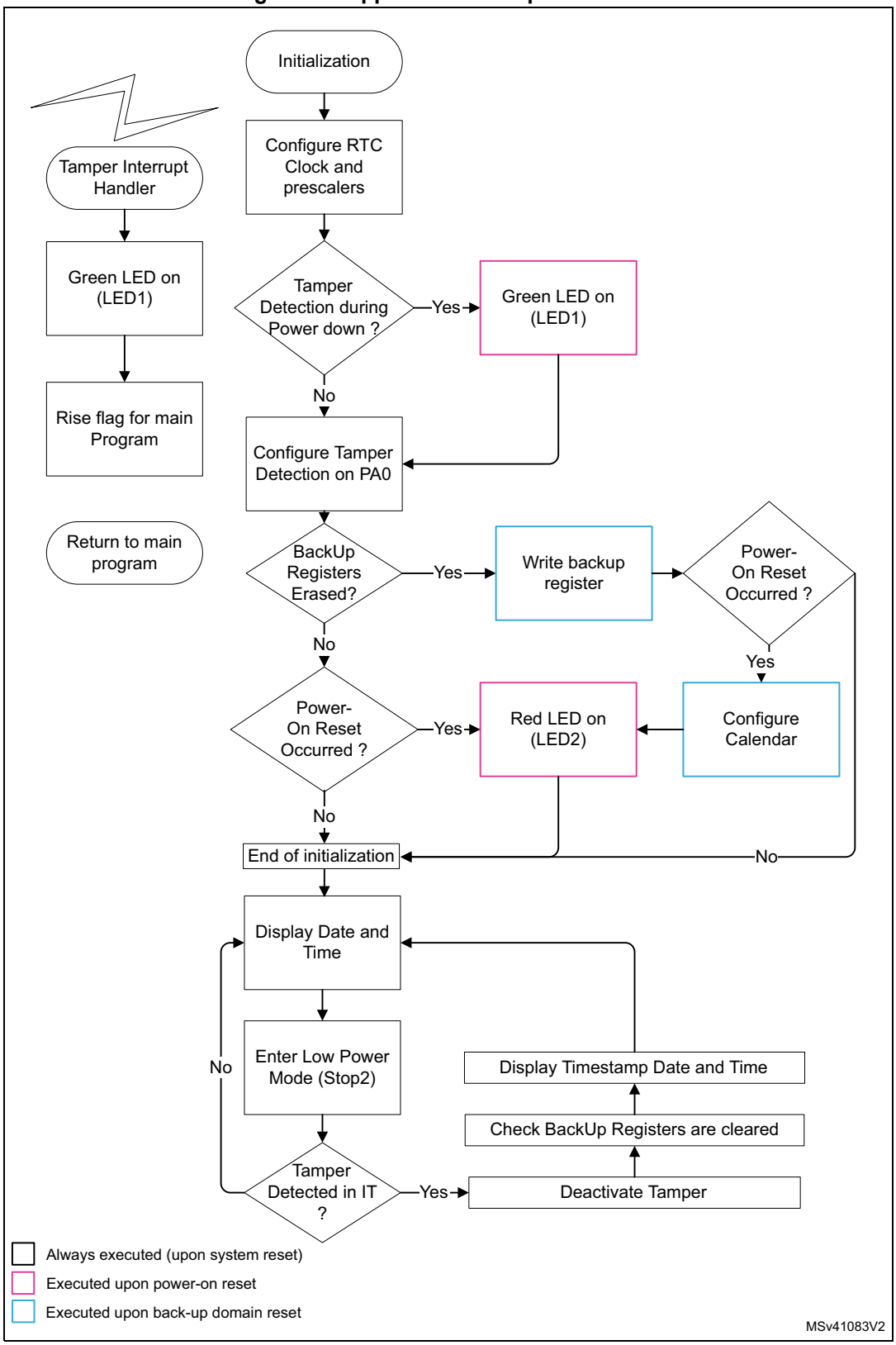
X-CUBE-RTC shows a concrete example where a real-time clock must be maintained alive while using as less power as possible: STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\STM32L476G_EVAL\RTC_TamperVBATT.

The firmware implements:

- a few hints order to ensure a low current consumption
- a display in the debugger (date and time, timestamp date and time) and on LEDs (power up events, reset events, tamper events, error)
- a display on the TFT screen, using STemWin library
- the tampering detection capability both when the main power supply (VDD) is present and when the RTC is battery powered
- the ability to timestamp the tampering event
- the ability to erase the backup registers that may contain sensitive data upon tamper detection

The application flowchart is shown in *Figure 19*.

Figure 19. Application example flowchart

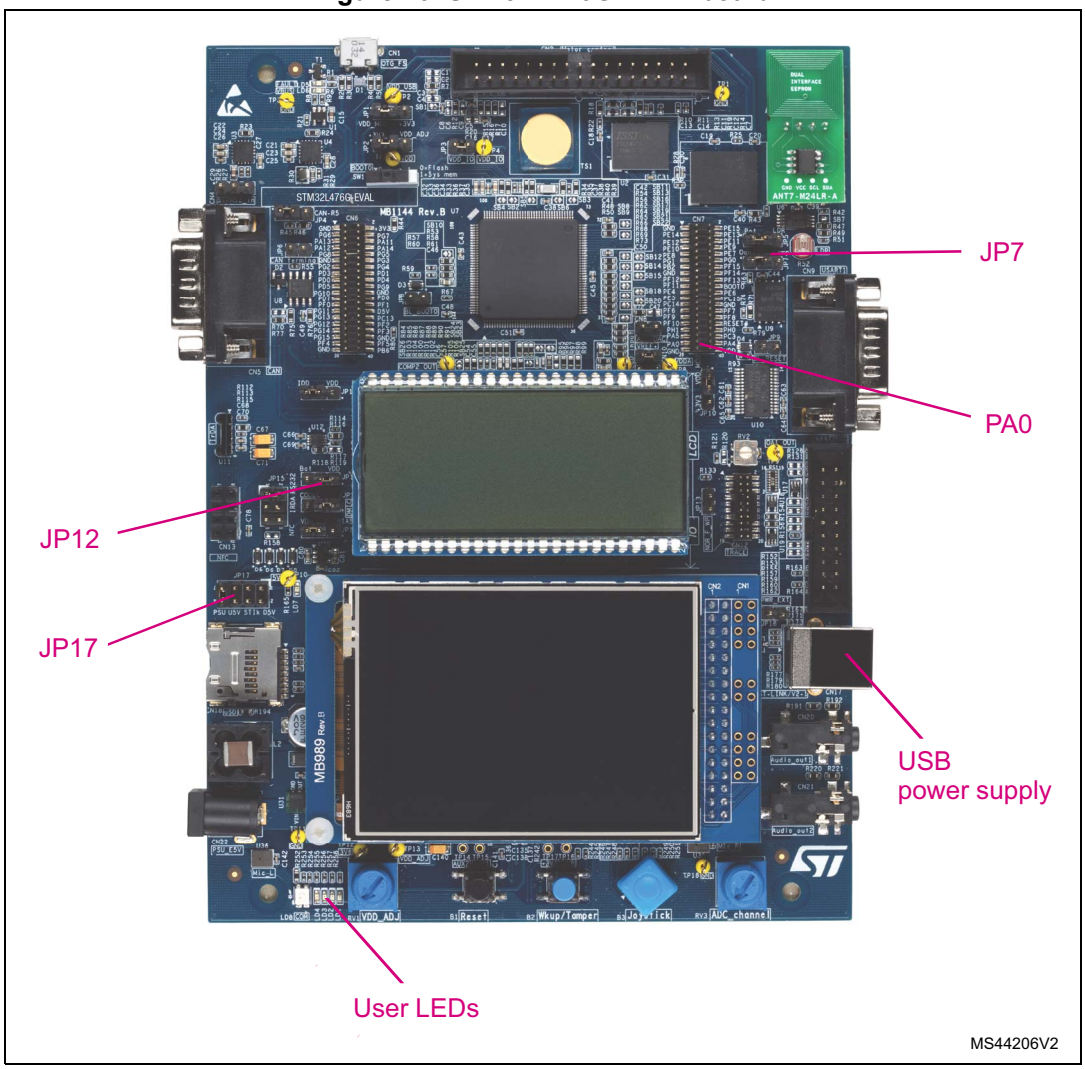


3.3 Application example project

3.3.1 Hardware setup

In order to use the application example project, the user needs a STM32L476G evaluation board (STM32L476G-EVAL).

Figure 20. STM32L476G-EVAL board



1. This picture is not contractual.

Supply the board using the USB cable. For this example to work, the user needs to tie PA0 (tamper pin 2) to 0, with the following actions:

- Remove the jumper JP7.
- Connect the CN7 pin 37 to GND.

Note: PC13 (tamper pin 1) connected to the blue push-button is not used in this demonstration firmware as it is connected to an external pull-down resistor preventing the use of the tamper detection on level with internal pull-up (the lowest consumption mode).

For more information, refer to the user manual *Evaluation board with STM32L476ZGT6 MCU* (UM1855).

3.3.2 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the «project.eww» file. For Keil® MDK-ARM, open the «project.uvprojx» file, compile and launch the debug session. For SW4STM32, open the SW4STM32 toolchain, browse to the SW4STM32 workspace directory, select and import the project RTC_TamperVBATT. Launching the debug session loads the program in the internal Flash memory and execute it.

In EWARM debug session, the user can view the calendar and tamper event timestamp (time and date) by opening a "Live watch" window (View -> Live watch) and observe the following global variables:

- aShowTime
- aShowDate
- aShowTimeStampTime
- aShowTimeStampDate

The same observation can be done using Keil/MDK-ARM development tools. To open "Watch1" window, do View->Watch Windows -> Watch1.

3.3.3 LED meaning

- LD1 (green): tamper event detected

  TFT displays: the Tamper date and the tamper time are different from their initialization state
- LD2 (orange): power-on reset occurred

  TFT displays: the tamper date and the tamper time are at their initialization state
- LD3 (red): error (RTC or RCC configuration error, backup registers not erased)

  TFT displays: "error occurred" is displayed
- LD4 (blue): reset occurred

  TFT displays: the tamper date and the tamper time are at their initialization state

3.3.4 Tampering detection during normal operation

Disconnect the debugger. A power-on reset can be generated by removing the JP17 jumper and placing it again on STlk.

Open PA0 (input is now floating), the LD1 lits showing that a tamper event is detected. On TFT, the Tampering date and Tampering time is updated with the timestamp of the Tamper event. Note that the user observes a long delay (up to 2 seconds) before the tamper event is detected. This is due to the trade-off in the tamper detection frequency (TAMPFREQ) chosen which ensures the lowest power consumption.

A reset can be applied by pushing the black reset button. The application is now able the detect a new tamper event.

3.3.5 Tampering detection when main power supply is off

If the user supplies the VBAT pin with the external CR1220 battery (JP12 on BAT position), the tamper event can be detected even if the main power supply is off.Proceed as follows:

- Remove the JP17 jumper.
- Open PA0.
- Tie PA0 to GND (this is simulating that the end-user takes care to close the box containing the electronics before supplying it again).
- Set JP17 on STlk.
- The LD1 lits showing that a tamper event has been detected during the power-down and on TFT, the tamper date and tamper time are updated with the timestamp of the tamper event

Note: If the user supplies the VBAT pin with 3 V (JP12 on VDD position), the RTC is no longer supplied when the main power supply is switched off. In this case, the application is no longer able to detect the tampering event while the main power supply is off.

4 STM32L4 API and digital smooth calibration application example

4.1 STM32Cube firmware libraries

The timer peripheral comes with:

- a firmware driver API abstracting the timer features for the end-user. Refer to stm32l4xx_hal_tim.c and stm32l4xx_hal_tim_ex.c files in \STM32Cube_FW_L4_Vx.y.z\Drivers\STM32L4xx_HAL_Driver\
- a set of example projects so that the user can quickly become familiar with the timer peripheral. Refer to the examples in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Nucleo\Examples\TIM

The RTC peripheral comes with:

- a firmware driver API abstracting the RTC features for the end-user. Refer to stm32l4xx_hal_rtc.c and stm32l4xx_hal_rtc_ex.c files in \STM32Cube_FW_L4_Vx.y.z\Drivers\STM32L4xx_HAL_Driver\
- a set of example projects so that the user can quickly become familiar with the RTC peripheral. Refer to the examples in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Nucleo\Examples\RTC

4.2 STM32Cube expansion firmware

This application note comes with the X-CUBE-RTC Expansion Package upon the STM32Cube firmware libraries.

This example shows an implementation of the digital smooth calibration feature of the RTC. It also uses the GPTIMER HAL API.

X-CUBE-RTC shows a concrete example where RTC_CLK is smoothly calibrated based on an external 1 Hz reference: STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\ Projects\STM32L476RG_Nucleo\RTC_SmoothCalib.

The firmware implements:

- a GP timer in Master mode using channel 1 in output compare mode. It uses an external trigger clock.
- a GP timer in Master mode using channel 1 in input capture mode and channel 2 in output compare mode. It is also configured to use the LSE clock thanks to the TIM option register.
- the RTC with the programmation of the CALR register
- GPIOs to connect the input 1 Hz reference clock and the “corrected” RTC_OUT_CALIB clock.

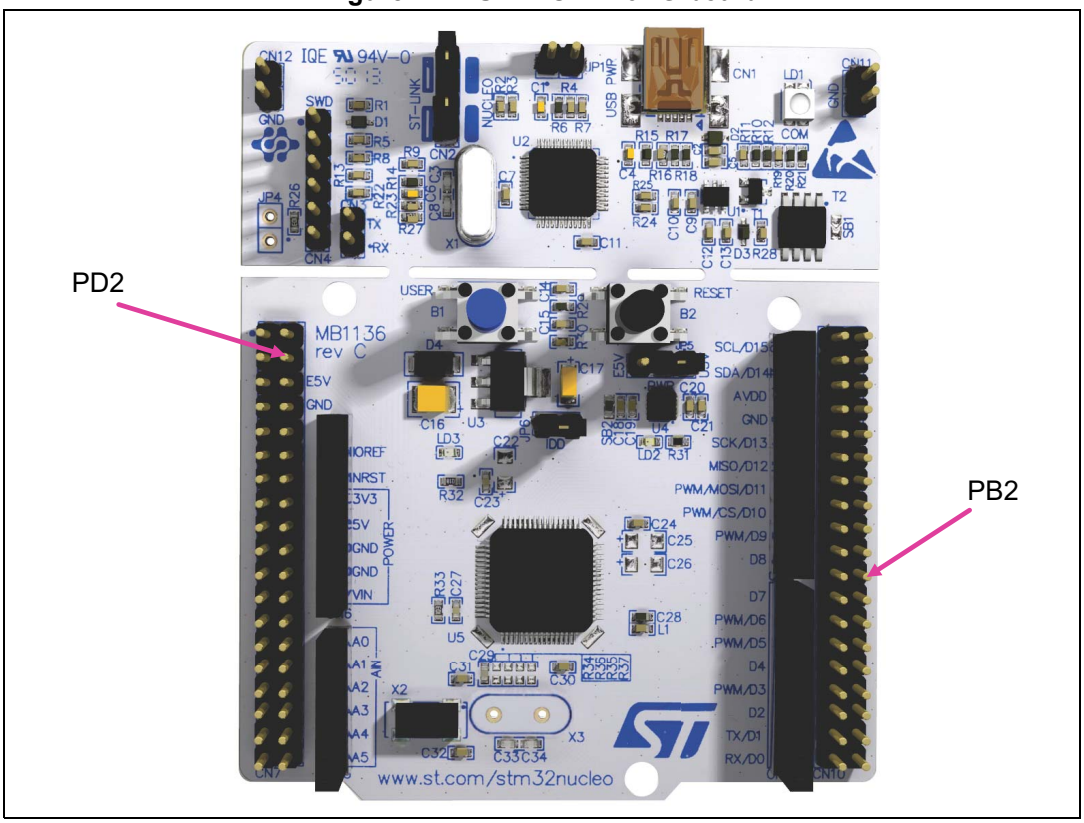
The application block diagram is shown in *Figure 21*.

4.3 Application example project

4.3.1 Hardware setup

In order to use the application example project, the user needs a STM32L476RG Nucleo board (NUCLEO-L476RG).

Figure 21. NUCLEO-L476RG board



1. This picture is not contractual.

Supply the board using the USB cable. For this example, the user needs to:

- connect PD2 to a signal generator driving 1 Hz clock with 3.3 V amplitude. It is advised to control the generated frequency at few ppm
- connect PB2 to an oscilloscope. Note that the accuracy of the frequency measurement is ideally in the range of 1 ppm

4.3.2 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the "RTC_SmoothCalib.eww" file. For Keil MDK-ARM, open the "RTC_SmoothCalib.uvprojx" file. For SW4STM32, open the SW4STM32 toolchain, browse to the SW4STM32 workspace directory, select and import the project RTC_SmoothCalib.

Compile and launch the debug session. It loads the program in the internal Flash memory and executes it.

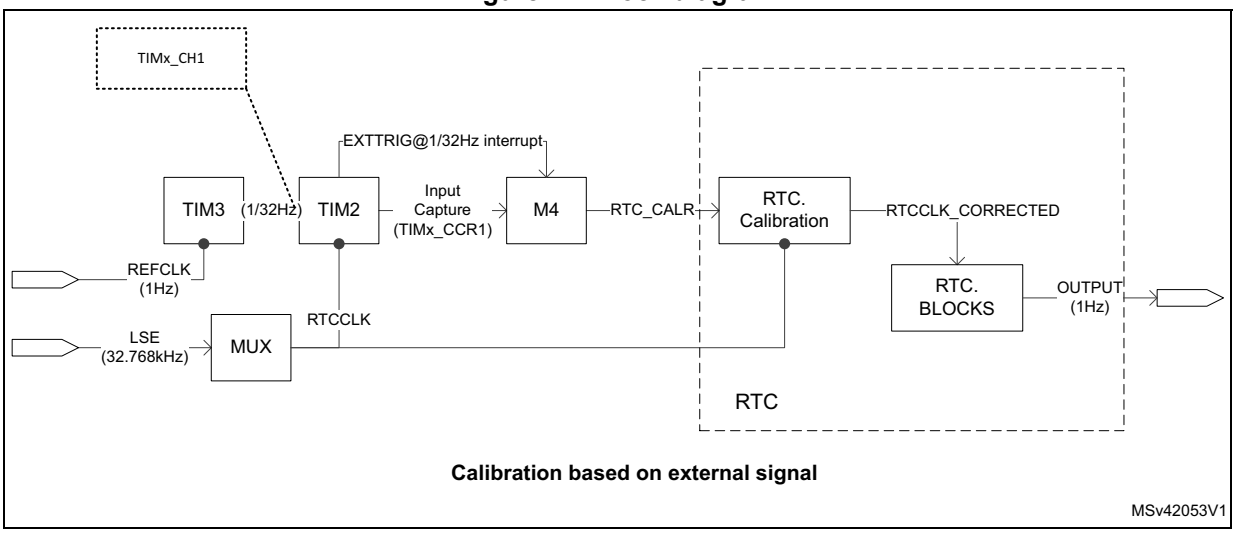
After a maximum of 64 seconds, make a frequency measurement of the CALIB_OUT clock on PB2. The user gets a frequency accurate at 1 ppm compared to the 1 Hz reference clock sent on PD2.

The user can then modify the PD2 clock by few ppm, wait 64 seconds and check that the PB2 clock has been changed accordingly.

Under debugger, the user can monitor the evolution of the CALR register fields: CALP and CALM.

4.3.3 Application block diagram

Figure 22. Block diagram



The application is based on the following four steps:

1. A 1 Hz reference clock is connected to TIM3 through the PD2 GPIO.
2. TIM3 is configured to generate a rising edge after 32 x 1 Hz reference rising edges. This gives an event every 32 seconds.
3. TIM2 is configured to increment its counter on CLK_RTC, to get the counter value on the rising edge generated by TIM3 and to store it in the TIM2_CCR1 register. It then reset itself.
4. The Cortex®-M4 core gets the TIM2_CCR1 value, compares it with the number of CLK_RTC cycles expected in 32 seconds and processes the comparison results to update the CALP and CALM fields of the RTC_CALR register.

The calibration is continuous.

4.3.4 Run time observations

The following steps are needed to check the correct functionality:

1. Modify the 1 Hz reference clock, for instance by forcing 1.0001 Hz.
2. Wait 2 x 32 seconds.
3. Measure the output frequency. Depending of the accuracy of the generator and the measurement device used, the user can see around 1 ppm accuracy.

The user can also monitor in Debug mode:

- the RTC_CALR register (CALM and CALP fields)
- the TIM2_CCR1 register (that contains the number of CLK_RTC cycles within a 32-s window)

5 STM32L0 API and tampering detection application example

5.1 STM32Cube firmware libraries

The RTC peripheral comes with:

- a firmware driver API abstracting RTC features for the end-user. Refer to stm32l0xx_hal_rtc.c and stm32l0xx_hal_rtc_ex.c files in \STM32Cube_FW_L0_Vx.y.z\Drivers\STM32L0xx_HAL_Driver\
- a set of example projects so that the user can quickly become familiar with the RTC peripheral. Refer to the examples in \STM32Cube_FW_L0_Vx.y.z\Projects\STM32L053R8-Nucleo\Examples\RTC

5.2 STM32Cube expansion firmware

This application note comes with the X-CUBE-RTC Expansion Package upon the STM32Cube firmware libraries.

X-CUBE-RTC shows a concrete example where a real-time clock must be maintained alive while using as less power as possible: STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\STM32L053R8-Nucleo\RTC_Tamper.

The firmware implements:

- a few hints order to ensure a low current consumption
- a display in the debugger (date and time, timestamp date and time) and on a single LED (power up events, reset events, tamper events, error)
- the tampering detection capability
- the ability to timestamp the tampering event
- the ability to erase the backup registers that may contain sensitive data upon tamper detection

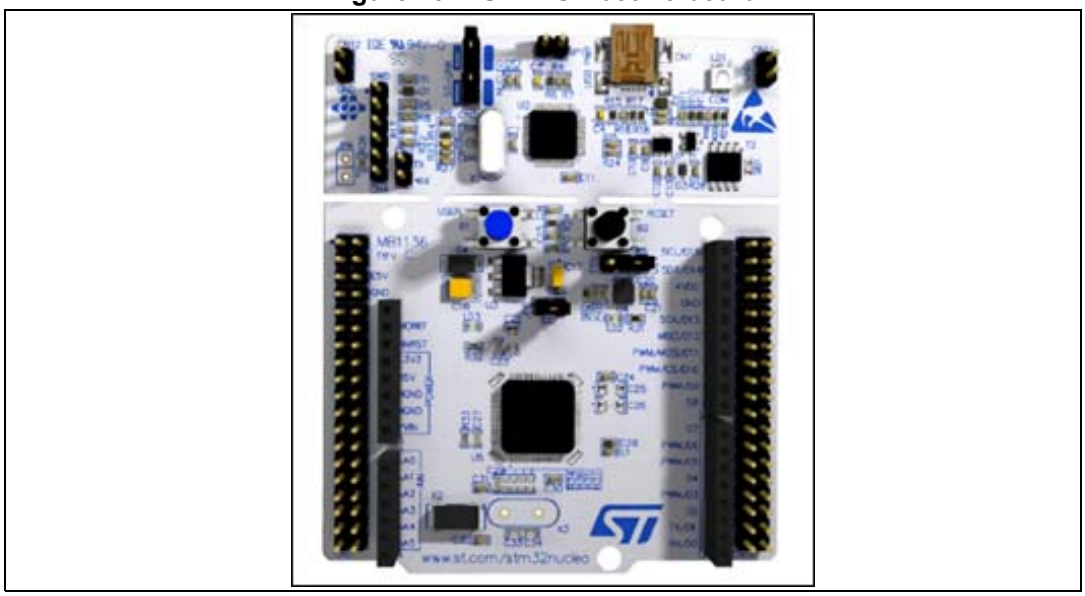
The application flowchart is shown in *Figure 19*.

5.3 Application example project

5.3.1 Hardware setup

In order to use the application example project, the user needs a STM32L053R8 Nucleo board (NUCLEO-L053R8).

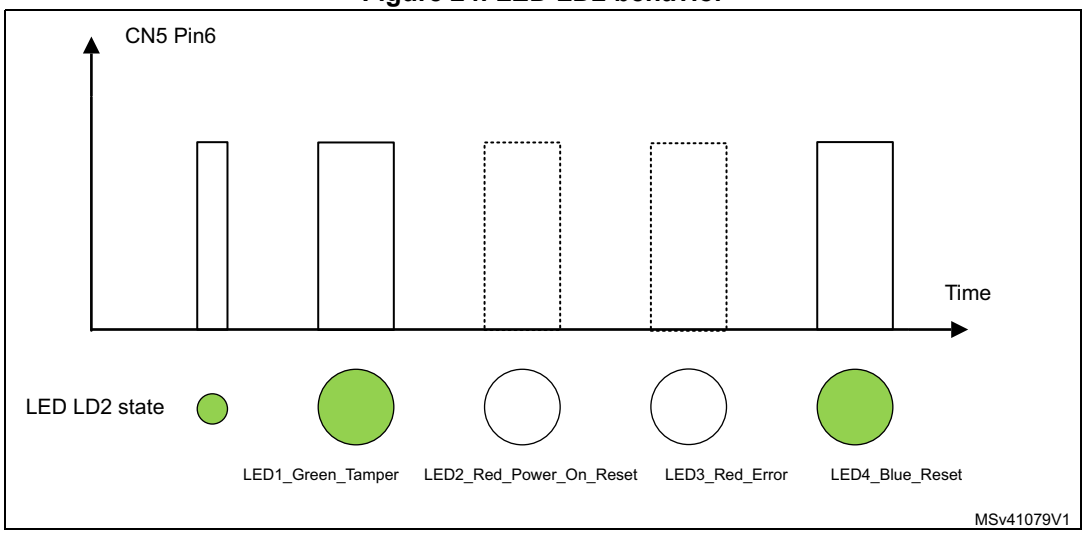
Figure 23. NUCLEO-L053R8 board



1. This picture is not contractual.

Supply the board using the USB cable. In this example, the user uses the B1 push button to simulate the external tamper event. The user can observe the software "private variables" on LED LD2 and CN5 Pin 6 (SCK/D13). Details regarding the LED LD2 behavior are available in main.c file.

Figure 24. LED LD2 behavior



5.3.2 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the "project.eww" file. For Keil MDK-ARM, open the "project.uvprojx" file. For SW4STM32, open the SW4STM32 toolchain, browse to the SW4STM32 workspace directory, select and import the project RTC_Tamper.

Compile and launch the debug session. This loads the program in the internal Flash memory and execute it.

In EWARM debug session, the user can view the calendar and tamper event timestamp (time and date) by opening a "Live watch" window (View -> Live Watch) and observe the following global variables:

- aShowTime
- aShowDate
- aShowTimeStampTime
- aShowTimeStampDate

The user can add and follow these "private variables":

- LED1_Green_Tamper_event_detected
- LED2_Red_Power_On_Reset_occurred
- LED3_Red_Error
- LED4_Blue_Reset_occurred

The same observation can be done using Keil/MDK-ARM development tools. To open "Watch1" window, do View->Watch Windows -> Watch1.

5.3.3 LED meaning

Only one LED (LD2) is available on the STM32L053R8 Nucleo board. To match with the STM32L4 examples, described in *Section 3: STM32L4 API and tampering detection application example*, LED1, LED2, LED3 and LED4 are replaced by the following four 4 integers:

- uint32_t LED1_Green_Tamper_event_detected:
  Set when tamper1 event is detected.
  Equivalent to LED1_Green in STM32L4 examples.
- uint32_t LED2_Red_Power_On_Reset_occurred:
  Set when power-on reset is detected.
  Equivalent to LED2_Red in STM32L4 examples.
- uint32_t LED3_Red_Error:
  Set in errors cases.
  Equivalent to LED3_Red in STM32L4 examples.
- uint32_t LED4_Blue_Reset_occurred:
  Set when reset is detected (B2 black push button).
  Equivalent to LED4_Blue in STM32L4 examples.

The user can observe these "private variables" using the live watch feature in the debugger.

The user can observe these "private variables" on LED LD2 and CN5 Pin 6 (SCK/D13). Regarding the LED LD2 behavior, see details in main.c file.

5.3.4 Tampering detection during normal operation

Step1: a power-on reset can be generated by disconnecting/connecting the USB cable on CN1.

Table 18. Tampering detection status when power-on reset is detected

Meaning	Status	LED LD2 (blink)
LED1_Green_Tamper_event_detected	0	OFF
LED2_Red_Power_On_Reset_occurred	1	ON
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

Step2: push B1 to simulate a tamper event.

Table 19. Tampering detection status when a tamper event is detected

Meaning	Status	LED LD2 (blink)
LED1_Green_Tamper_event_detected	1	ON
LED2_Red_Power_On_Reset_occurred	1	ON
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

Step3: push B2 to simulate a reset.

Table 20. Tampering detection status when reset is detected

Meaning	Status	LED LD2 (blink)
LED1_Green_Tamper_event_detected	0	OFF
LED2_Red_Power_On_Reset_occurred	0	OFF
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

Step4: push B1 to simulate a tamper event.

Table 21. Tampering detection status when tamper event is detected

Meaning	Status	LED LD2 (blink)
LED1_Green_Tamper_event_detected	1	ON
LED2_Red_Power_On_Reset_occurred	0	OFF
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

6 STM32L5 API and digital smooth calibration application example

6.1 STM32Cube firmware libraries

The timer peripheral comes with:

- a firmware driver API abstracting the timer features for the end-user. Refer to stm32l5xx_hal_tim.c and stm32l5xx_hal_tim_ex.c files in \STM32Cube_FW_L5_Vx.y.z\Drivers\STM32L5xx_HAL_Driver\
- a set of example projects so that the user can quickly become familiar with the timer peripheral. Refer to the examples in \STM32Cube_FW_L5_Vx.y.z\Projects\ NUCLEO-L552ZE-Q\Examples\TIM

The RTC peripheral comes with:

- a firmware driver API abstracting the RTC features for the end-user. Refer to stm32l5xx_hal_rtc.c and stm32l5xx_hal_rtc_ex.c files in \STM32Cube_FW_L5_Vx.y.z\Drivers\STM32L5xx_HAL_Driver\
- a set of example projects so that the user can quickly become familiar with the RTC peripheral. Refer to the examples in \STM32Cube_FW_L5_Vx.y.z\Projects\ NUCLEO-L552ZE-Q\Examples\RTC

6.2 STM32Cube expansion firmware

This application note comes with the X-CUBE-RTC Expansion Package upon the STM32Cube firmware libraries.

This example shows an implementation of the digital smooth calibration feature of the RTC.

It also uses the GPTIMER HAL API.

X-CUBE-RTC shows a concrete example where RTC_CLK is smoothly calibrated based on an external 1 Hz reference: STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\NUCLEO-L552ZE-Q\RTC_SmoothCalib.

The firmware implements:

- a GP timer in Master mode using channel 1 in output compare mode. It uses an external trigger clock
- a GP timer in Master mode using channel 1 in input capture mode and channel 2 in output compare mode. It is also configured to use the LSE clock thanks to the TIM option register
- the RTC with the programming of the CALR register
- GPIOs to connect the input 1 Hz reference clock and the “corrected” RTC_OUT CALIB clock

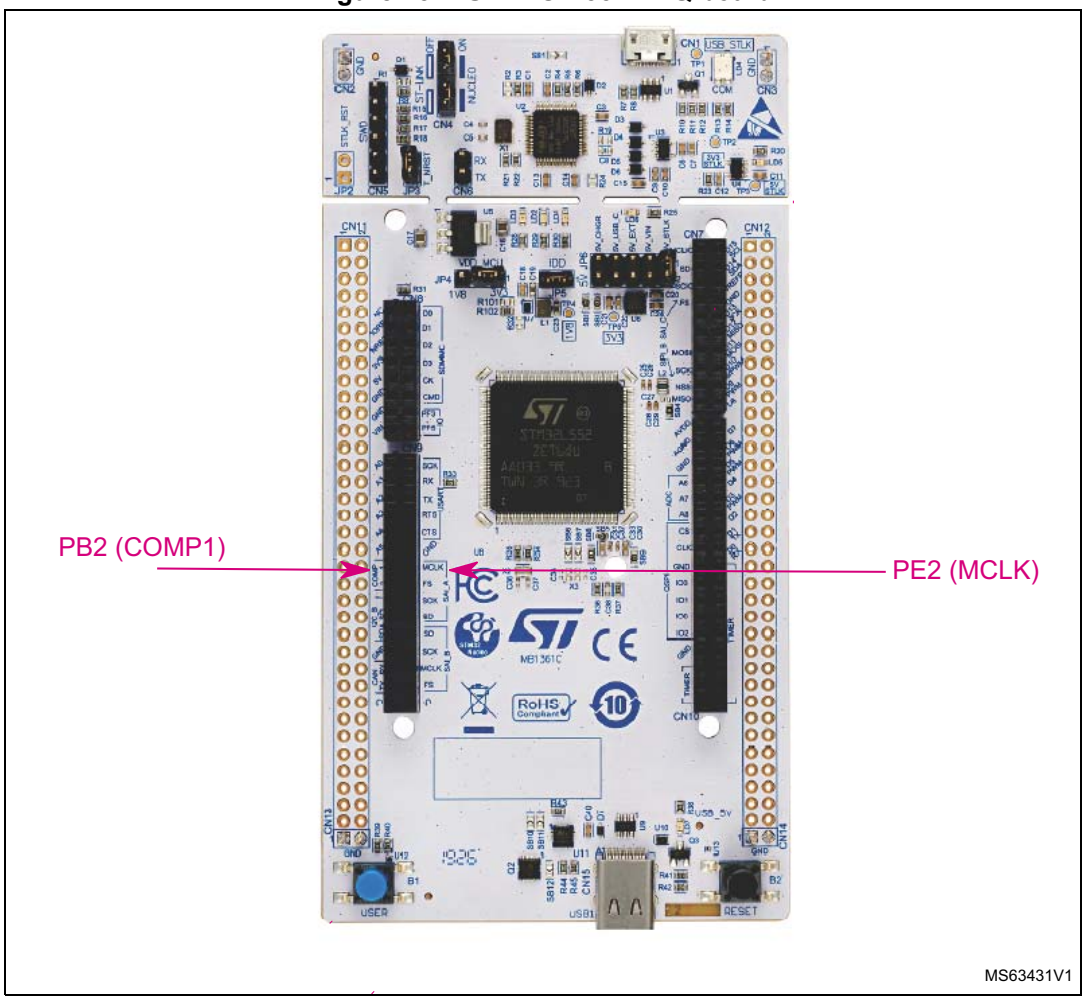
The application block diagram is shown in *Figure 26.*

6.3 Application example project

6.3.1 Hardware setup

In order to use the application example project, the user needs a STM32L552ZE-Q Nucleo board (NUCLEO-L552ZE-Q).

Figure 25. NUCLEO-L552ZE-Q board



1. This picture is not contractual.

Supply the board using the USB cable. For this example, the user needs to:

- connect PE2 to a signal generator driving 1 Hz clock with 3.3 V amplitude. It is advised to control the generated frequency at few ppm.
- connect PB2 to an oscilloscope. Note that the accuracy of the frequency measurement is ideally in the range of 1 ppm.

6.3.2 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the "RTC_SmoothCalib.eww" file. For Keil MDK-ARM, open the "RTC_SmoothCalib.uvprojx" file. For STM32CubeIDE, open the ".cproject" file.

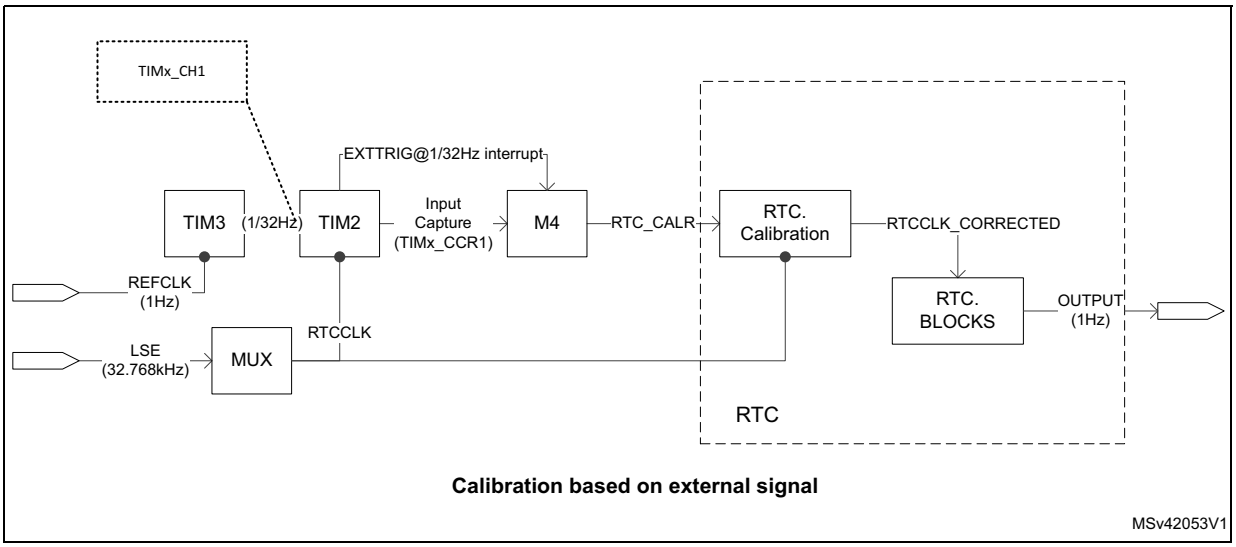
Compile and launch the debug session. It loads the program in the internal Flash memory and executes it.

After a maximum of 64 seconds, make a frequency measurement of the CALIB_OUT clock on PB2. The user gets a frequency accurate at 1 ppm compared to the 1 Hz reference clock sent on PE2.

The user can then modify the PE2 clock by few ppm, wait 64 seconds and check that the PB2 clock has been changed accordingly.

Under debugger, the user can monitor the evolution of the CALR register fields: CALP and CALM.

Figure 26. Application block diagram



The application is based on the following four steps:

1. A 1 Hz reference clock is connected to TIM3 through the PE2 GPIO.
2. TIM3 is configured to generate a rising edge after 32 x 1 Hz reference rising edges. This gives an event every 32 seconds.
3. TIM2 is configured to increment its counter on CLK_RTC, to get the counter value on the rising edge generated by TIM3 and to store it in the TIM2_CCR1 register. It then reset itself.
4. The Cortex®-M33 core gets the TIM2_CCR1 value, compares it with the number of CLK_RTC cycles expected in 32 seconds and processes the comparison results to update the CALP and CALM fields of the RTC_CALR register.

The calibration is continuous.

6.3.3 Run time observations

The following steps are needed to check the correct functionality:

1. Modify the 1 Hz reference clock, for instance by forcing 1.0001 Hz.
2. Wait 2 x 32 seconds.
3. Measure the output frequency. Depending of the accuracy of the generator and the measurement device used, the user can see around 1 ppm accuracy.

The user can also monitor in Debug mode:

- the RTC_CALR register (CALM and CALP fields)
- the TIM2_CCR1 register (that contains the number of CLK_RTC cycles within a 32-s window)

7 STM32L5 API and synchronization application example

7.1 STM32Cube firmware libraries

Refer to *Section 6.1: STM32Cube firmware libraries*.

7.2 STM32Cube expansion firmware

This application note comes with the X-CUBE-RTC Expansion Package upon the STM32Cube firmware libraries.

X-CUBE-RTC shows a concrete example where the RTC calendar is synchronized with the HSE clock (provided by an external signal: the HSE bypass feature is used): STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\NUCLEO-L552ZE-Q\RTC_Synchronization.

The firmware implements:

- TIMER1 in master mode with update interrupt activated to generate an interruption each second. It is configured to exploit an external 16 MHz signal thanks to the bypass of HSE clock. A PLL is also used for the timer to be clocked by a 100 MHz signal
- the RTC with the programming of the SHIFTR register
- GPIO: RTC_OUT CALIB to check if the RTC 1 Hz output's frequency is really 1 Hz (synchronization does not affect the RTC_OUT CALIB output but this signal needs to be as close as possible to 1 Hz for the synchronization to work). The user LED2 (PB7) allows to visualize the 1 second period and the good working of the application

7.3 Application example project

7.3.1 Hardware setup

In order to use the application example project, the user needs a STM32L552ZE-Q Nucleo board (NUCLEO-L552ZE-Q). Supply the board using a USB cable.

The TIMER1 clock source is HSE. However, X3 crystal associated to HSE is not implemented by default on the Nucleo board. Then, the application exploits the HSE bypass functionality of the STM32 which allows to input an external 16MHz signal to substitute the crystal (for this feature the user must have SB142 ON, SB145 ON, SB143 OFF, SB6 OFF, SB7 OFF). This signal must be generated on PH0 (position 29 on CN11, raw next to the connection A1 of CN9).

The internal HSI clock is not used since it is less precise than the LSE oscillator that clocks the RTC. Moreover, the HSE external 16 MHz signal must be more precise than the LSE.

The user can check that the RTC prescalers are well configured for his board by verifying a 1 Hz signal is output on PB2 (RTC_OUT CALIB output).

7.4 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the "RTC_Synchronization.eww" file. For Keil MDK-ARM, open the" RTC_Synchronization.uvprojx" file. For STM32CubeIDE, open the ".cproject" file.

Compile and launch the debug session. It loads the program in the internal Flash memory and executes it.

7.4.1 Application principle

The application is based on what follows:

1. TIMER1 is configured and launched so as to generate an interruption each second (APB2 timer clock = 100MHz).
2. RTC is configured and launched including its calendar (RTC clock = LSE).
3. The 1 second period counted by the TIMER1's interruptions allows a more precise time base than with RTC (the external HSE bypass signal must be more precise than LSE).
4. With this time base and thanks to the sub-second register, the advance/delay of the RTC clock can be analysed on the TIMER1 1 second period.
5. With the SHIFTR register, this advance or delay is compensated.

The synchronization is continuous.

7.4.2 Run time observations

The following is needed to check the correct functionality of the application:

1. Set the SYNCHRO_ACTIVATED to 0 and observe the time_elapsed variable deriving (getting away from 255/1sec). The user can monitor it in debug mode.
2. Set the SYNCHRO_ACTIVATED to 1 and observe that the time_elapsed variable derivation is less important.

Time_elapsed embodies the 1 second period measured for RTC.

Note: A typical quartz has an uncertainty of 53 s/month, this must be attenuated with synchronization that is efficient if the clock used (external 16 MHz in the present case) has a better uncertainty. So using HSE X3 crystal is not so efficient if the LSE crystal has the same uncertainty.

To check that the application is running properly, the user can also visualize the user LED2 blinking each second and the RTC_OUT CALIB output on the PB2 pin (these two observations do not witness the efficiency of the synchronization but can help the user to understand what happens).

8 STM32L5 API and reference clock detection application example

8.1 STM32Cube firmware libraries

Refer to *Section 6.1: STM32Cube firmware libraries*.

8.2 STM32Cube expansion firmware

This application note comes with the X-CUBE-RTC Expansion Package upon the STM32Cube firmware libraries.

X-CUBE-RTC shows a concrete example where the RTC calendar is synchronized with an external 50/60 Hz signal: STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\NUCLEO-L552ZE-Q\RTC_RefClockDetection.

The firmware implements:

- TIMER1 in master mode with update interrupt activated to generate an interruption each second. It is configured to use the HSI clock through a PLL to be clocked by a 100 MHz signal (HSI is less precise than the LSE clocking the RTC but allows the measure of the RTC period without the need to inject another signal than the 50/60 Hz one)
- the RTC with the activation of the reference clock detection feature
- GPIO: RTC_OUT CALIB (PB2) to check if the RTC 1 Hz output frequency is really 1 Hz. The user LED2 (PB7) to visualize the 1 second period and the good working of the application. The RTC_REFIN pin (PB15) to host the 50/60 Hz signal used as the reference clock detected

8.3 Application example project

8.3.1 Hardware setup

In order to use the application example project, the user needs a STM32L552ZE-Q Nucleo board (NUCLEO-L552ZE-Q). Supply the board using the USB cable.

As indicated by the application name, a reference clock must be delivered to the STM32 for this application. For this goal, the user has to take a 50/60 Hz signal more precise than the LSE clock of the Nucleo board, and to inject it on the PB15 pin (position 26 on CN12 of the Nucleo board).

The user can check that the RTC prescalers are well configured for his board by verifying a 1 Hz signal is output on PB2 (RTC_OUT CALIB output).

8.3.2 Software setup

Open the project under the user favorite integrated development environment. For IAR EWARM, open the "RTC_Synchronization.eww" file. For Keil MDK-ARM, open the “RTC_Synchronization.uvprojx" file. For STM32CubeIDE, open the “.cproject” file.

Compile and launch the debug session. It loads the program in the internal Flash memory and executes it.

8.3.3 Application principle

The application is based on what follows:

1. TIMER1 is configured and launched so as to generate an interruption each second (APB2 timer clock = 100 MHz).
2. RTC is configured and launched with its calendar and reference clock detection feature (RTC clock = LSE).
3. When the 50/60 Hz signal is available on PB15, the RTC automatically aligns the rising edges of this reference clock with the ones of the output of the synchronous RTC prescaler (1 Hz clock). Thus, the RTC 1 Hz clock is synchronized with the signal provided by the user.
4. The 1 second period counted by the TIMER1 interruptions allows the measure of the RTC period. However, HSI which is TIMER1 clock source is less precise than the LSE which is the RTC clock source, which is synchronized with an even more precised reference clock by the way. Thus, this measure is made for testing the application.

The reference clock detection and synchronization are continuous. If the reference is lost, the RTC is still clocked by LSE.

8.3.4 Run time observations

The following is needed to check the correct functionality of the application:

1. Set the CLK_REF_ACTIVATED to 0 and observe the time_elapsed variable deriving (getting away from 255/1sec). The user can monitor it in debug mode.
2. Set the CLK_REF _ACTIVATED to 1 and observe that the time_elapsed variable derivation is less important.

Time_elapsed embodies the 1 second period measured for RTC.

Note: A typical quartz has an uncertainty of 53 s/month, this must be attenuated with the reference clock detection and synchronization.

Moreover, to check that the application is running properly, the user can also visualize the user LED2 blinking each second and the RTC_OUT CALIB output on PB2 pin (these two observations do not witness the efficiency of the reference synchronization).

9 STM32L5 API and internal tamper detection application example

9.1 Internal tamper detection firmware example presentation

A firmware example for internal tamper detection is developed and included in the X-CUBE-RTC Expansion Package associated to this application note:

STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\NUCLEO-L552ZE-Q\RTC_InternalTamperCalOvf

This example uses the RTC firmware driver API mentioned in *Section 6*, *Section 7* and *Section 8*. It is developed for the NUCLEO-L552ZE-Q board and exploits the ITAMP5 signal associated to a RTC calendar overflow event.

The user must supply the Nucleo board via USB, then compile and run the firmware with the preferred IDE. For IAR EWARM, open the "RTC_Synchronization.eww" file. For Keil MDK-ARM, open the "RTC_Synchronization.uvprojx" file. For STM32CubeIDE, open the ".cproject" file.

The firmware simply configures the RTC calendar at 23:59:30 on the 31st of december of the year xx99 and activates the internal tamper 5. The user can choose the polling mode or interrupt mode with the ITAMP_INTERRUPT constant in main.h: 1 is interrupt mode, 0 is polling mode.

When the seconds pass 59, the internal tamper event occurs and the green user led 1 PC7 if in polling mode, or the blue user led 2 PB7 if in interrupt mode, is turned on.

The RTC_DR and RTC_TR registers allow the evolution of the RTC calendar to be followed in debug mode.

The user can easily modify this example to exploit all the others internal tamper event sources.

10 Revision history

Table 22. Document revision history

Date	Revision	Changes
26-May-2016	1	Initial release.
25-Oct-2016	2	– Updated *Table 1: Applicable products* adding product series and part number. – Updated *Figure 3: STM32F2, STM32F4, STM32L0 and STM32L1 Series RTC clock sources*. – Updated *Figure 4: STM32F0, STM32F3, STM32F7, STM32L4 and STM32WB Series RTC clock sources* – Updated *Table 3: Calendar clock equal to 1 Hz with different clock sources*. – Updated *Section 1.4: Digital smooth calibration* adding the RTC calibration basics. – Updated *Table 14: Advanced RTC features*. – Updated *Section 4.3.3: LED meaning*. – Added *Section 5: STM32L4 API and digital smooth calibration application example*. – Updated *Section 6.3.1: Hardware setup*. – Updated *Section 6.3.2: Software setup*. – Updated *Section 7: Reference documentation*.
11-May-2017	3	– Updated *Figure 4: STM32F0, STM32F3, STM32F7, STM32L4 and STM32WB Series RTC clock sources*. – Updated *Table 3: Calendar clock equal to 1 Hz with different clock sources* adding note. – Updated *Section 1.4.1: RTC calibration basics*. – Updated *Section 1.11.1: RTC register write protection*. – Updated *Table 14: Advanced RTC features* RTC calibration for STM32F2 Series.
11-Feb-2019	4	Updated: – *Table 1* with the STM32WB Series – *Section 1: Overview of the STM32 MCUs advanced RTC* – *Figure 4: STM32F0, STM32F3, STM32F7, STM32L4 and STM32WB Series RTC clock sources* – *Advanced RTC features table* Removed Section 7. Reference documentation.

Table 22. Document revision history (continued)

Date	Revision	Changes
12-Feb-2020	5	Updated cover Updated *Table 1: Applicable products* Updated *Section 1: Overview of the STM32 MCUs advanced RTC*: – Adding *Table 2: RTC/TAMP types versus features* – Adding*Table 3: RTC type versus STM32 products* – Updating *Table 4: Advanced RTC2 features* – adding *Table 5: Advanced RTC3 features* Updated: – *Table 6: Steps to initialize the calendar* – Foonote *Table 7: Calendar clock equal to 1 Hz with different clock sources* – *Section 2.1.4: RTC clock configuration* – *Table 8: Steps to configure the alarm* – *Section 2.3.2: Alarm sub-second configuration* – *Section 2.4: RTC periodic wakeup unit* – *Table 11: Steps to configure the auto-wakeup unit* – *Section 2.5.2: Methodology* paragraph and *Figure 10: Smooth calibration block for RTC3 with LPCAL=1* – *Section 2.9: Timestamp function* – *Section 2.10: RTC tamper detection function* – *Section 2.11: Backup registers* – *Section 2.12: Alternate function RTC outputs* – *Section 2.13: RTC safety aspects*. – *Section 2.14: Reducing power consumptionSection 2.13: RTC safety aspects* Added: – *Section 2.1.5: Calendar firmware examples* – *Section 2.4.3: Wakeup firmware examples* – *Section 2.8: RTC prescaler adjustment with LSI measurements* – *Section 2.10.4: Active tamper detection (RTC3 only)* – *Figure 17: Tamper detection* – *Section 2.10.6: Tamper detection firmware examples* – *Section 6: STM32L5 API and digital smooth calibration application example* – *Section 7: STM32L5 API and synchronization application example* – *Section 8: STM32L5 API and reference clock detection application example* – *Section 9: STM32L5 API and internal tamper detection application example*

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to *www.st.com/trademarks*. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved