Introduction to using the hardware real-time clock (RTC) and the tamper management unit (TAMP) with STM32 MCUs

Introduction

A real-time clock (RTC) is a computer clock that tracks 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.

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

Depending on the RTC type, the product documentation may refer to an independent TAMP that is also detailed in this application note.

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

The dedicated software is provided through the X-CUBE-RTC Expansion Package delivered with this application note. This software contains the source code of these examples and all the embedded software modules required to run the examples.

In this document, the STM32 microcontroller terminology applies to the products listed in the table below.

Table 1. Applicable products

Type	Products
Expansion Package	X-CUBE-RTC
Microcontrollers	STM32C0 series, STM32F0 series, STM32F2 series, STM32F3 series, STM32F4 series, STM32F7 series, STM32G0 series, STM32G4 series, STM32H5 series, STM32H7 series, STM32L0 series, STM32L1 series, STM32L4 series, STM32L4+ series, STM32L5 series, STM32N6 series, STM32U0 series, STM32U3 series, STM32U5 series, STM32WB series, STM32WB0 series, STM32WBA series, STM32WL series

1 Overview of the STM32 MCUs advanced RTC

The RTC is embedded in STM32 Arm® Cortex®-based MCUs.

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

The RTC provides a full-featured calendar, alarm, periodic wakeup unit, digital calibration, synchronization, timestamp, and advanced tamper detection.

The RTC features and their implementation can be significantly different (regarding the registers map for example) depending on the product. Two RTC types, RTC2 and RTC3, are distinguished and characterized in the table below. These types affect the information presented in the rest of this application note.

Table 2. RTC/TAMP types

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 h/24 h format	X	X
		Hour, minute, second	X	X
		Subsecond	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 h/24 h format	X	X
		Hour, minutes, seconds	X	X
		Subseconds	X	X
	Date or week day		X	X
	Binary mode alarm		-	X
Tamper detection	Tamper effects	Backup registers erase(1)	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
	V_{BAT} mode pins		X	X
	NOERASE mode		X	X

Features			RTC2	RTC3
Tamper detection	Independent watchdog linked to an internal tamper for potential detection timeout		-	X
	Active tamper		-	X
	Monotonic counter		-	X
	SAES key storage		-	X
	Secure protection modes		-	X
	Privileged protection mode		-	X
Timestamp	Configurable input mapping		X	X
	Time	Hour, minute, second	X	X
		Subsecond	X	X
	Date (day, month)		X	X
	Timestamp on tamper detection event		X	X
	Timestamp on switch to V_{BAT} 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
RTC synchronization			X	X
Reference clock detection			X	X
Backup registers	Reset on a tamper detection		X	X
	Reset when Flash memory readout protection is disabled		X	X
RTC secure protection mode			-	X
RTC privileged protection mode			-	X

1. *Depending on devices, other resources can also be erased.*

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

A given feature can be available for a RTC type but not implemented on all products (see Table 4 and Table 5 for more details).

The table below specifies which RTC type is implemented on which product.

Table 3. RTC type on STM32 MCUs

RTC type	STM32 MCUs
RTC2	STM32F0 series, STM32F2 series, STM32F3 series, STM32F4 series, STM32F7 series, STM32H72/H73/H74/H75xxx, STM32L0 series, STM32L1 series, STM32L43/L44/L45/46/47/48xxx, STM32L4R5/Q5 line, STM32L4R7/S7 line, STM32L4R9/S9 line, STM32WB series, STM32WB0 series
RTC3	STM32C0 series, STM32G0 series, STM32G4 series, STM32H5 series, STM32H7A3/7B3 line, STM32H7Rx/7Sx line, STM32L41/L42xxx, STM32L4P5/Q5 line, STM32L5 series, STM32N6 series, STM32U0 series, STM32U3 series, STM32U5 series, STM32WL series, STM32WBA series

2 Advanced RTC features

The following tables summarize the RTC features available on each STM32 MCU.

Table 4. Advanced features for RTC2 type

Features			STM32F0	STM32F2	STM32F3	STM32F4	STM32F7	STM32L0	STM32L1		STM32L4[1] - STM32L4+[2]	STM32H72/H73/ H74/H75	STM32WB	STM32WB0
									Cat. 2/3/4/5/6	Cat. 1				
RTC clock source (LSE, LSI, HSE with prescaler)			X	X	X	X	X	X	X	X	X	X	X	X
Prescalers	Asynchronous (number of bits)		X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)
	Synchronous (number of bits)		X (15)	X (13)	X (15)	X (15)	X (15)	X (15)	X (15)	X (13)	X (15)	X (15)	X (15)	X (15)
Calendar	Time	12 h/24 h format	X	X	X	X	X	X	X	X	X	X	X	X
		Hour, minute, second	X	X	X	X	X	X	X	X	X	X	X	X
		Subsecond	X	N/A	X	X	X	X	X	N/A	X	X	X	X
	Date		X	X	X	X	X	X	X	X	X	X	X	X
	Daylight operation		X	X	X	X	X	X	X	X	X	X	X	X
	Bypass shadow registers		X	N/A	X	X	X	X	X	N/A	X	X	X	X
	V_{BAT} mode		X	X	X	X	X	N/A	N/A	N/A	X	X	X	X
Alarm	Alarms available	Alarm A	X	X	X	X	X	X	X	X	X	X	X	X
		Alarm B	N/A	X	X	X	X	X	X	X	X	X	X	N/A
	Time	12 h/24 h format	X	X	X	X	X	X	X	X	X	X	X	X
		Hour, minute, second	X	X	X	X	X	X	X	X	X	X	X	X
		Subsecond	X	N/A	X	X	X	X	X	N/A	X	X	X	X
	Date or week day		X	X	X	X	X	X	X	X	X	X	X	X
Tamper detection	Configurable input mapping		X	X	X	X	X	X	N/A	N/A	X	X	X	X
	Configurable edge detection		X	X	X	X	X	X	X	X	X	X	X	N/A
	Configurable level detection (filtering, sampling, and precharge configuration on tamper input)		X	N/A	X	X	X	X	X	N/A	X	X	X	N/A
	Number of tamper inputs		2[3]	2	2	2	3	3	3	1	3	3	3	N/A
	Number of tamper events		0	1	2	2	3	3	3	1	3	3	3	N/A
	V_{BAT} mode pins (inputs)		1	1	1	2	2	N/A	N/A	N/A	3	3	3	N/A
Timestamp	Configurable input mapping		X	X	X	X	X	X[4]	N/A	N/A	X[4]	X	X[4]	N/A
	Time	Hour, minute, second	X	X	X	X	X	X	X	X	X	X	X	N/A
		Subseconds	X	N/A	X	X	X	X	X	N/A	X	X	X	N/A
	Date (day, month)		X	X	X	X	X	X	X	X	X	X	X	N/A
	Timestamp on tamper detection event		X	X	X	X	X	X	X	X	X	X	X	N/A

Features			STM32F0	STM32F2	STM32F3	STM32F4	STM32F7	STM32L0	STM32L1		STM32L4[1] - STM32L4+[2]	STM32H72/H73/ H74/H75	STM32WB	STM32WB0
									Cat. 2/3/4/5/6	Cat. 1				
Timestamp	Timestamp on switch to V_{BAT} mode		N/A	N/A	N/A	N/A	X	N/A	N/A	N/A	X	X	X	N/A
RTC outputs	RTC_ ALARM	Alarm event	X	X	X	X	X	X	X	X	X	X	X	X
		Wakeup event	X	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	X
		1 Hz	X	N/A	X	X	X	X	X	N/A	X	X	X	X

Features		STM32F0	STM32F2	STM32F3	STM32F4	STM32F7	STM32L0	STM32L1		STM32L4[1] - STM32L4+[2]	STM32H72/H73/ H74/H75	STM32WB	STM32WB0
								Cat. 2/3/4/5/6	Cat. 1				
RTC calibration	Coarse calibration	N/A[5]	X	N/A[5]	X	N/A[5]	N/A[5]	X	X	N/A[5]	N/A[5]	N/A[5]	N/A[5]
	Smooth calibration	X	N/A	X	X	X	X	X	N/A	X	X	X	X
Synchronizing the RTC		X	N/A	X	X	X	X	X	N/A	X	X	X	X
Reference clock detection		X	X	X	X	X	X	X	X	X	X	X	X
Backup registers	Powered-on V_{BAT}	X	X	X	X	X	N/A	N/A	N/A	X	X	X	VDD12
	Reset on a tamper detection	X	X	X	X	X	X	X	X	X	X	X	N/A
	Reset when flash memory readout protection is disabled	X	N/A	X	N/A	X	X	X	X	X	X	X	N/A
	Number of backup registers	5	20	16	20	32	5	32[6]	20	20[7]	32	20	5

1. *Except STM32L41/42xxx.*
2. *Except STM32L4P5/Q5 line.*
3. *3 inputs for STM32F07/F09x.*
4. *Thanks to timestamp on tamper event.*
5. *Obsolete, replaced by smooth calibration.*
6. *Only 20 for Cat 2.*
7. *32 for STM32L4R/4S line.*

Table 5. Advanced features of RTC3 type

Features			STM32C0	STM32G0	STM32G4	STM32H5	STM32H7A3/7B3	STM32L41/42/4P5/4Q5	STM32L5	STM32N6	STM32U5	STM32U3	STM32WL	STM32U0	STM32H7Rx/7Sx	STM32WBA
RTC clock source (LSE, LSI, HSE with prescaler)			X	X	X	X	X	X	X	X	X	X	X	X	X	X
Binary mode			N/A	N/A	N/A	X	N/A	N/A	N/A	X	X	X	X	X	N/A	X
Mixed mode (BCD and binary)			N/A	N/A	N/A	X	N/A	N/A	N/A	X	X	X	X	X	N/A	X
Prescalers	Asynchronous (number of bits)		X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)	X (7)
	Synchronous (number of bits)		X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)	X (15)
Calendar	Time	12 h/24 h format	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Hour, minute, second	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Subsecond	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Date		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Daylight operation		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Bypass the shadow registers		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Power optimization mode		N/A	N/A	N/A	X	N/A	X	X	X	X	X	X	X	X	X
Alarm	Alarms available	Alarm A	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Alarm B	N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
	Time	12 h/24 h format	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Hour, minute, second	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Subsecond	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Date or week day		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Binary mode alarm		N/A	N/A	N/A	X	N/A	N/A	N/A	X	X	X	X	X	X	X
Tamper detection	Tamper reactions	Backup registers erasing	N/A	X	X	X[2]	X[1]	X	X[1]	X[2]	X[2]	X[2]	X[1]	X[1]	X[1]	X[2]
		Interruption	N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
		Trigger for low-power timer	N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
	Configurable edge detection		N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
	Configurable level detection[3]		N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
	Number of internal tamper events		N/A	4	4	13	7	N/A	5	10	11	9	4	4	11	9
	Number of external tamper inputs		N/A	2	3	8 (11 pins)	3	3 [4]	8	7	8	5	3	5	8	6 (6 pins)
	V_{BAT} mode pins		N/A	N/A	N/A	TAMP_IN[8:1], TAMP_OUT[8:1], RTC_TS, RTC_OUT1/2	RTC_TS, RTC_OUT1/2	RTC_TAMP1/2/3, RTC_TS, RTC_OUT for STM32L441/42. RTC_TS, RTC_OUT1 for STM32L4P5/Q5	TAMP_IN1/2/3, TAMP_OUT2	TAMP_IN[4:1], TAMP_OUT[2:1], RTC_OUT1, RTC_TS	TAMP_IN[8:1], TAMP_OUT[8:1], RTC_TS, RTC_OUT1/2	RTC_OUT1, RTC_TS, TAMP_IN[5:1]	RTC_TS, RTC_OUT1	TAMP_IN[5:0], RTC_TS, RTC_OUT1	TAMP_IN[3:1], TAMP_OUT2, RTC_TS, RTC_OUT1/2	N/A
	NOERASE mode		N/A	N/A	N/A	X	N/A	X	N/A	X	X	X	N/A	N/A	N/A	X

Features			STM32C0	STM32G0	STM32G4	STM32H5	STM32H7A3/7B3	STM32L41/42/4P5/4Q5	STM32L5	STM32N6	STM32U5	STM32U3	STM32WL	STM32U0	STM32H7Rx/7Sx	STM32WBA
Tamper detection	Independent watchdog linked to an internal tamper for false detection timeout		N/A	N/A	N/A	X	N/A	N/A	N/A	X	X	X	N/A	N/A	N/A	X
	Active tamper		N/A	N/A	N/A	X	X	N/A	X	X	X	X	N/A	N/A	X	X
	Monotonic counter		N/A	N/A	N/A	X	X	N/A	X	X	X	X	X	N/A	X	X
	LP mode effect on TAMP	Sleep	N/A	No effect												
		Stop	N/A	No effect except for filtered level detection and active tamper if clock source is not LSE or LSI												
		Standby	N/A													
		Shutdown	N/A	No effect except for filtered level detection and active tamper if clock source is not LSE		N/A	No effect except for filtered level detection and active tamper if clock source is not LSE		N/A	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 or LSI
	SAES key storage protection		N/A	N/A	N/A	X	N/A	N/A	N/A	X	X	X	N/A	N/A	N/A	X
	Secure protection modes		N/A	N/A	N/A	X[6]	N/A	N/A	X	X	X	X	N/A	N/A	N/A	X
	Privileged protection mode		N/A	N/A	N/A	X[6]	N/A	N/A	X	X	X	X	N/A	N/A	N/A	X
Timestamp	Configurable input mapping		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Time	Hour, minute, second	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Subsecond	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Date (day, month)		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Timestamp on tamper detection event		N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
	Timestamp on switch to VBAT mode		N/A	X	X	X	X	X	X	X	X	X	X	X	X	N/A
Wakeup unit	Clock source available		N/A	RTC clock divided by 2, 4, 8, 16 or RTC synchronous prescaler output												
	Hardware automatic flag clearing		N/A	N/A	N/A	X	N/A	X	X	X	X	X	X	X	X	X
RTC outputs	Number of outputs		2	2	2	2	2	2	2	2	2	X	2	2	2	2
	TAMP ALRM	Alarm event	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Wakeup event	N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
		Tamper event	N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
	CALIB	512 Hz	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		1 Hz	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Digital calibration	Smooth calibration		X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Ultra-low-power mode		N/A	N/A	N/A	X	N/A	X	X	X	X	X	X	X	N/A	X
RTC synchronization			X	X	X	X	X	X	X	X	X	X	X	X	X	X
Reference clock detection			X	X	X	X	X	X	X	X	X	X	X	X	X	X
Backup registers	Powered on V_{BAT}		N/A	X	X	X	X	X	X	X	X	X	X	X	X	N/A
	Reset on a tamper detection		N/A	X	X	X	X	X	X	X	X	X	X	X	X	X
	Reset when flash memory readout protection is disabled		N/A	X	X	X	X	X	X	X	X	X	X	X	X	X

Features		STM32C0	STM32G0	STM32G4	STM32H5	STM32H7A3/7B3	STM32L41/42/4P5/4Q5	STM32L5	STM32N6	STM32U5	STM32U3	STM32WL	STM32U0	STM32H7Rx/7Sx	STM32WBA
Backup registers	Number of backup registers (size in bits)	4 (32) (within PWR)	5 (32)	32 or 16 (32)[5]	32 (32)	32 (32)	32 (32)	32 (32)	32 (32)	32 (32)	32 (32)	20 (32)	9 (32)	32 (32)	32 (32)
LP mode effect on RTC	Sleep	RTC interrupts may exit sleep mode	No effect										No effect; RTC interrupts may exit Sleep mode		
	Stop	Active if RTC is clocked by LSE or LSI	Active if RTC is clocked by LSE or LSI												
	Standby	RTC powered down and must be reinitialized													
	Shutdown	N/A	Active if RTC is clocked by LSE		N/A	Active if RTC is clocked by LSE			N/A	Active if RTC is clocked by LSE			N/A	Active if RTC is clocked by LSE	N/A
RTC secure protection mode		N/A	N/A	N/A	X[6]	N/A	N/A	X	X	X	X	N/A	N/A	N/A	X
RTC privilege protection mode		N/A	N/A	N/A	X[6]	N/A	N/A	X	X	X	X	N/A	N/A	X	X

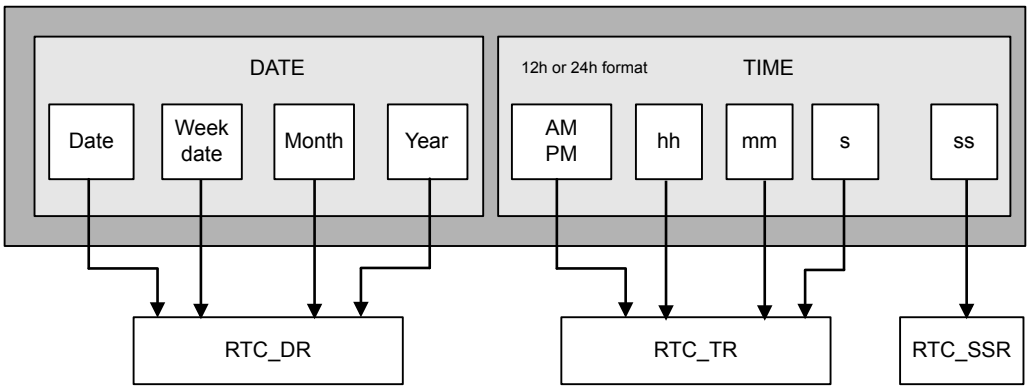
1. *Erasing part of SRAM is also possible.*
2. *Erasing of part of SRAM, caches and cryptographic registers is also possible.*
3. *Filtering, sampling and precharge configuration, internal pull-up.*
4. *Only two for STM32L4 devices.*
5. *Depends on device category.*
6. *This feature is not available for STM32H503.*

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 subseconds (not programmable), seconds, minutes, hours (12 h/24 h format), day of the week (day), day of the month (date), month, year
- Calendar in 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**



Note: RTC_DR and RTC_TR are RTC date and time registers. RTC_SSR (subsecond) gives the value of the synchronous prescaler counter (read only). In binary mode (when available), RTC_SSR is not the value of the synchronous prescaler counter but of the asynchronous one.

2.1.1 Software calendar

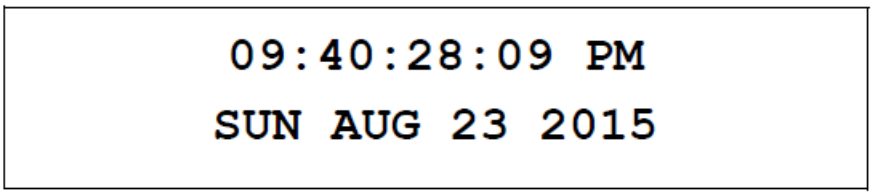
A software calendar is a software counter (usually 32-bit long) that represents the number of seconds. The software routines convert the counter value to hours, minutes, day of the month, day of the week, month and year. This data can be converted to BCD format and displayed on a standard LCD. Conversion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

the table below describes the steps required to correctly configure the calendar time and date.

Table 6. Steps to initialize the calendar

Step	What	How	Comments
1	Disable the RTC registers write protection.	Write 0xCA and then 0x53 into RTC_WPR.	RTC registers can be modified.
2	Enter initialization mode.	Set INIT = 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 two RTCCLK clock cycles due to clock synchronization. • For RTC3: If LPCAL = 0, INITF is set around two RTCLK cycles after INIT is set. If LPCAL = 1, INITF is set up to two ck_apre cycles after INIT is set.
4	Program the prescaler values (if needed).	In RTC_PRER, write first the synchronous value, and then write the asynchronous value. For RTC3, program also BIN and BCDU in RTC_ICSR, if in binary or mixed mode.	By default (in BCD mode for RTC3), RTC_PRER is 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.	-
6	Configure the time format (12 h or 24 h)	Set FMT bit in RTC_CR.	If FMT = 0, the format is 24 hour/day. If FMT = 1, the format is 12 h am/pm.
7	Exit initialization mode.	Clear INIT in RTC_ISR (RTC2)/ RTC_ICSR (RTC3).	For RTC2: The current calendar counter is automatically loaded and the counting restarts after four RTCCLK clock cycles. For RTC3: If LPCAL = 0, the counting restarts after four RTCCLK clock cycles. If LPCAL = 1, the counting restarts after up to two RTCCLK + 1 ck_apre cycles.
8	Enable write protection of the RTC registers.	Write 0xFF into RTC_WPR.	The 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 the HSE is selected, a prescaler must be selected. The user can refer to the product reference manual to know its possible values and configurations.

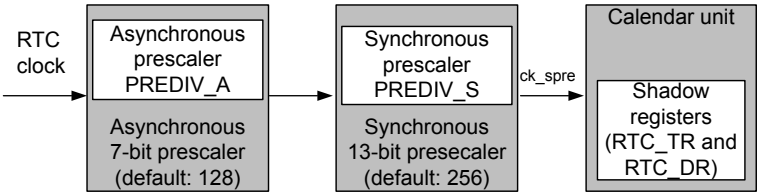
Moreover, the choice of the RTC clock source is done thanks to the RTCSEL[1:0] in a RCC register. This register depends on the product and this information can also be found in the product reference manual.

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.

For the RTC3 type, 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



Important: The calendar unit is linked to ck_spre but belongs to the ck_apre (asynchronous prescaler) clock domain. 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 is any clock source (HSE_RTC, LSE or LSI).
- PREDIV_A is any number from 1 to 127.
- PREDIV_S is any number from 0 to 32767.

The table below shows several ways to obtain the calendar clock ck_spre = 1 Hz.

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.

Table 7. Calendar clock ck_spre= 1 Hz with various clock source

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

2.1.5 Calendar firmware examples

The RTC comes with a set of example projects so that the user can quickly become familiar with this peripheral. Refer to the X-CUBE-RTC Expansion Package 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 RTC2:
  - STM32Cube_FW_WB_Vx.y.z\Projects\P-NUCLEO-WB55.Nucleo\Examples_LL\RTC\RTC_Calendar_Init

If one example is not available in the X-CUBE-RTC for a given STM32 MCU, the user can adapt it.

2.2 Binary and mixed modes (RTC3 only)

For the RTC3 type, a binary mode is available. In this mode, the time and date BCD calendar are disabled but the RTC sub-second register (SSR) is extended to 32 bits (16 bits in normal mode) and is used as a binary down-counter.

Thanks to this feature, a 32-bit counter is available in low-power mode (modes compatibilities are the same than for the other RTC features) and a BCD-to-binary conversion is no more needed (when required by an application).

The binary mode implementation is simple:

1. Initialization: Set INIT = 1 in RTC_ICSR and wait for INITF to be set.
2. Define the asynchronous prescaler value in RTC_PRER to clock the binary counter.
3. Set BIN[1:0] = 01 in RTC_ICSR: the RTC_SSR register is initialized to 0xFFFF FFFF.
4. After exiting initialization (INIT bit cleared), the counter start to count down from the RTC_SSR initial value.
5. The counter is clocked by the RTC asynchronous prescaler output. When it reaches 0, it is reloaded with 0xFFFF FFFF.

In binary mode, the RTC_ALRABINR register is used to program the subsecond field of the alarm.

The mixed mode (also offered by RTC3 type) 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 BCDU[2:0] in RTC_ICSR. These bits code how many least significant bits from SSR (subsecond register) need to be at 0 for the calendar to be incremented by 1 second.

A shadow register exists for RTC_SSR. It can be bypassed when BYPSHAD = 1 in RTC_CR. See Section 2.13.3 for more details on BYPSHAD use and associated cautions to take.

2.3 RTC alarms

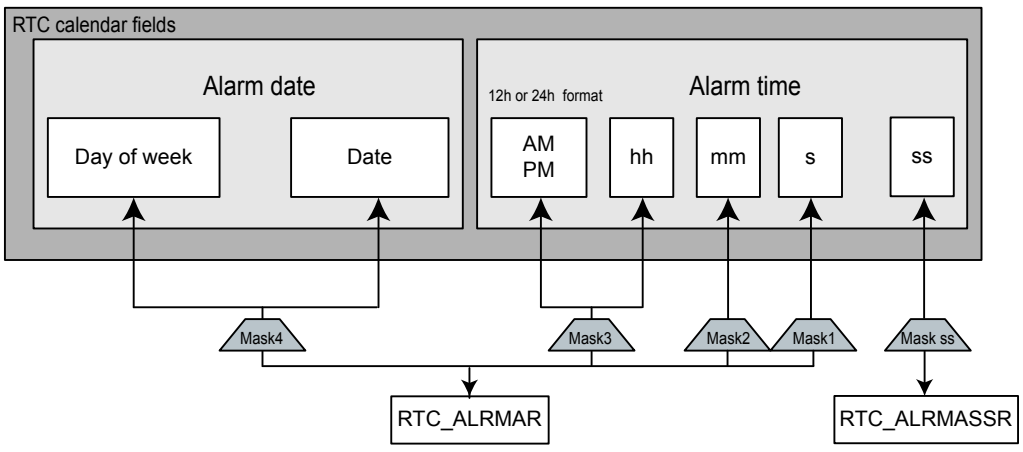
2.3.1 RTC alarm configuration

The RTC embeds two equivalent alarms: alarm A and alarm B. An alarm can be generated at a given time or/and date programmed by the user. The RTC provides a rich combination of alarm settings, and offers many features to make them easy to configure and display.

Each alarm unit provides the following features:

- Fully programmable alarm: subsecond, second, minute, hour and date fields can be independently selected or masked to provide a rich combination of alarms.
- The device can be wake up 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 are available.

Figure 4. Alarm A fields



Notes: . Sames fields are available in RTC_ALRMAR and RTC_ALRMBR registers.
. Sames fields are available in RTC_ALRMASSR and RTC_ARRMBSSR registers.
. Maskx are RTC_ALRMAR (resp. RTC_ALRMBR) bits used to enable/disable the RTC_ALRMAR (resp. RTC_ALRMBR) fields.
. Mask ss bits are available in RTC_ALRMASSR and RTC_ALRMBSSR.

The time configuration bitfields 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_SR).

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

2.3.1.1 *Program the alarm*

The table below describes the steps required to configure alarm A.

Table 8. Steps to confirm alarm A

Step	What	How	Comment
1	Disable write protection on RTC registers.	Write 0xCA and then 0x53 into RTC_WPR.	The RTC registers can be modified.
2	Disable alarm A.	Clear ALRAE(1) in RTC_CR(2).	-
3(3)	Check that RTC_ALRMAR can be accessed.	Poll ALRAWF(4) until it is set in RTC_ISR.	It takes around two RTCCLK clock cycles due to clock synchronization.
4	Configure the alarm.	Configure RTC_ALRMAR(5).	The alarm hour format must be the same(6) as the RTC calendar in RTC_ALRMAR(5).
5	Re-enable alarm A.	Set ALRAE(7) in RTC_CR.	-
6	Enable write protection on the RTC registers.	Write 0xFF into RTC_WPR.	The RTC registers can no longer be modified.

1. *Respectively ALRBE 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 type.*
4. *Resp. ALRBWF for alarm B. ALRAWF and ALRBWF does not exist on RTC3 type.*
5. *Respectively RTC_ALRMBR 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 for alarm B.*

2.3.1.2 *Configure the alarm behavior using the MSKx bits*

The alarm behavior can be configured using the MSKx bits (x = 1, 2, 3, 4) of RTC_ALRMAR for alarm A (RTC_ALRMBR for alarm B). The table below shows all possible alarm settings.

Table 9. Alarm combinations

MSK4	MSK3	MSK2	MSK1	Alarm behavior
0	0	0	0	All fields are used in the alarm comparison. Example: the alarm occurs at 23:15:07, each Monday.
0	0	0	1	Seconds do not matter in the alarm comparison. Example: the alarm occurs every second of 23:15, each Monday.
0	0	1	0	Minutes do not matter in the alarm comparison. Example: the alarm occurs at the 7th second of every minute of 23:XX, each Monday.
0	0	1	1	Minutes and seconds do not matter in the alarm comparison.
0	1	0	0	Hours do not matter in the alarm comparison.
0	1	0	1	Hours and seconds do not matter in the alarm comparison.
0	1	1	0	Hours and minutes do not matter in the alarm comparison.

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

Example

To configure the alarm time to 23:15:07 on Monday assuming WDSEL = 1, configure all MSKx to zero with DU[3:0] = 0001 (Monday), HT[1:0] = 10, HU[3:0] = 0011, MNT[2:0] = 001, MNU[3:0] = 0101, ST[2:0] = 000, SU[3:0] = 0111 (23:15:07), PM = 0 (24h format) in RTC_ALRMAR (resp. RTC_ALRMBR) .

When WDSEL = 0, an alarm occurs on the day number specified in DT and DU bitfields in RTC_ALRMxR, instead of the day of the week. To get this alarm once a month at 23:15:07, on the 14th day, RTC_ALRMxR bitfields must be set as follows: all MSKx = 0, WDSEL = 0, DT[1:0] = 01, DU[3:0] = 0100, HT[1:0] = 10, HU[3:0] = 0011, MNT[2:0] = 001, MNU[3:0] = 0101, ST[2:0] = 000, SU[3:0] = 0111, PM = 0.

Caution: *If the second field is selected (MSK1 reset in RTC_ALRMAR or RTC_ALRMBR), the synchronous prescaler division factor PREDIV_S[14:0] (set in RTC_PRER) must be at least 3 to ensure a correct behavior.*

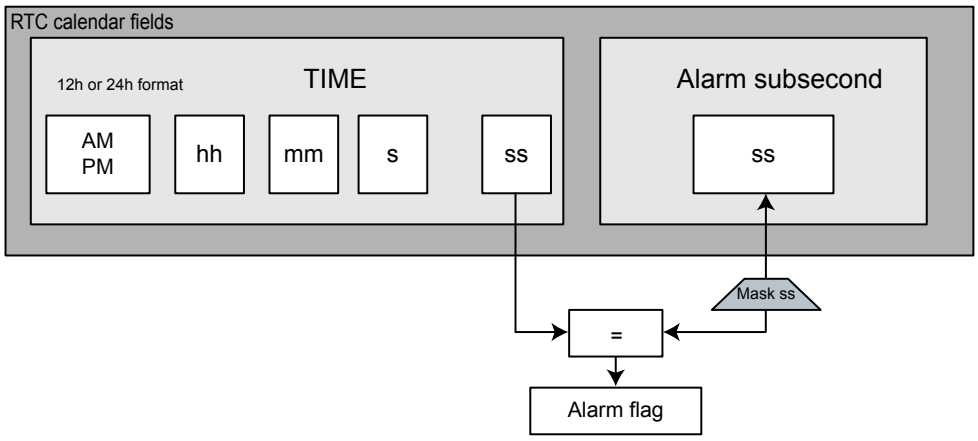
2.3.2 Alarm subsecond configuration

Only RTC3 type with BCD or mixed mode is considered in this section.

The RTC provides similar programmable alarms, subsecond A and B. They generate alarms with a high resolution (for the second division). The value programmed in the alarm subsecond register is compared to the content of the subsecond field in the calendar unit.

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

Figure 5. Alarm subsecond field



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

The subsecond alarm is configured using MASKSS[5:0] in RTC_ALRMASSR (resp. RTC_ALRMBSSR). The table below shows the configuration possibilities for the mask register and provides an example with the following settings:

- LSE selected as RTC clock source (for example LSE = 32768 Hz)
- Asynchronous prescaler = 127
- Synchronous prescaler = 255 (calendar clock = 1 Hz)
- Alarm A subsecond = 255 (SS[14:0] = 255)

Table 10. Alarm subsecond mask combinations (RTC2 type)

MASKSS[5:0]	Alarm A subsecond behavior	Example result
0	No comparison on subsecond 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 RTC_SSR.	Alarm activated every 1/128 s
2	Only the AlarmA_SS[1:0] bits are compared to RTC_SSR.	Alarm activated every 1/64 s
3	Only the AlarmA_SS[2:0] bits are compared to RTC_SSR.	Alarm activated every 1/32 s
4	Only the AlarmA_SS[3:0] bits are compared to RTC_SSR.	Alarm activated every 1/16 s
5	Only the AlarmA_SS[4:0] bits are compared to RTC_SSR.	Alarm activated every 125 ms
6	Only the AlarmA_SS[5:0] bits are compared to RTC_SSR.	Alarm activated every 250 ms
7	Only the AlarmA_SS[6:0] bits are compared to RTC_SSR.	Alarm activated every 500 ms
8	Only the AlarmA_SS[7:0] bits are compared to RTC_SSR.	Alarm activated every 1 s
9	Only the AlarmA_SS[8:0] bits are compared to RTC_SSR.	Alarm activated every 1 s
10	Only the AlarmA_SS[9:0] bits are compared to RTC_SSR.	Alarm activated every 1 s
11	Only the AlarmA_SS[10:0] bits are compared to RTC_SSR.	Alarm activated every 1 s
12	Only the AlarmA_SS[11:0] bits are compared to RTC_SSR.	Alarm activated every 1 s
13	Only the AlarmA_SS[12:0] bits are compared to RTC_SSR.	Alarm activated every 1 s
14	Only the AlarmA_SS[13:0] bits are compared to RTC_SSR.	Alarm activated every 1 s
15	Only the AlarmA_SS[14:0] bits are compared to RTC_SSR.	Alarm activated every 1 s

Note: For RTC3 type, this table can be completed up to MASKSS equal to 31 and the SS[31:16] value is given by RTC_ALRABINR (resp. RTC_ALRBBINR).

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

2.3.3 Alarm firmware examples

The RTC comes with a set of example projects so that the user can quickly become familiar with this peripheral. Refer to the X-CUBE-RTC Expansion Package 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

If one example is not available in the X-CUBE-RTC for a given STM32 MCU, the user can adapt it.

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 a wakeup unit that 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 that can be routed to the RTC_ALARM output for RTC2 and the TAMPALRM output for RTC3 (unique pad for alarm A, alarm B or wakeup events) with configurable polarity
- Full set of prescalers to select the desired waiting period

2.4.1 Program the auto-wakeup unit

The table below describes the steps required to configure the auto-wakeup unit.

Table 11. Steps to configure the auto-wakeup unit

Step	What	How	Comment
1	Disable write protection on RTC registers.	Write 0xCA and then 0x53 into RTC_WPR.	The RTC registers can be modified.
2	Disable the wakeup timer.	Clear WUTE bit in RTC_CR.	-
3	Ensure access to wakeup auto-reload counter and WUCKSEL[2:0] are allowed.	Poll WUTWF until it is set in RTC_ISR (RTC2)/ RTC_ICSR (RTC3)	• For RTC2: It takes around two 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 is cleared. – If WUCKSEL[2] = 1, WUTWF is set. It takes up to 1 ck_apre + 1 RTCCLK cycles after WUTE is cleared.
4	Program the value into the wakeup timer.	Set WUT[15:0] in RTC_WUTR. For RTC3, the user must also program WUTOCLR[15:0][2]in RTC_WUCR	See Section 2.4.2: Maximum and minimum RTC wakeup period.
5	Select the desired clock source.	Program WUCKSEL[2:0] in RTC_CR.	
6	Re-enable the wakeup timer.	Set WUTE in RTC_CR.	The wakeup timer restarts counting down.
7	Enable write protection on RTC registers.	Write 0xFF into the RTC_WPR.	The RTC registers cannot be modified.

1. *ck_wut is the wakeup timer clock input, and ck_spre the clock output by the RTC synchronous prescaler (usually 1 Hz).*
2. *WUTOCLR bits are only available on RTC3 type. They are used 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 type, WUTF must always be cleared by software.*

2.4.2 Maximum and minimum RTC wakeup period

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

- Configuration 1: WUCKSEL[2:0] = 0xx for short wakeup periods (see Section 2.4.2.1)
- Configuration 2: WUCKSEL[2:0] = 10x for medium wakeup periods (see Section 2.4.2.2)
- Configuration 3: WUCKSEL[2:0] = 11x for long wakeup periods (see Section 2.4.2.3)

2.4.2.1 *Periodic timebase/wakeup configuration for clock configuration 1*

The figure below shows the prescaler connection to the timebase/wakeup unit and the table below gives the timebase/wakeup clock resolutions corresponding to configuration 1.

The prescaler depends on the wakeup clock selection as follows:

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

Figure 6. Prescalers connected to the timebase/wakeup unit for configuration 1

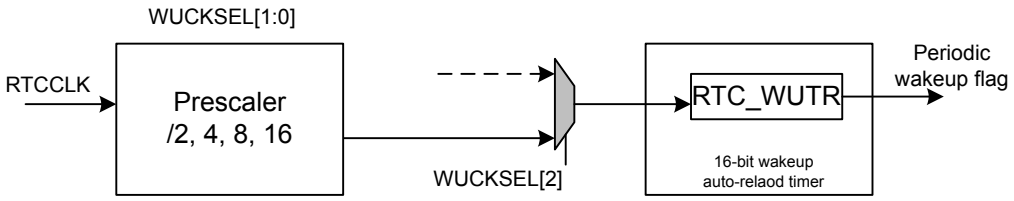


Table 12. Timebase/wakeup unit period resolution with clock configuration 1

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

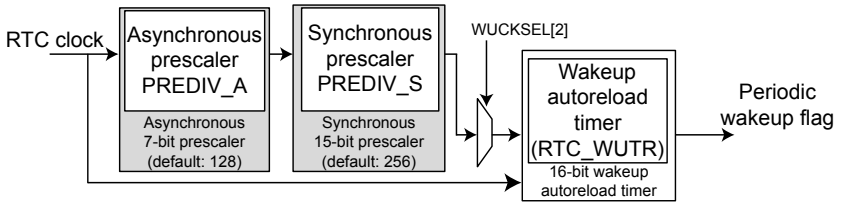
When RTCCLK= 32768 Hz, the minimum timebase/wakeup resolution is 61.035 µs, and the maximum resolution is 488.28 µs. As a result:

- 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] = 011 ($f_{RTCCLK}/2$) because this configuration is prohibited. Refer to the product reference manuals for more details.
- The maximum timebase/wakeup period is (0xFFFF+ 1) x 488.28 µs = 32 s.

2.4.2.2 *Periodic timebase/wakeup configuration for clock configuration 2*

The figure below 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).

2.4.2.3 *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 down counts 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).

Note: In binary or mixed modes, ck_spre is replaced by the clock used to update the calendar (as defined by BCDU bits in RTC_ICSR). See Section 2.2 for more details on BCDU).

2.4.2.4 *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 the table below.

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 comes with a set of example projects so that the user can quickly become familiar with this peripheral. Refer to the X-CUBE-RTC Expansion Package 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_ProgrammingTheWakeUpTime

If one example is not available in the X-CUBE-RTC for a given STM32 MCU, the user can adapt it.

2.5 Smooth digital calibration

2.5.1 RTC calibration basics

The “quartz-accurate” term describes the accuracy of many time-keeping functions. The quartz oscillators provide a far superior accuracy versus other conventional oscillator designs, but they are not perfect, as 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, following the general formula:

$$acc = K \times (T - T_0)^2$$

where To = 25 °C ± 5 °C and K = –0.032 ppm/°C^2.

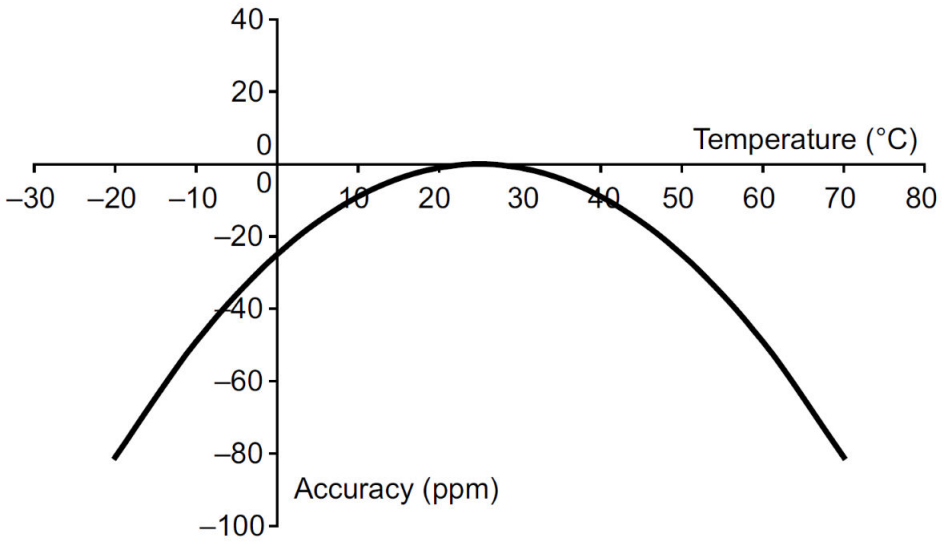
Note: The K variable is crystal-dependent. The value indicated here is for the crystal mounted on the STM32L476RG-Nucleo board. Refer to the crystal manufacturer for more details on this parameter.

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 an oscillation frequency deviation of an oscillator when the generated signal is used to clock a time-keeping logic:

- Analog approach
  The oscillator is built with embedded capacitor banks on both input and output of the oscillator. 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.
  Note 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 load capacitance of the crystal resonator. The oscillator 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, such as relatively important value dispersion from one chip to another.
- Digital approach
  The oscillation frequency deviation is not compensated at oscillator level. The compensation is made at time-keeping logic/function level. The time-keeping logic either adds or removes few clock cycles to/from the deviating clock signal that feeds its counter. The main advantage of this approach is the deterministic compensation effect from one device to another. This approach is adopted to compensate the LSE oscillation frequency deviation when LSE is used to clock the RTC. The RTC 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 RTC calibration 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 type, ck_cal is always RTCCLK.

For RTC3 type, if LPCAL = 0 in RTC_CALR, ck_cal = RTCCLK. If LPCAL = 1, ck_cal = ck_apre (clock output by the RTC asynchronous prescaler). Moreover, when setting LPCAL to 1 and choosing ck_apre, the calibration consumption decreases, 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 smooth digital calibration is designed to compensate the inaccuracy of crystal oscillators (due to temperature or crystal aging).

Figure 9. **Smooth calibration block for RTC2 type**

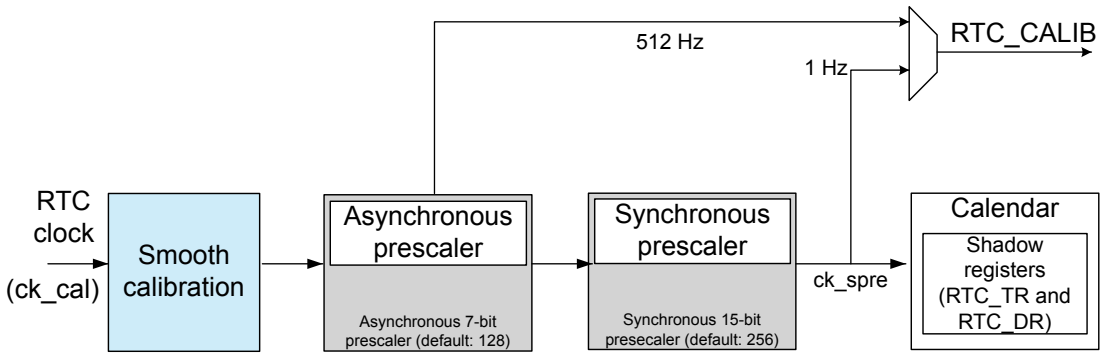
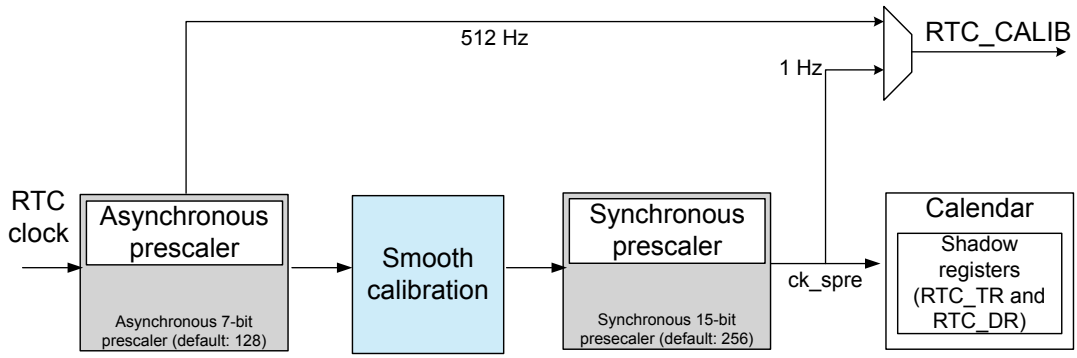


Figure 10. **Smooth calibration block for RTC3 type (LPCAL = 1)**



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 and in Section 6.

The calibration result can be checked by using the calibration output 512 Hz or 1 Hz for the RTC_CALIB signal. Refer to Table 4 and Table 5.

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[8:0] in 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 CALW8 or CALW16 in RTC_CALR.

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] = 1, CALP = 0 and using 32 s calibration window, results in exactly one pulse being masked for 32 s.

Example 2

Setting CALM[2] = 1, CALP = 0 and using 32 s calibration window, results in exactly four pulses being masked for 32 s.

Note: Both CALM[8:0] and CALP can be used. In this case, an offset ranging from -511 to +512 pulses can be added for 32 s (calibration window).

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

The formula to calculate the effective calibrated frequency (F_{CAL}), given the input frequency F_{RTCCLK}, 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, changed when the temperature changes or if other factors are detected.

Check the smooth calibration

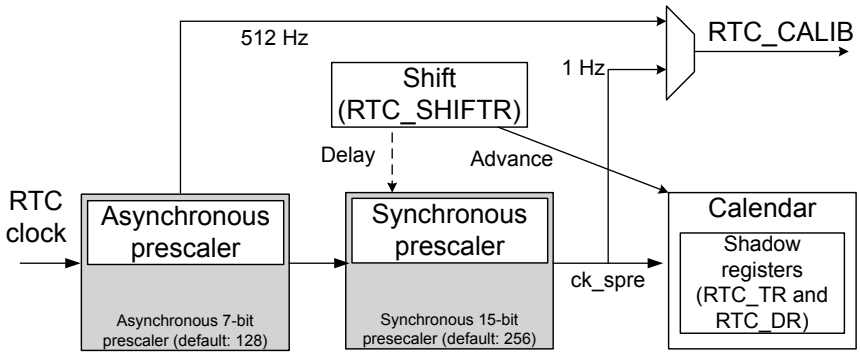
The smooth calibration effect on the calendar clock (RTC clock) can be checked by calibration using one of the following:

- RTC_CALIB output (1 Hz)
- Subsecond alarms
- Wake-up timer

2.6 Synchronize the RTC

The RTC calendar can be synchronized to a more precise clock, “remote clock”, using the RTC shift feature. After reading the RTC subsecond 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



The synchronization shift function cannot be checked using the RTC_CALIB output since the shift operation has no impact on the RTC clock (except adding or subtracting a few fractions from the calendar counter).

Correct 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[14:0] in RTC_SHIFTR, that is added to the synchronous prescaler 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 ADD1S in RTC_SHIFTR is used in conjunction with SUBFS, effectively advancing the clock by:

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

Caution: *For RTC3 type, ADD1S has no effect in binary mode. In mixed mode, the SUBFS[14:BCDU+8] must be written with 0. Before initiating a shift operation in BCD mode, the user must check that SS[15] = 0 in order to ensure that no overflow occurs. In mixed mode, the user must check that the bit SS[BCDU+8] = 0.*

An example for this feature is provided in the X-CUBE-RTC (see Section 7 for more details).

2.7 RTC reference clock detection

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

The reference clock (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 REFCKON in RTC_CR. 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 three ck_apre periods.

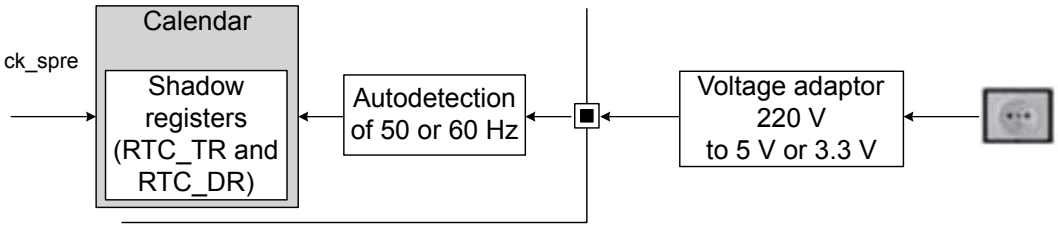
If the reference clock halts, the calendar is updated continuously based only on the LSE clock. The RTC waits for the reference clock using a detection window centered on the ck_spre (synchronous prescaler output clock) edge. The detection window is seven ck_apre periods.

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 one ck_ref period (20 ms for a 50 Hz reference clock) because it is possible to have two ck_ref edges in the detection window. The update window is then 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, the total uncertainty per month is 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 V_{BAT} 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 (see Section 8 for more details).

2.8 RTC prescaler adjustment with LSI measurements

When the LSI is selected as RTC clock source, a timer can be used 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 LSI clock signal is connected to the input capture of one of the STM32 timers (see the timers connected to LSI in the product reference manual). For each LSI period, the timer must count the number of system core clock periods elapsed between two LSI rising edges associated. The software exploits 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.

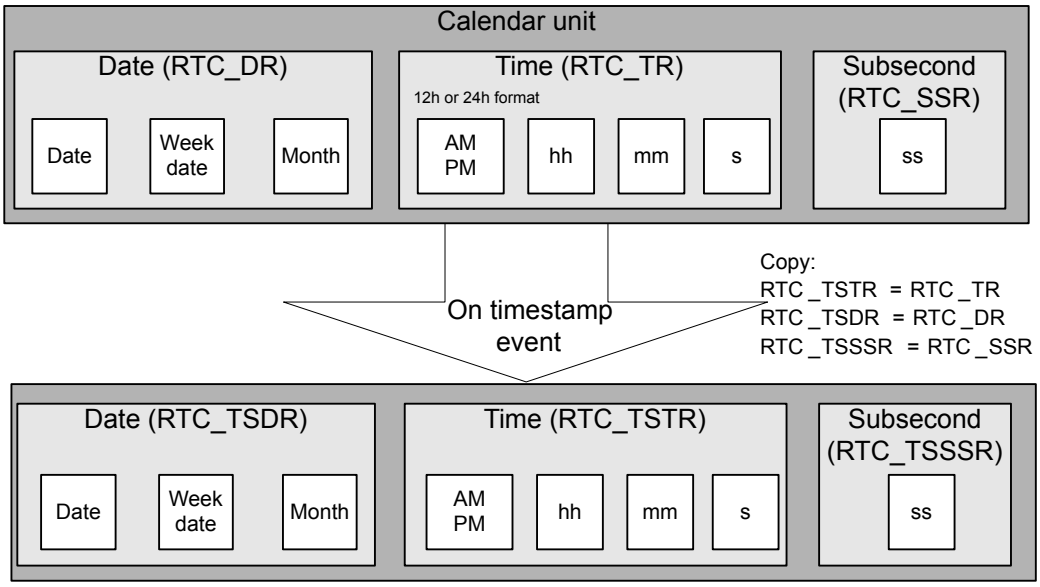
A firmware example is available in the STM32CubeL4 MCU Package for NUCLEO-L412RB-P (RTC3 type): STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_LSI

If one example is not available in the STM32Cube for a given STM32 MCU, the user can adapt it.

2.9 Timestamp function

The timestamp feature is used to automatically save the current calendar when some specific events occur.

Figure 13. Timestamp event procedure



When this 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) is set in RTC_ISR (RTC2)/RTC_SR (RTC3).

The following events can generate a timestamp:

- Edge detection on the RTC_TS I/O
- Tamper event detection (from all RTC_TAMP I/Os)
- Switch to V_{BAT} when the main supply if powered off (available for example on STM32L4 series, see the product reference manual and datasheet to verify availability for other products)

Table 14. Timestamp features

What	How	Comments
Enable timestamp.	Set to 1 TSE, ITSE or TAMPTS in RTC_CR.	TAMPTS is only available on RTC3 to allow tamper events to trigger timestamp.
Detect a timestamp event by interrupt.	Set TSIE in RTC_CR.	An interrupt is generated when a timestamp event occurs.
Detect a timestamp event by polling.	Poll on the timestamp flags (TSF or ITSF(1)) in RTC_ISR (RC2)/RTC_SR (RTC3).	To clear the flag, write zero on TSF or ITSF(2).
Detect a timestamp overflow event.(3)	Check on TSOVF(4)) in RTC_ISR (RC2)/ RTC_SR (RTC3).	• To clear the flag, write zero in TSOVF. • RTC_TSTR, RTC_TSDR, and RTC_TSSSR maintain the results of the previous event. • If a timestamp event occurs immediately after TSF is supposed to be cleared, then both TSF and TSOVF are set.

1. *TSF is set two 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 unless the software already read this bit at 1.*
3. *The timestamp overflow event is not connected to an interrupt.*
4. *There is no delay in TSOVF setting: if two timestamp events are close to each other, TSOVF can be seen as 1 while TSF is still 0. It is then recommended to poll TSOVF only after TSF has been set.*

2.9.1 Timestamp firmware examples

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

If an example is not available in the STM32Cube for a given STM32 MCU, the user can adapt it.

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. Each tamper input has an individual flag (TAMPxF in RTC_ISR (RTC2)/TAMP_SR (RTC3)).

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

The configuration of the tamper filter, TAMPFLT[1:0] in TAMP_FLTCR, defines whether the tamper detection is activated on edge (set TAMPFLT[1:0] = 00), or on level (TAMPFLT[1:0] ≠ 00).

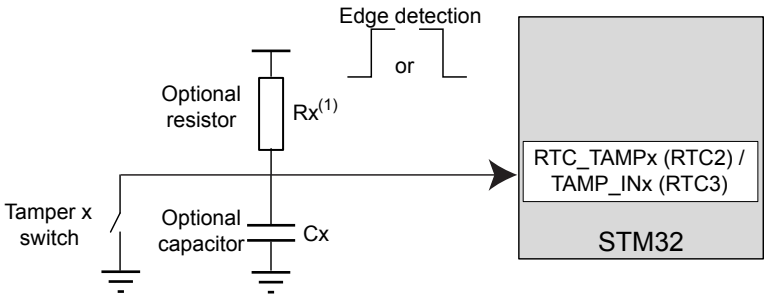
The number of tamper inputs depends on product packages. Each input has a TAMPxF individual flag in RTC_ISR (RTC2)/TAMP_SR (RTC3).

The DBP bit (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 TAMPFLT[1:0] = 00, the tamper input detection triggers when a rising edge or a falling edge (depending on TAMPxTRG) is observed on the input pin RTC_TAMPx (RTC2)/TAMP_INx (RTC3).

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)

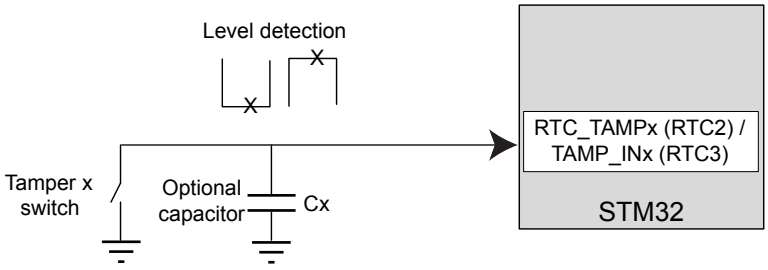
What	How	Comment
Enable tamper.	Set TAMPxE = 1 in RTC_TAMPCR (RTC2)/ TAMP_CR1 (RTC3).	-
Select tamper active edge detection.	Select with TAMPxTRG in RTC_TAMPCR (RTC2)/ TAMP_CR2 (RTC3).	The default edge is rising edge.
Detect a tamper event by interrupt.	Set TAMPIE or TAMPxIE in RTC_TAMPCR (RTC2)/ TAMP_IER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

2.10.2 Level detection on tamper input

Setting TAMPFLT[1:0] to a value other than zero means that the tamper input triggers when a selected level (high or low) is observed on the corresponding input pin RTC_TAMPx (RTC2)/TAMP_INx (RTC3).

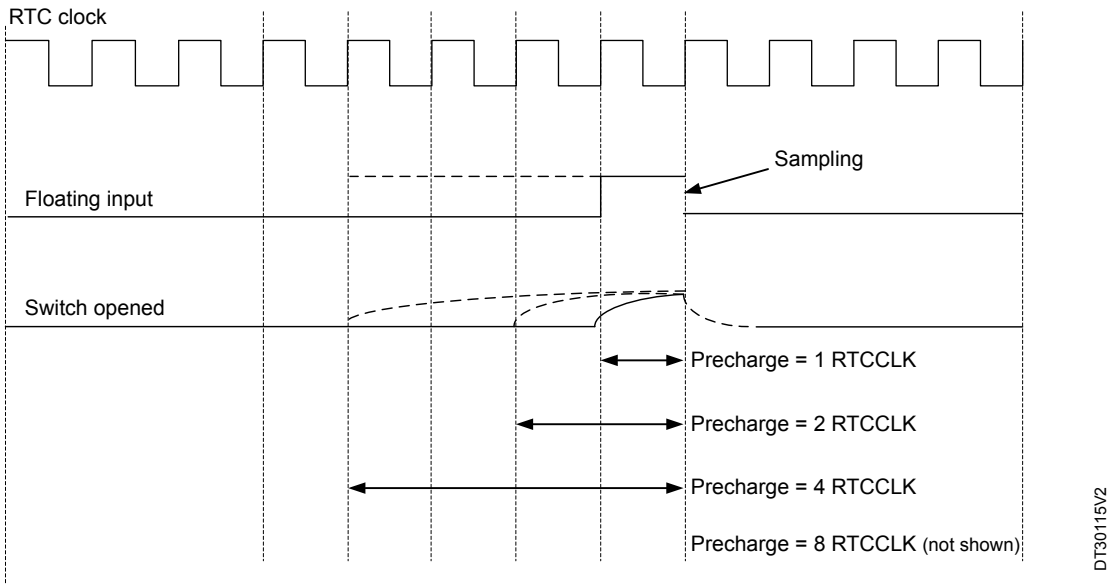
A tamper detection event is generated when either two, four or eight (depending on the TAMPFLT value) consecutive samples are observed at the selected level.

Figure 15. Tamper with level detection



Using the level detection (TAMPFLT[1:0] ≠ 0), 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 one, two, four or eight RTCCLK cycles (see TAMPPRCH[1:0]).

Figure 16. **Tamper sampling with precharge pulse**



Note: When the internal pull-up is not applied, the I/O 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 TAMPFREQ[2:0] in 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	How	Comment
Enable tamper.	Set to 1 TAMPxE in RTC_TAMPCR (RTC2)/TAMP_CR1 (RTC3).	-
Configure tamper filter count.	Configure TAMPFLT bits in RTC_TAMPCR (RTC2)/ TAMP_FLTCR (RTC3).	-
Configure tamper sampling frequency .	Configure TAMPFREQ bits in RTC_TAMPCR (RTC2)/ TAMP_FLTCR (RTC3).	Default value is 1Hz.
Configure tamper internal pull-up and precharge duration.	Configure TAMPPUDIS and TAMPPPRCH in RTC_TAMPCR (RTC2)/ TAMP_FLTCR (RTC3).	-
Select tamper active edge/level detection.	Select with RTC_TAMPCR (RTC2)/TAMP_CR2 (RTC3).	Edge or level depends on tamper filter configuration.
Detect a tamper event by interrupt.	Set TAMPIE or TAMPxIE in RTC_TAMPCR (RTC2)/ TAMP_IER (RTC3).	An interrupt is generated when tamper detection event occurs.
Detect a tamper event by polling.	Poll TAMPxF in RTC_ISR (RTC2)/TAMP_SR (RTC3).	To clear the flag, write zero in TAMPxF.

2.10.3 Action on tamper detection event

By setting TAMPTS = 1, any tamper event (with edge or level detection) causes a timestamp to occur. The timestamp and timestamp overflow flags are then 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.

Other actions than a timestamp can be triggered by a tamper detection depending on the product used. Refer to the product documentation for a complete list of actions.

For example, a tamper detection can:

- Erase the backup registers, SRAMs content or specific peripheral registers
- Generate an interrupt that can wake up the device from low-power modes
- Generate an hardware trigger for a low-power timer or a RTC timestamp event (see TAMPTS bit mentioned before)

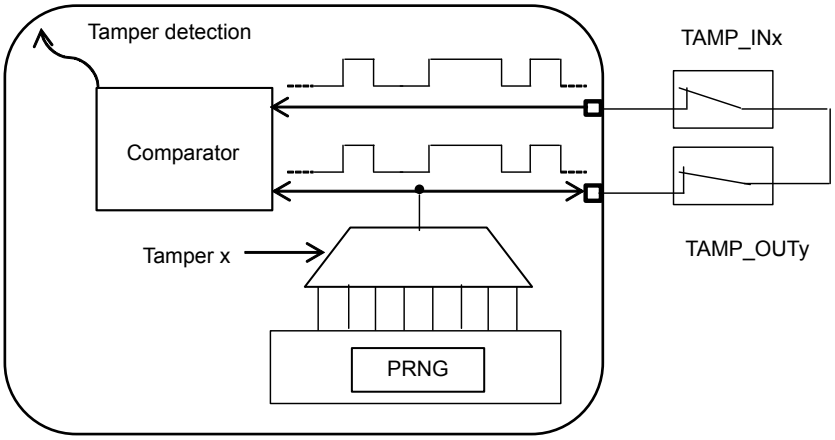
On some product, the protection of specific device assets can be enabled/disabled thanks to the ERCFGy bit in TAMP_ERCFGR. If this bit is set, in case of a tamper detection or when BKBLOCK = 1 in TAMP_CR2, the read/write accesses to the concerned device assets are blocked.

2.10.4 Active tamper detection (RTC3 only)

The tamper detection can be made robust against tamper pin being externally opened or shorted. That is done thanks to the 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 occurred: TAMPxF is set in TAMP_SR for the TAMP_INx pin detecting the error. Depending on the tamper pins available, several organizations can be configured. For example, for eight tamper pins, the user can have four outputs and four inputs with four different messages, or seven inputs with one output and one message exchanged.

Figure 17. Tamper detection



2.10.4.1 Active pin IN/OUT association

The user configures TAMP_INx as active input pins by setting TAMPxAM = 1 in TAMP_ATCR1.

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

Example: considering there are at least two active tamper pins on the product, 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 = 0 and ATOSEL2 = 0.

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

2.10.4.2 Filtering

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

2.10.4.3 *Message randomization*

Concerning the randomization of the message sent by the active output pins, a pseudo-random number generator (PRNG value in TAMP_ATOR) 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 a mathematical 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 does not 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.

2.10.4.4 *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 ITAMPxE in TAMP_CR1. The interrupt is enabled by setting ITAMPxIE in TAMP_IER. Then, if the event occurs, ITAMPxF is set in TAMP_SR, and ITAMPxMF is set in TAMP_MISR if the interrupt has been enabled.

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

The STM32 MCUs implement some of the internal tampers listed below (refer to product reference manual):

- ITAMP1: supply voltage monitoring (or backup domain voltage monitor)
- ITAMP2: temperature monitoring
- ITAMP3: LSE monitoring (CSS)
- ITAMP4: HSE monitoring (CSS)
- ITAMP5: RTC calendar overflow
- ITAMP6: JTAG/SWD access when RDP > 0 (or NVSTATE != open in STM32H5)
- ITAMP7: ADC analog watchdog monitoring 1
- ITAMP8: monotonic counter 1 overflow
- ITAMP9: cryptographic peripheral fault (SAES, AES, PKA or RNG)
- ITAMP10: reserved
- ITAMP11: IDWG reset when tamper flag is set (potential tamper timeout)
- ITAMP12: ADC analog watchdog monitoring 2
- ITAMP13: ADC analog watchdog monitoring 3
- ITAMP14: reserved
- ITAMP15: system fault
- ITAMP16: reserved

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

As for every tamper detection features, the internal tamper events allow attacks to be detected on the MCU. 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 and detailed in Section 9.

The RTC3 tamper detection unit embeds also a monotonic counter implemented in TAMP_COUNTR register (called TAMP_COUNT1R on some products) and associated to ITAMP8. See Table 4 and Table 5 for the availability in the product. This register can be read and is incremented each time the user writes it whatever the written value. TAMP_COUNTR cannot roll-over and is frozen when reaching its maximum. This register 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. In that case, ITAMP8 can be used to trigger a tamper event.

ITAMP7, ITAMP12 and ITAMP13 are associated to analog watchdogs and are used to consider that an ADC channel going out of a configured voltage range is a tamper event (see the documentation of the products featuring this internal tamper for more details).

2.10.6 Potential detection management (RTC3 only)

If TAMPxNOER = 0 in TAMP_CR2 or ITAMPxNOER = 0 in TAMP_CR3, the backup registers and other device secrets are automatically erased by hardware on an external (passive or active) or internal tamper event.

In order to perform software filtering of the tamper event, the user must configure these bits to 1. In this case, the backup registers and device secrets are not erased but their read/write access is blocked as long as at least one of the tamper flag is set (TAMPxF and ITAMPxF). These tamper events are considered as potential tampers, and the user can investigate if the tamper event corresponds to a true or to a false tamper, and then take the necessary actions.

If the user confirms it is a true tamper detection, backup registers and device secrets can be erased by setting BKERASE = 1 in TAMP_CR2. If the tamper is not confirmed and is considered a false detection, the user must clear the tamper status flag. When all tamper flags are cleared, the access to backup registers and device secrets is possible again.

By associating the NOERASE feature with the IWDG (independent watchdog), a timeout can be implemented to automatically launch hardware erasing after a potential tamper detection. On some products, the ITAMP11 is associated to the IWDG reset. In this case, if a tamper event in NOERASE configuration is detected, and if a IWDG reset occurs before the software has cleared the tamper flag, the backup registers and device secrets are automatically erased by hardware. The probability that a software issue, preventing from refreshing the watchdog, occurs at the same time than a tamper detection is very low so the hardware considers it is a real attack.

2.10.7 Tamper detection firmware examples

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

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

- For most microcontrollers equipped with either an RTC2 or an RTC3:
  STM32Cube_FW_product_Vx.y.z\Projects\board\Examples\RTC\RTC_Tamper
- For many products equipped with either an RTC2 or an RTC3:
  STM32Cube_FW_product_Vx.y.z\Projects\board\Examples_LL\RTC\RTC_Tamper_Init
- For the STM32L552E evaluation board, the STM32H7B3I evaluation board and the STM32U575ZI Nucleo board equipped with an RTC3:
  STM32Cube_FW_product_Vx.y.z\Projects\board\Examples\RTC\RTC_ActiveTamper

If an example is not available in the STM32Cube for a given STM32 MCU, the user can adapt it from a similar example.

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 V_{BAT} when V_{DD} is switched off (some products do not implement this feature, like 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 type on STM32L0, STM32L4, and STM32F7 series, the 32-bit backup registers (RTC_BKPxR) are not reset if TAMPxNOERASE = 1 or if TAMPxMF = 1 in RTC_TAMPCR.

For RTC3 type, the 32-bit backup registers (TAMP_BKPxR) are not reset if TAMPxNOER = 1 or if TAMPxMSK = 1 in TAMP_CR3.

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

Note: The V_{BAT} availability, the LPTIM availability and the number of backup registers depend on the product. Refer to Table 4 and Table 5.

The RTC3 backup registers can be protected against nonsecure or unprivileged accesses (see Section 2.14: Reduce power consumption).

BKPRWDPROT[7:0] and BKPWDPROT[7:0] in TAMP_SMCR (except for the STM32U3 and STM32U5 series: BKPRWSEC[7:0] and BKPWSEC[7:0] in TAMP_SECCFGR) are used to program the backup registers offset, delimiting three protection zones for the backup registers:

- Zone 1: read and write secure
- Zone 2: read nonsecure and write secure
- Zone 3: read and write nonsecure

Secure AES boot hardware key (RTC3 only)

Note: This section is valid only for products embedding a SAES.

The backup registers can be used to store the BHK (boot hardware key) of the SAES (secure AES). If the device supports Arm® TrustZone® security, the eight first backup registers (TAMP_BKP0R to TAMP_BKP7R) must be configured as belonging to the zone 1 (r/w secure) by setting BKPRWSEC ≥ 8 in TAMP_SECCFGR.

Once the registers are written with the key, BHKLOCK must be set to 1 in TAMP_SECCFGR to definitively block the read/write accesses to these registers (any reading returns 0 and any writing is ignored). BHKLOCK can only be cleared by hardware after a tamper event or when readout protection is disabled. In these two case, the backup registers are also erased. After BHKLOCK is set, the SAES coprocessor can use the BHK thanks to a private hardware bus (see the SAES section in the product reference manual and more precisely KEYSEL bits in the SAES control register).

2.12 Alternate function RTC outputs

The RTC has two outputs:

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

For RTC3 type, these outputs can be associated to two pins against one pin for RTC2 type. Refer to OUT2EN in RTC_CR of RTC3 type 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 to measure the RTC clock deviation when compared to a more precise clock.

Setting 512 Hz as 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 = 1.
4. Select 512 Hz as the calibration output by setting COSEL = 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 = 1.
5. Select 1 Hz as the calibration output by setting COSEL = 1.

Note: Refer to figures in Section 2.5.2: RTC calibration methodology.

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 can 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] = 111111 (div64)	Maximum PREDIV_A[5:0] = 100000(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] = 0, no signal is output on RTC_CALIB.*

2.12.2 RTC_ALARM (RTC2)/TAMPALRM (RTC3) output

The alarm, wakeup and tamper flags can be routed to the RTC outputs (one output for RTC2 type, three for RTC3 type) thanks to OSEL[1:0] in RTC_CR.

For RTC2 type

- OSEL[1:0] = 01 roots alarm A to the RTC_ALARM output.
- OSEL[1:0] = 10 roots alarm B to the RTC_ALARM output.
- OSEL[1:0] = 11 roots wakeup flag to the RTC_ALARM output.

Once OSEL[1:0] is fixed, the output reflects the selected flag stored in RTC_ISR.

For RTC3 type

The OSEL[1:0] selection follows the same principle as for RTC2 but, when TAMPOE = 1 in RTC_CR, tamper flags are ORed with the flag already selected (alarm A, alarm B or wakeup) before the OR result to be rooted to the TAMPALRM output.

If TAMPOE = 1 and OSEL[1:0] = 00, TAMPALRM output reflects only the tamper flags.

For both RTC2 and RTC3 types, the output pin polarity is selected with POL 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 for push-pull, 1 for open-drain).

For RTC3, an internal pull-up can be added to the output pin by setting TAMPALRM_PU = 1 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 locked after a Backup domain reset. These registers 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 RTC_WPR, **only** with the following sequence:

1. Write 0xCA in RTC_WPR.
2. Write 0x53 in RTC_WPR.

Any other write access sequence to RTC_WPR 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 (counters stopped)
- Free-running mode (counters 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 and they start counting from the new value when the RTC enters Free-running mode.

The INIT bit in RTC_ISR (RTC2)/RTC_ICSR (RTC3) enables the user to switch from one mode to another. The INITF bit in the same register 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 INIT = 1 and waiting until INITF flag is set.

To return to the Free-running mode and restart counting, the RTC must exit the Initialization mode (done by resetting INIT = 0).

Only a power-on reset can reset the calendar on MCUs without the VBAT pin. A system reset does not affect the calendar but resets the shadow registers (APB registers clocked by the APB bus clock) that are read by the application. These registers are updated again when RSF = 1 in RTC_ISR (RTC2)/RTC_ICSR (RTC3).

After a system reset, the application can use the INITS status flag in RTC_ISR (RTC2)/RTC_ICSR (RTC3) to check 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 reset by setting BDRST = 1 in the RCC, even when V_{BAT} is present.

The LSE clock initialization is not always needed after a system reset. It is recommended to check LSERDY in RCC_BDCR and to initialize LSE only if this flag is not already set.

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). There is a shadow register associated to RTC_TR, RTC_DR and RTC_SSR. RSF is set in RTC_ISR (RTC2)/RTC_ICSR (RTC3) each time the calendar time and date shadow registers are updated with the real calendar value. The copy is performed every RTCCLK cycle, synchronized with the APB bus clock (APB bus by which the RTC 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 RSF, and then wait until RSF 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 BYPSHAD = 1 in RTC_CR, 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 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 setting BYPSHAD = 0, the shadow registers may be incorrect until the next synchronization. In this case, the software must clear RSF = 0, wait for the synchronization (RSF must be set), and finally read the shadow registers.

2.14 Reduce power consumption

The RTC is designed to minimize the power consumption and can be configured to optimize further its consumption.

2.14.1 Use the right power reduction mode

The RTC can be active in the following low-power modes (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 product reference manual and datasheet.

Thanks to the RTC wakeup unit, the user can wake up the STM32 MCU from this mode. Depending on the application needs in term of low-power phases and current consumption target, the adequate low-power mode and wakeup frequency must be selected.

2.14.2 Use internal pull-up resistor on tamper pin

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). Using this RTC internal pull-up resistor instead of standard I/O pull-up/pull-down or external pull-up/pull-down ensures a 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[2:0], that determines the frequency of the tamper sampling from 128 Hz to 1 Hz when RTCCLK equals 32768 Hz.

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

2.14.3 Set 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 type, the RTC dynamic consumption is optimized for LPCAL = 1 and PREDIV_A + 1 being a power of two (considering the RTC as the only peripheral using LSE).

2.14.4 External optimization factors

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

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 = 00 in RCC_BDCR.

2.14.5 Low-power management firmware examples

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

For example, for the small STM32L4 product line, the user can find the following projects concerning low-power management:

- For NUCLEO-L412RB-P:
  STM32Cube_FW_L4_Vx.y.z\Projects\NUCLEO-L412RB-P\Examples\RTC\RTC_LowPower_STANDBY
- For NUCLEO-L412KB:
  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

Analogous examples are available for STM32F0, STM32F2, STM32F3, STM32F4, STM32F7, STM32G4, STM32H7, STM32L0, STM32L1, STM32U3, STM32U5, STM32WB and STM32WL. If one example is not available in the STM32Cube for a given STM32 MCU, the user can adapt it.

2.15 RTC3 secure and privileged protection modes

Some STM32 MCUs can run in secure or nonsecure mode. Each mode having its own environment, allowing a trustful zone, not accessible in nonsecure mode, to be maintained in secure mode (TrustZone® feature).

A second level of restriction and security is available with part of each environment associated to a privileged or unprivileged mode.

On some products (see Table 4 and Table 5), the registers programming allow the user to restrict the read/write operation on RTC/TAMP registers to secure or privileged modes. Some configuration bits establish the restriction only for some RTC registers. These registers/bits to program are product dependent, then not detailed here (refer to the product reference manual).

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

3 STM32L4 API and tamper detection application example

3.1 STM32CubeL4 firmware libraries for tamper detection

The RTC 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 in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Eval\Examples\RTC

3.2 X-CUBE-RTC for tamper detection

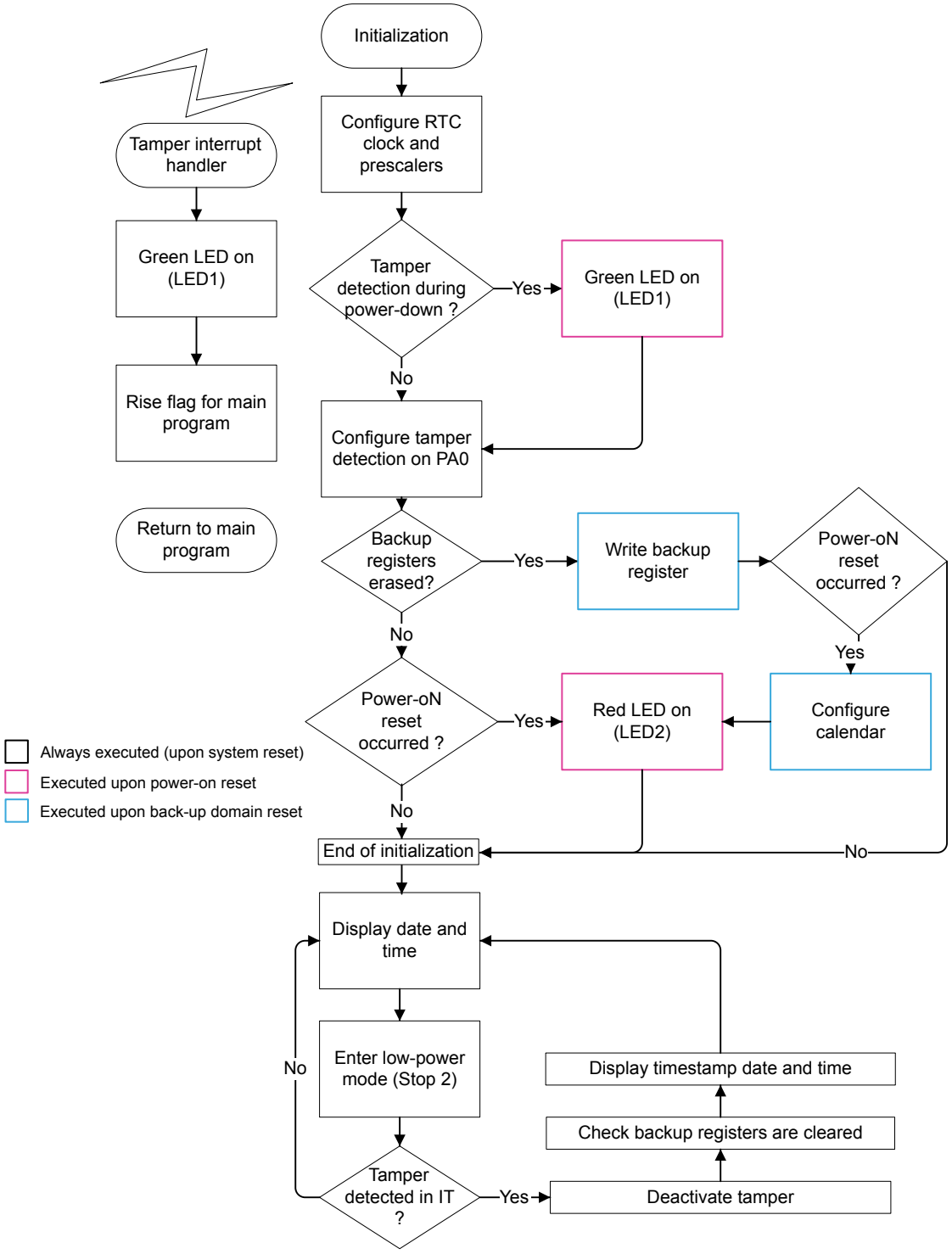
X-CUBE-RTC shows an example where a real-time clock must be maintained alive while using as low power as possible (`STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\STM32L476G_EVAL\RTC_TamperVBATT`).

The firmware implements:

- Hints 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
- Tampering detection capability, both when the main power supply (V_{DD}) 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 the figure below.

Figure 18. Application example flowchart

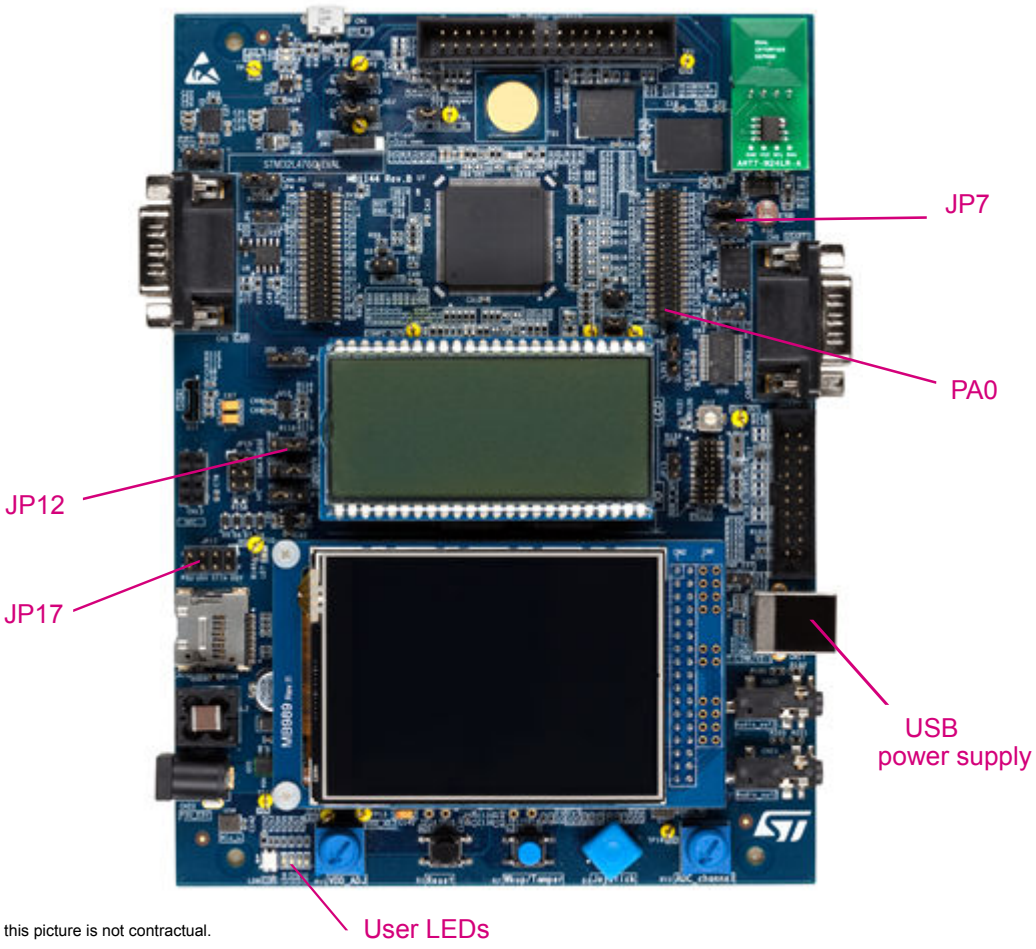


3.3 Tamper detection application example (STM32L4)

3.3.1 Hardware setup

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

Figure 19. STM32L476G-EVAL board



Note: this picture is not contractual.

Supply the board using the USB cable and 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 favorite IDE (integrated development environment). For IAR Embedded Workbench®, 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 executes it.

In IAR Embedded Workbench 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 MDK-ARM development tools. To open the *Watch1* window, do View\Watch Windows\Watch1.

3.3.3 LED meaning

- LD1 (green): tamper event detected
  TFT displays: the tamper date and tamper time are different from their initialization state.
- LD2 (orange): power-on reset occurred
  TFT displays: the tamper date and 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 tamper time are at their initialization state.

3.3.4 Tamper detection during normal operation

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

Open PA0 (input is now floating), the LD1 lights showing that a tamper event is detected. On TFT, the tampering date and time are updated with the timestamp of the tamper event.

Note: A long delay (up to 2 seconds) may be observed before the tamper event is detected. This is due to the trade-off in the tamper detection frequency (TAMPFREQ) chosen to ensure 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 Tamper 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, with the following steps:

- Remove the JP17 jumper.
- Open PA0.
- Tie PA0 to GND (simulating that the end-user takes care to close the box containing the electronics before supplying it again).
- Set JP17 on STlk.
- The LD1 lights showing that a tamper event has been detected during the power-down. 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. The application is no longer able to detect the tampering event while the main power supply is off.

4 STM32L4 API and smooth digital calibration application example

4.1 STM32CubeL4 firmware libraries for smooth calibration

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 in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Nucleo\Examples\TIM

The RTC 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 in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Nucleo\Examples\RTC

4.2 X-CUBE-RTC for smooth calibration

X-CUBE-RTC shows an implementation of the smooth digital calibration feature of the RTC (using also the GPTIMER HAL API).

In this example, the 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:

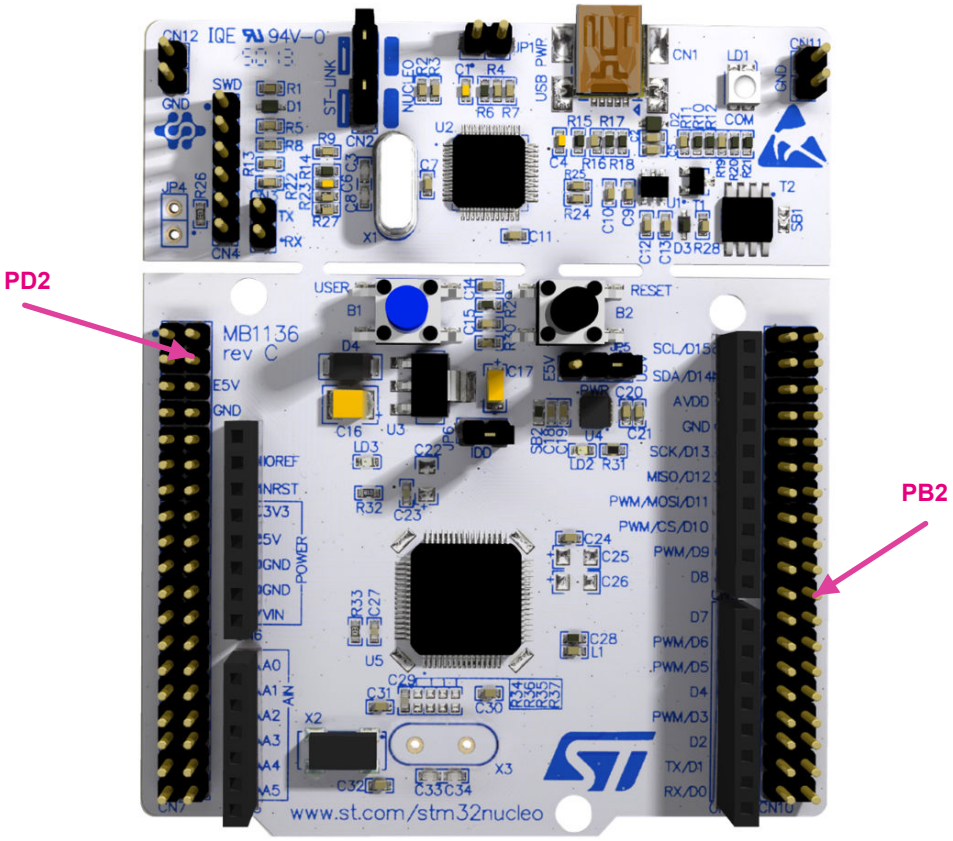
- A GPTIMER (general-purpose timer) in Master mode using channel 1 in output compare mode (and using an external trigger clock)
- A GPTIMER in Master mode using channel 1 in input capture mode and channel 2 in output compare mode
  This timer is also configured to use the LSE clock thanks to the TIM option register.
- The RTC with the CALR register programming
- GPIOs to connect the input 1 Hz reference clock and the “corrected” RTC_OUT_CALIB clock

4.3 Smooth calibration application example (STM32L4)

4.3.1 Hardware setup

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

Figure 20. NUCLEO-L476RG board



Note: this picture is not contractual.

Supply the board using the USB cable and perform the following actions:

- 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 (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 favorite IDE. For IAR Embedded Workbench, open the `RTC_SmothCalib.eww` file. For 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.

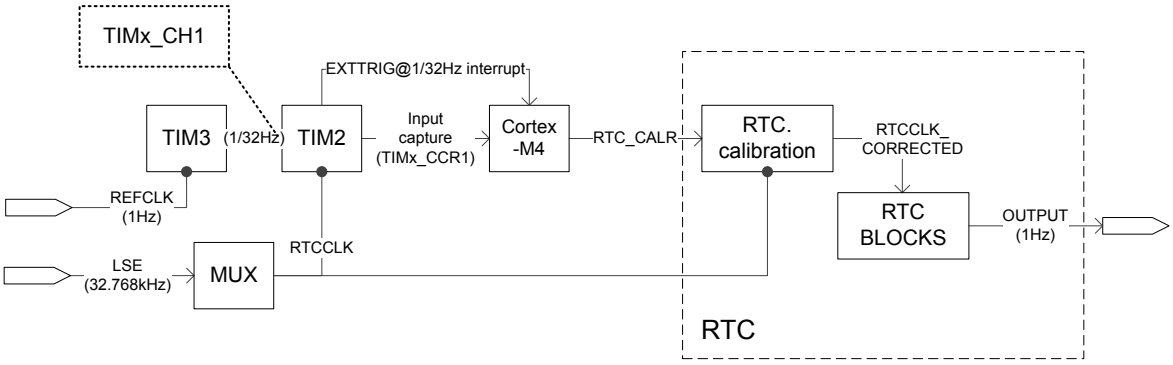
The user can then modify the PD2 clock by few ppm, wait 64 seconds, and check that the PB2 clock has been changed accordingly.

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

4.3.3 Smooth calibration application principle

The figure below provides a block diagram of the smooth digital calibration application.

Figure 21. Block diagram of a smooth digital calibration



This application is based on the following 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 s.
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 TIM2_CCR1 register. Then TIM2 counter is reset.
4. The Cortex-M4 core gets the TIM2_CCR1 value, compares it with the number of CLK_RTC cycles expected in 32 s and processes the comparison results to update the CALP and CALM[8:0] in RTC_CALR.

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 s.
3. Measure the output frequency. Depending on 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:

- CALP and CALM[8:0] in RTC_CALR
- TIM2_CCR1 register (contains the number of CLK_RTC cycles within a 32 s window)

4.3.5 Porting suggestions

The software example expects that the TIM2 is more than 20 bit, 32 bit in case of L476RB. With products featuring only 16-bit timers it is recommended to keep track of 16-bit timer overflows using the software and perform the calibration with 16-th overflow to compensate for the missing 4 bits.

5 STM32L0 API and tampering detection application example

5.1 STM32CubeL0 firmware libraries

The RTC 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 in \STM32Cube_FW_L0_Vx.y.z\Projects\STM32L053R8-Nucleo\Examples\RTC

5.2 X-CUBE-RTC for tamper detection

X-CUBE-RTC shows an 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)
- 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

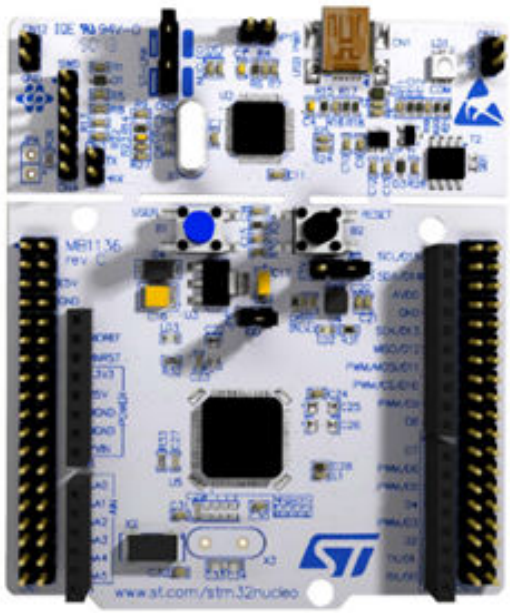
The application flowchart is shown in Figure 18.

5.3 Tamper detection application example (STM32L0)

5.3.1 Hardware setup

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

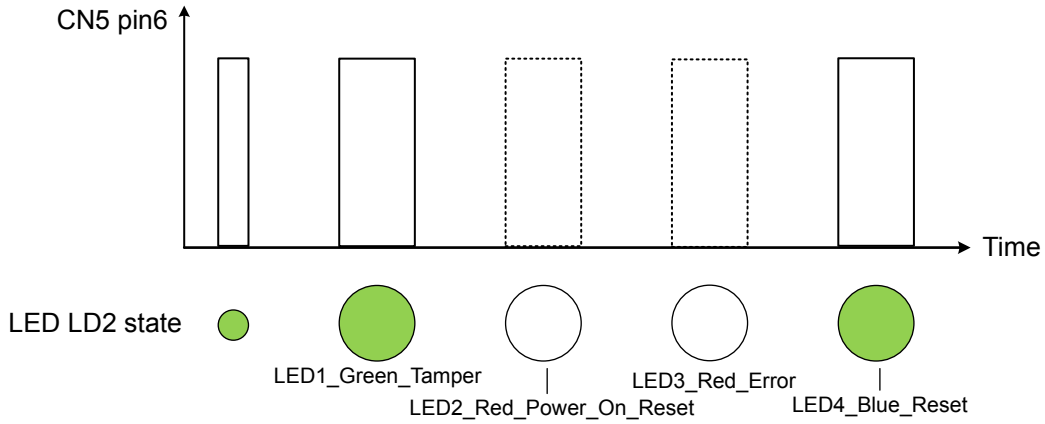
Figure 22. NUCLEO-L053R8 board



Note: this picture is not contractual.

Supply the board using the USB cable. In this example, the B1 push button is used 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 23. LED LD2 behavior



5.3.2 Software setup

Open the project under the user favorite IDE (integrated development environment). For IAR Embedded Workbench, 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_Tamper.

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

In IAR Embedded Workbench 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 with:

- 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 MDK-ARM development tools. To open the *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 3.3, LED1, LED2, LED3 and LED4 are replaced by the following integers:

- `uint32_t LED1_Green_Tamper_event_detected`: set when tamper1 event is detected (LED1 equivalent)
- `uint32_t LED2_Red_Power_On_Reset_occurred`: set when power-on reset is detected (LED2 equivalent)
- `uint32_t LED3_Red_Error`: set in errors cases (LED3 equivalent)
- `uint32_t LED4_Blue_Reset_occurred`: set when reset is detected (B2 black push button, LED4 equivalent)

The user can observe these private variables using the live watch feature in the debugger, or 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

Step 1

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

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

Meaning	Status	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

Step 2

Push B1 to simulate a tamper event.

Table 19. Tamper detection status when a tamper event is detected (after a power-on reset)

Meaning	Status	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

Step 3

Push B2 to simulate a reset.

Table 20. Tamper detection status when a reset is detected

Meaning	Status	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

Step 4

Push B1 to simulate a tamper event.

Table 21. Tamper detection status when a tamper event is detected (after a reset)

Meaning	Status	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 smooth digital calibration application example

6.1 STM32CubeL5 firmware libraries for smooth calibration application

The timer peripheral (TIM) comes with:

- A firmware driver API abstracting TIM 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 in \STM32Cube_FW_L5_Vx.y.z\Projects\ NUCLEO-L552ZE-Q\Examples\TIM

The RTC comes with:

- A firmware driver API abstracting TIM 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 in \STM32Cube_FW_L5_Vx.y.z\Projects\ NUCLEO-L552ZE-Q\Examples\RTC

6.2 X-CUBE-RTC for smooth calibration

X-CUBE-RTC shows an implementation of the smooth digital calibration feature of the RTC (using also the GPTIMER HAL API).

In this example, the 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:

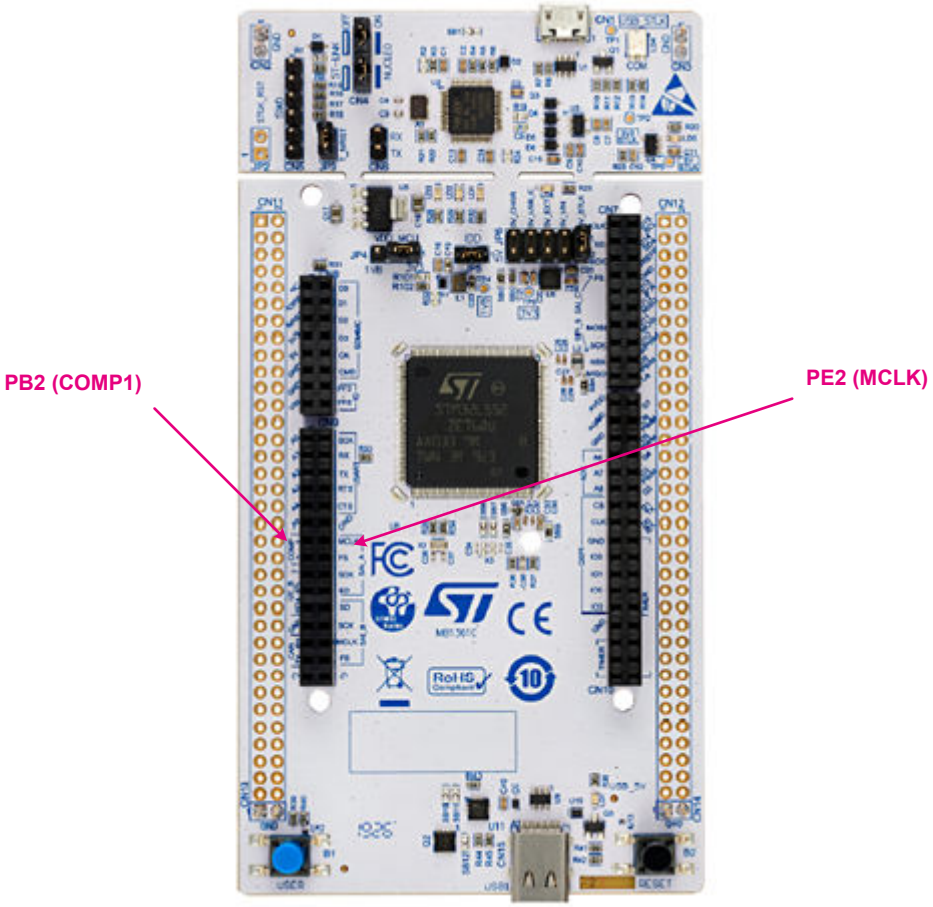
- A GPTIMER (general-purpose timer) in Master mode using channel 1 in output compare mode (and using an external trigger clock)
- A GPTIMER in Master mode using channel 1 in input capture mode and channel 2 in output compare mode
  This timer is also configured to use the LSE clock thanks to the TIM option register.
- The RTC with the CALR register programming
- GPIOs to connect the input 1 Hz reference clock and the “corrected” RTC_OUT_CALIB clock

6.3 Smooth calibration application example (STM32L5)

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 24. NUCLEO-L552ZE-Q board



Note: this picture is not contractual.

Supply the board using the USB cable and perform the following actions:

- 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 (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 favorite IDE. For IAR Embedded Workbench, open the `RTC_SmoothCalib.eww` file. For 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).

6.3.3 Smooth calibration principle

Figure 21 provides a block diagram of the smooth digital calibration application.

This application is based on the following 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 s.
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 TIM2_CCR1 register. Then, TIM2 counter is reset.
4. The Cortex-M33 core gets the TIM2_CCR1 value, compares it with the number of CLK_RTC cycles expected in 32 s and processes the comparison results to update the CALP and CALM[8:0] in RTC_CALR.

The calibration is continuous.

6.3.4 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 s.
3. Measure the output frequency. Depending on 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:

- CALP and CALM[8:0] in RTC_CALR
- TIM2_CCR1 register (contains the number of CLK_RTC cycles within a 32 s window)

7 STM32L5 API and synchronization application example

STM32CubeL5 firmware libraries are the same as in Section 6.1.

7.1 X-CUBE-RTC for synchronization application

X-CUBE-RTC shows an implementation of the RTC calendar synchronization with the HSE (provided by an external signal: HSE bypass feature used)

(STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\NUCLEO-L552ZE-Q\RTC_Synchronization).

The firmware implements:

- TIM1 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 SHIFTR register programming
- GPIO: RTC_OUT CALIB to check if the RTC 1 Hz output 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) is used to visualize the 1-s period and the good working of the application.

7.2 Synchronization application example (STM32L5)

7.2.1 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.

The TIM1 clock source is HSE. However, the 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 to input an external 16 MHz 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, right 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 the board by verifying a 1 Hz signal is output on PB2 (RTC_OUT CALIB output).

7.2.2 Software setup

Open the project under the user favorite IDE. For IAR Embedded Workbench, open the `RTC_Synchronization.eww` file. For 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.

7.2.3 Synchronization application principle

This application is based on the following steps:

1. TIM1 is configured and launched to generate an interruption each second (APB2 timer clock = 100 MHz).
2. RTC is configured and launched including its calendar (RTC clock = LSE).
3. The 1-second period counted by the TIM1 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 subsecond register, the advance/delay of the RTC clock can be analysed on the TIM1 1-second period.
5. With the SHIFTR register, this advance or delay is compensated.

The synchronization is continuous.

7.2.4 Run time observations

The following steps are needed to check the correct functionality:

- Set the SYNCHRO_ACTIVATED to 0 and observe the `time_elapsed` variable deriving (getting away from 255/1 sec). The user can monitor it in Debug mode.
- 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 uncertainty must be attenuated with synchronization that is efficient if the clock used (external 16 MHz in the present case) has a better uncertainty. Using an HSE X3 crystal is not so efficient if the LSE crystal has the same uncertainty.

To check that the application runs 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

STM32CubeL5 firmware libraries are the same as in Section 6.1.

8.1 X-CUBE-RTC for reference clock detection application

X-CUBE-RTC shows an implementation of the reference clock detection (STM32CubeExpansion_AN4759_RTC_STM32L_Vx.y.z\Projects\NUCLEO-L552ZE-Q\RTC_RefClockDetection).

The firmware implements:

- TIM1 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.

8.2 Reference clock detection application example

8.2.1 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 the board by verifying a 1 Hz signal is output on PB2 (RTC_OUT CALIB output).

8.2.2 Software setup

Open the project under the user favorite IDE. For IAR Embedded Workbench, open the `RTC_Synchronization.eww` file. For 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.

8.2.3 Reference clock detection principle

This application performs the following steps:

1. TIM1 is configured and launched to generate an interruption each second (APB2 timer clock = 100 MHz).
2. RTC is configured and launched including 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 TIM1 interruptions allows the measure of the RTC period. The HSI (TIM1 clock source) is less precise than the LSE (RTC clock source). The LSE 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.

8.2.4 Run time observations

The following steps are needed to check the correct functionality:

- Set the CLK_REF_ACTIVATED to 0 and observe the `time_elapsed` variable deriving (getting away from 255/1 s). The user can monitor it in Debug mode.
- 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 uncertainty must be attenuated with the reference clock detection and synchronization.

Moreover, to check that the application runs 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 X-CUBE-RTC for internal tamper detection application

X-CUBE-RTC shows an implementation of the internal tamper detection in the folder:

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 and Section 7. 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 Embedded Workbench, open the `RTC_Synchronization.eww` file. For MDK-ARM, open the `RTC_Synchronization.uvprojx` file. For STM32CubeIDE, open the `.cproject` file.

The X-CUBE-RTC 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 LED2 PB7 if in interrupt mode, is turned on.

RTC_DR and RTC_TR allow the contents of the RTC calendar to be observed in debug mode.

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

Revision history

Table 22. **Document revision history**

Date	Version	Changes
26-May-2016	1	Initial release.
25-Oct-2016	2	Updated: • Table 1: Applicable products adding product series and part number. • Figure 3: STM32F2, STM32F4, STM32L0 and STM32L1 Series RTC clock sources. • Figure 4: STM32F0, STM32F3, STM32F7, STM32L4 and STM32WB Series RTC clock sources • Table 3: Calendar clock equal to 1 Hz with different clock sources • Section 1.4: Digital smooth calibration adding the RTC calibration basics • Table 14: Advanced RTC features • Section 4.3.3: LED meaning • new Section 5: STM32L4 API and digital smooth calibration application example • Section 6.3.1: Hardware setup • Section 6.3.2: Software setup • Section 7: Reference documentation
11-May-2017	3	Updated: • Figure 4: STM32F0, STM32F3, STM32F7, STM32L4 and STM32WB Series RTC clock sources • Table 3: Calendar clock equal to 1 Hz with different clock sources adding note • Section 1.4.1: RTC calibration basics • Section 1.11.1: RTC register write protection • 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.
12-Feb-2020	5	Updated: • Introduction • Table 1: Applicable products • Section 1: Overview of the STM32 MCUs advanced RTC: new Table 2: RTC/TAMP types versus features and new Table 3: RTC type versus STM32 products • Table 4: Advanced RTC2 features • new Table 5: Advanced RTC3 features • Table 6: Steps to initialize the calendar • Footnote 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 consumption
12-Feb-2020 (cont'd)	5	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

Date	Version	Changes
		• Section 2.10.7: 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
4-Jun-2021	6	Updated with the STM32U5 Series: • Introduction • Table 2. RTC/TAMP types • Table 3. RTC type on STM32 MCUs • STM32H7 in Table 4. Advanced features for RTC2 type • STM32U5 in Table 5. Advanced features of RTC3 type • Notes in Figure 1. RTC calendar fields • Example in Section 2.3.1.2 Configure the alarm behavior using the MSKx bits • Note in Section 2.3.2 Alarm subsecond configuration • Note in Section 2.4.2.3 Periodic timebase/wakeup configuration for clock configuration 3 • Caution in Section 2.6 Synchronize the RTC • Section 2.10.3 Action on tamper detection event • Section 2.10.5 Internal tamper detection (RTC3 only) • Section 2.11 Backup registers • Section 2.13.2 Enter/exit initialization mode • Titles and structure of to Section 3 to Section 9 Added: • Section 2.2 Binary and mixed modes (RTC3 only) • Section 2.10.6 Potential detection management (RTC3 only)
20-Sep-2021	7	Updated: • Section 2.1.5: Calendar firmware examples • Figure 9. Smooth calibration block for RTC2 type
08-Nov-2022	8	Updated: • Section Introduction to add the STM32C0 and the STM32H5 Series. • Section 2: Advanced RTC features to add data for STM32U5, STM32L5, STM32L41/42/4P5/4Q5, STM32G4, STM32G0, STM32H7A3/7B3, STM32H5 and STM32C0 in Table 5. Advanced features of RTC3 type. • Section 2.10.5: Internal tamper detection (RTC3 only) to update the internal tamper list. • Section 2.10.7: Tamper detection firmware examples • Section 2.14.5: Low-power management firmware examples
12-Mar-2024	9	Updated: • Document title • Table 1. Applicable products • Table 3. RTC type on STM32 MCUs • Table 4. Advanced features for RTC2 type • Table 5. Advanced features of RTC3 type • Section 2.13.3: RTC clock synchronization
30-Jan-2025	10	Updated: • Table 1. Applicable products • Table 3. RTC type on STM32 MCUs • Table 4. Advanced features for RTC2 type and Table 5. Advanced features of RTC3 type • Section 2.8: RTC prescaler adjustment with LSI measurements • Section 2.11: Backup registers • Section 2.14.5: Low-power management firmware examples

Contents

List of tables

List of figures