Автор: Сільчук Віталій, ДК-61
E-mail: vitalii.silchuk@gmail.com

Робота з модулем RTC в мікроконтролерах STM32 серії F4

В даному документі описана робота з модулем RTC в мікроконтролерах STM32 серії F4. Працездатність наведених прикладів була перевірена на налагоджувальній платі STM32F401 Nucleo.

Загалом, варто звернути увагу на такий документ, як AN3371 Application note — Using the hardware real-time clock (RTC) in STM32 F0, F2, F3, F4 and L1 series of MCUs **[1]**, в котрому, як зрозуміло з його назви, компанія STMicroelectronics описує принципи роботи з модулем RTC, котрий є вбудованим (реалізованим апаратно) в мікроконтролерах STM32 серій F0, F2, F3, F4 та L1. Також стане у нагоді Reference manual — RM0368 Reference manual for STM32F401xB/C and STM32F401xD/E advanced Arm®-based 32-bit MCUs **[2]**, котрий містить повну інформацію про те, як працювати з пам’яттю та периферією даних мікроконтролерів.

Зміст:

1. Загальні відомості про влаштування та особливості роботи модуля RTC в мікроконтролерах STM32 серії F4.
2. Необхідні для початку роботи структури даних.
3. Функції для роботи з структурами даних, котрі зберігають значення дати та часу.
4. Початкова ініціалізація-запуск роботи модуля RTC.
5. Конфігурація роботи модуля RTC, налаштування формату відображення часу.
6. Налаштування-оновлення поточних значень дати та часу.
7. Запит та отримання значень поточних дати та часу.
8. Початкова ініціалізація роботи режиму будильника (Alarm).
9. Налаштування-оновлення значень дати та часу спрацювання сигналу будильника.
10. Деактивація запланованого сигналу будильника.
11. Обробка переривань, що генеруються при спрацюванні сигналу будильника.
12. Запит та отримання значень дати та часу спрацювання сигналу будильника.
13. Налаштування wakeup-режиму роботи модуля RTC.
14. Обробка переривань, що генеруються в wakeup-режимі роботи модуля RTC.

1. Загальні відомості про влаштування та особливості роботи модуля RTC в мікроконтролерах STM32 серії F4.

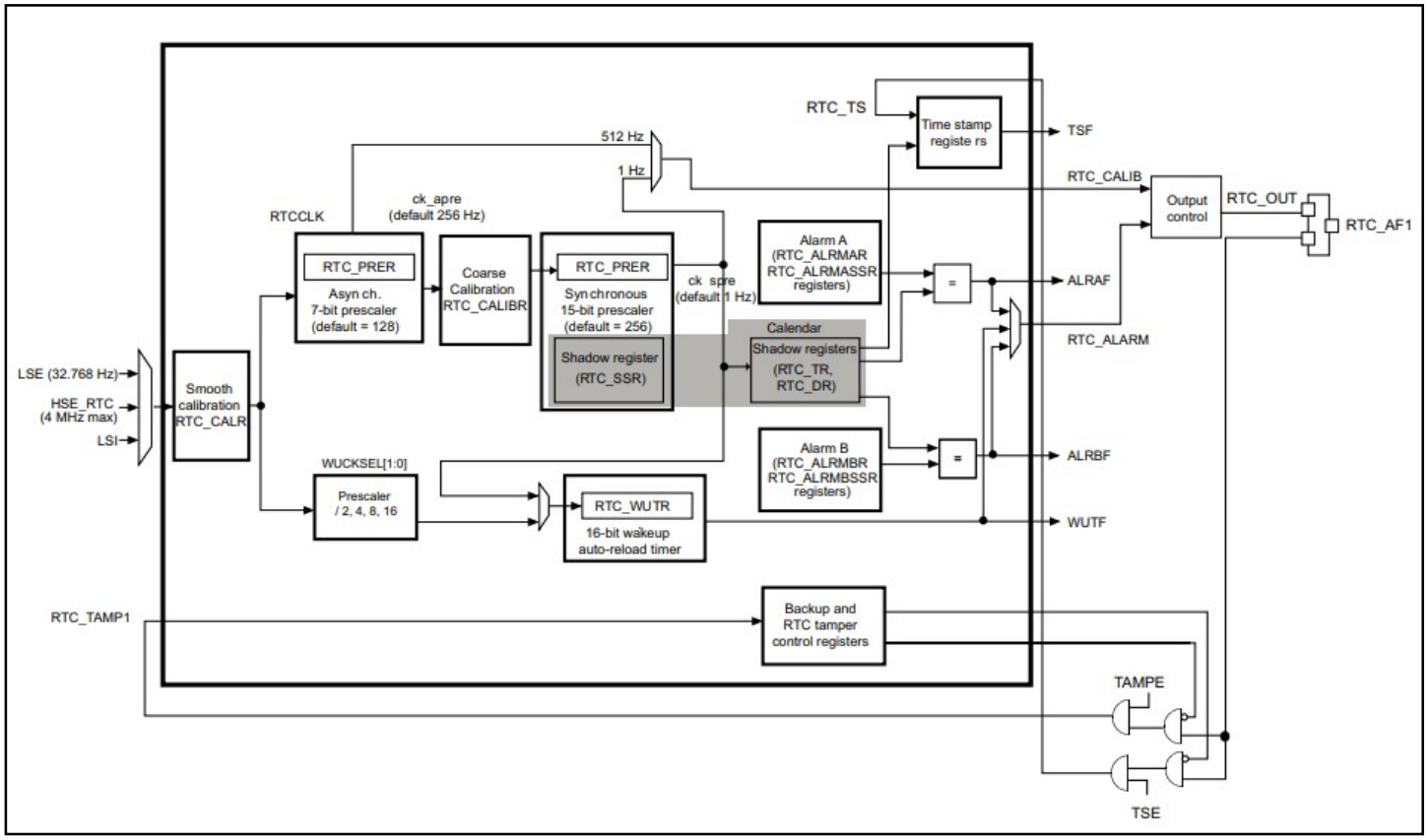

Рисунок 1 — Влаштування — структурна схема — модуля RTC в мікроконтролерах STM32 серії F4

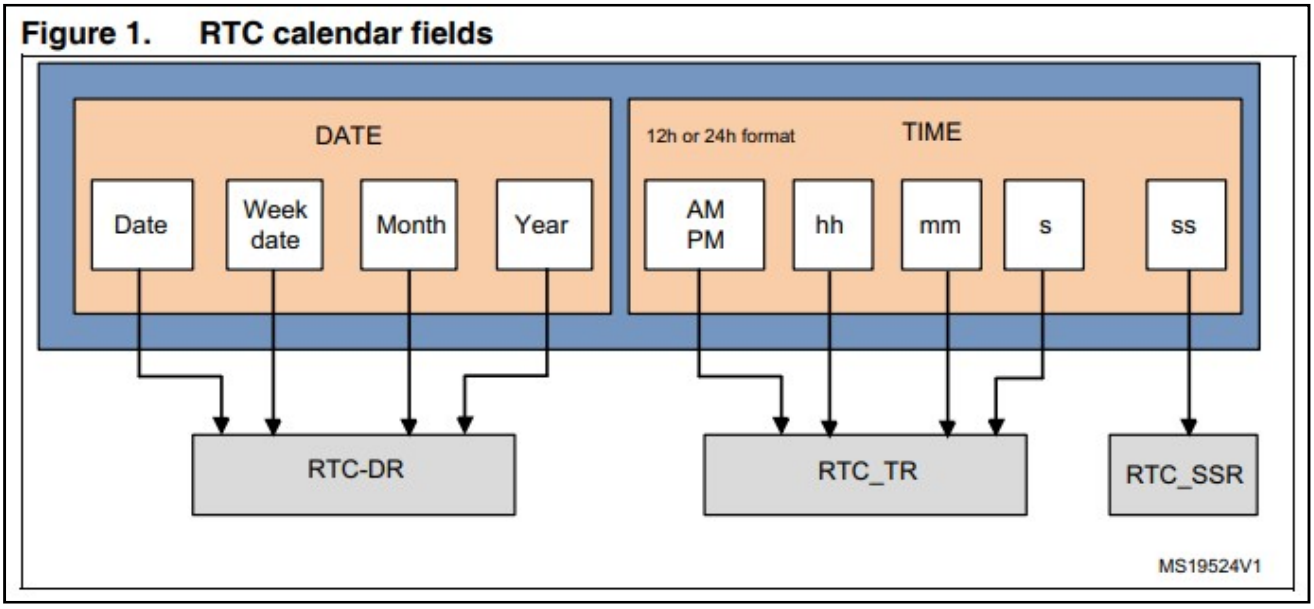

Рисунок 2 — Регістри дати та часу

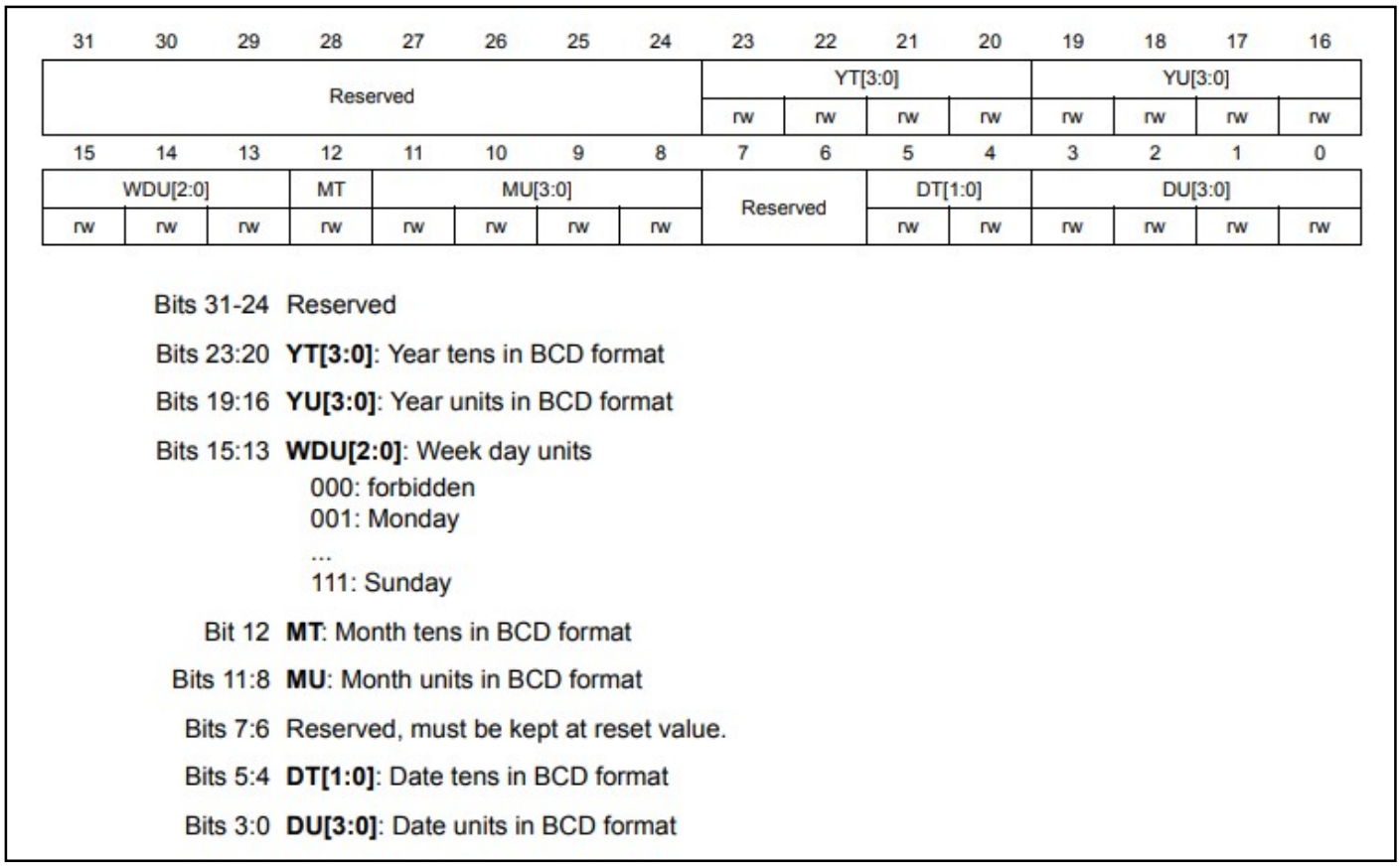

Рисунок 3 — Структура регістру дати RTC_DR

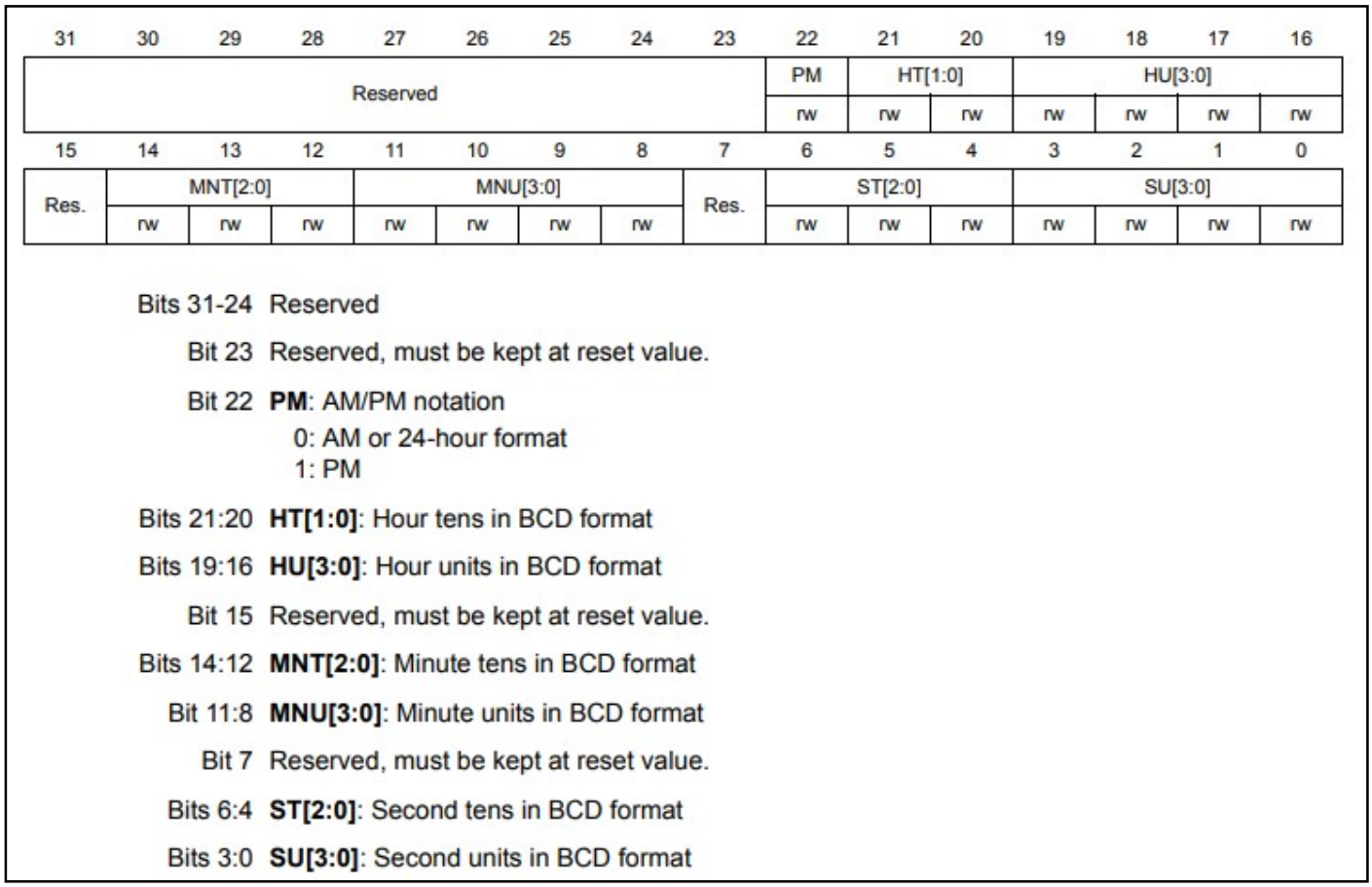

Рисунок 4 — Структура регістру часу RTC_TR

Figure 5. Prescalers from RTC clock source to calendar unit

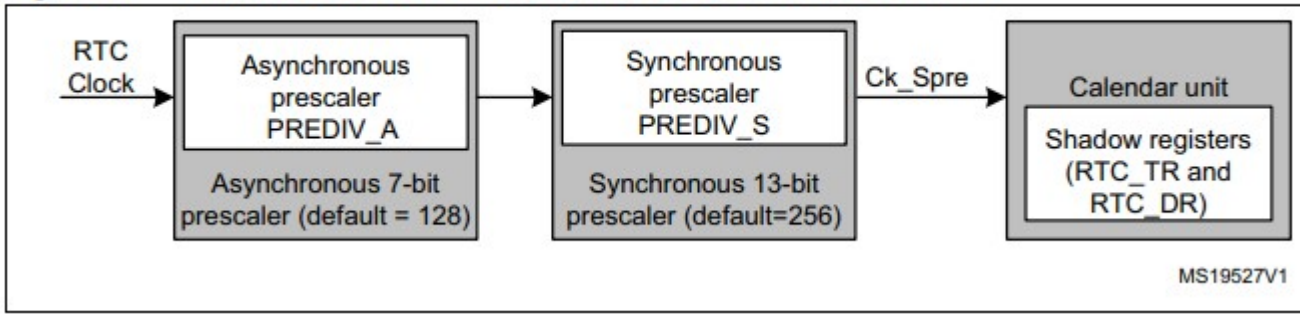

Рисунок 5 — Переддільники сигналу тактування модуля RTC

The formula to calculate ck_spre is:

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

where:

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

Table 3 shows several ways to obtain the calendar clock (ck_spre) = 1 Hz.

Table 3. Calendar clock equal to 1 Hz with different clock sources

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

Рисунок 6 — Розрахунок значення частоти сигналу, котрий буде тактувати основний лічильник модуля RTC, після проходження через синхронний та асинхронний переддільники

Загалом, в мікроконтролерах STM32 серії F4 наявні наступні додаткові функції модуля RTC:

- Alarm A;
- Alarm B;
- Wakeup interrupt;
- Time-stamp;
- Tamper detection;

Alarm A/B — використовується для реалізації будильників/таймерів — можна налаштувати дату та час спрацювання сигналу, в котрий буде згенеровано відповідне переривання.

Wakeup interrupt — виведення мікроконтролеру з режиму сну — можна налаштувати значення модулю wakeup-лічильника, дійшовши до якого, відбуватиметься генерація відповідного переривання, що виводитиме мікроконтролер зі сну.

Time-stamp — призначена для визначення точного часу настання зовнішньої події — для її активації потрібна наявність зовнішнього сигналу, по фронту якого значення рахункових регістрів зберігаються в регістрах RTC_TSTR і RTC_TSDR.

Tamper detection — функція виявлення несанкціонованого доступу до пристрою — передбачає наявність зовнішнього контакту, підключеного до лінії RTC_AF1 або RTC_AF2, за значенням якого постійно слідкуватиме модуль RTC — цей пін генерує подію сигналізації про втручання в цілісність корпусу пристрою, що призводить до скидання інформації в регістрах резервного зберігання (backup memory registers).

RTC features			F0 series	F3 series	F2 series	ULPM density	F4 series	ULPH density
Prescalers	Asynchronous		X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)
	Synchronous		X (15 bits)	X (13 bits)	X (13 bits)	X (13 bits)	X (15 bits)	X (15 bits)
Calendar	Time	12/24 format	X	X	X	X	X	X
		Hour, minutes and seconds	X	X	X	X	X	X
		Sub-second	X	X			X	X
	Date		X	X	X	X	X	X
	Daylight operation		X	X	X	X	X	X
	Bypass the shadow registers		X	X			X	X
Alarm	Alarms available	Alarm A	X	X	X	X	X	X
		Alarm B		X	X	X	X	X
	Time	12/24 format	X	X	X	X	X	X
		Hour, minutes and seconds	X	X	X	X	X	X
		Sub-second	X	X			X	X
	Date or week day		X	X	X	X	X	X
Tamper detection	Configurable input mapping		X	X	X		X	
	Configurable edge detection		X	X	X	X	X	X
	Configurable Level detection (filtering, sampling and precharge configuration on tamper input)		X	X			X	X
	Number of tamper inputs		2 inputs	2 inputs	2 inputs / 1 event	1 input / 1 event	2 inputs / 2 events	3 inputs / 3 events

Рисунок 7 — Можливості модуля RTC, реалізованого в мікроконтролерах STM32 різних серій

RTC features			F0 series	F3 series	F2 series	ULPM density	F4 series	ULPH density
Time Stamp	Configurable input mapping		X	X	X		X	
	Time	Hours, minutes and seconds	X	X	X	X	X	X
		Sub-seconds	X	X			X	X
	Date		X	X	X	X	X	X
	Active Time Stamp on tamper detection event		X	X	X	X	X	X
RTC Outputs	AFO_Alarm	Alarm event	X	X	X	X	X	X
		Wakeup event	X	X	X	X	X	X
	AFO_Calib	512 Hz	X	X	X	X	X	X
		1 Hz	X	X			X	X
RTC Calibration	Coarse Calibration			X		X	X	X
	Smooth Calibration		X	X			X	X
Synchronizing the RTC			X	X			X	X
Reference clock, detection			X	X	X	X	X	X
Backup registers	Powered-on Vbat		X	X	X		X	
	Reset on a tamper detection		X	X	X	X	X	X
	Reset when Flash readout protection is disabled		X	X		X		X
	RTC clock source configuration register		RCC_BDCR	RCC_BDCR	RCC_BDCR	RCC_CSR	RCC_BDCR	RTC_CSR
	Number of backup registers		5	20	20	20	20	32

Рисунок 8 — Можливості модуля RTC, реалізованого в мікроконтролерах STM32 різних серій

Table 67. Interrupt control bits

Interrupt event	Event flag	Enable control bit	Exit the Sleep mode	Exit the Stop mode	Exit the Standby mode
Alarm A	ALRAF	ALRAIE	yes	yes(1)	yes(1)
Alarm B	ALRBF	ALRBIE	yes	yes(1)	yes(1)
Wakeup	WUTF	WUTIE	yes	yes(1)	yes(1)
TimeStamp	TSF	TSIE	yes	yes(1)	yes(1)
Tamper1 detection	TAMP1F	TAMPIE	yes	yes(1)	yes(1)

1. Wakeup from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

Рисунок 9 — Контрольні біти подій/переривань

Table 68. RTC register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RTC_TR	Reserved									PM	HT [1:0]		HU[3:0]				Reserved	MNT[2:0]			MNU[3:0]				Reserved	ST[2:0]			SU[3:0]			
	Reset value										0	0	0	0	0	0	0		0	0	0	0	0	0	0		0	0	0	0	0	0	0
0x04	RTC_DR	Reserved								YT[3:0]				YU[3:0]				WDU[2:0]				MT	MU[3:0]			Reserved	DT [1:0]			DU[3:0]			
	Reset value																	0	0	1		0	0	0	0	1	0	0		0	0	0	1
0x08	RTC_CR	Reserved								COE	OSEL [1:0]			POL	COSEL	BKP	SUB1H	ADD1H	TSIE	WUTIE	ALRBIE	ALRAIE	TSE	WUTE	ALRBE	ALRAE	DCE	FMT	BYPSHAD	REFCKON	TSEDGE	WCKSEL [2:0]		
	Reset value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	RTC_ISR	Reserved																			TAMP1F	TSOVF	TSF	WUTF	ALRBF	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUTWF	ALRBWF	ALRAWF
	Reset value																				0	0	0	0	0	0	0	0	0	0	0	1	0	1
0x10	RTC_PRER	Reserved									PREDIV_A[6:0]							Reserved	PREDIV_S[14:0]															
	Reset value										1	1	1	1	1	1	1		0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0x14	RTC_WUTR	Reserved																WUT[15:0]															
	Reset value																	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x18	RTC_CALIBR	Reserved																								DCS	Reserved	DC[4:0]					
	Reset value																									0		0	0	0	0	0	

Рисунок 10 — Карта регістрів модуля RTC в мікроконтролерах STM32 серії F4

Table 68. RTC register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	RTC_ALRMAR	MSK4	WDSEL	DT [1:0]		DU[3:0]				MSK3	PM	HT [1:0]		HU[3:0]				MSK2	MNT[2:0]			MNU[3:0]				MSK1	ST[2:0]			SU[3:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	RTC_ALRMBR	MSK4	WDSEL	DT [1:0]		DU[3:0]				MSK3	PM	HT [1:0]		HU[3:0]				MSK2	MNT[2:0]			MNU[3:0]				MSK2	ST[2:0]			SU[3:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	RTC_WPR	Reserved																								KEY[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x28	RTC_SSR	Reserved																SS[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x30	RTC_TSTR	Reserved										PM	HT[1:0]		HU[3:0]			Reserved	MNT[2:0]			MNU[3:0]				Reserved	ST[2:0]			SU[3:0]			
	Reset value											0	0	0	0	0	0		0	0	0	0	0	0	0		0	0	0	0	0	0	0
0x38	RTC_TSSSR	Reserved																SS[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x3C	RTC_ CALR	Reserved																CALP	CALW8	CALW16	Reserved					CALM[8:0]							
	Reset value																	0	0	0						0	0	0	0	0	0	0	0

Рисунок 11 — Карта регістрів модуля RTC в мікроконтролерах STM32 серії F4

0x40	RTC_TAFCR	Reserved													ALARMOUTTYPE	TSINSEL	TAMP1INSEL	TAMPPUDIS	TAMPPRCH[1:0]		TAMPFLT[1:0]		TAMPFREQ[2:0]			TAMPTS	Reserved				TAMPIE	TAMP1ETRG	TAMP1E
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0					0	0	0
0x44	RTC_ALRMASSR	Reserved				MASKSS[3:0]				Reserved									SS[14:0]														
	Reset value					0	0	0	0										0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x48	RTC_ALRMBSSR	Reserved				MASKSS[3:0]				Reserved									SS[14:0]														
	Reset value					0	0	0	0										0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x50 to 0x9C	RTC_BKP0R	BKP[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	to RTC_BKP19R	BKP[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 12 — Карта регістрів модуля RTC в мікроконтролерах STM32 серії F4

Годинник реального часу (Real-time clock, RTC) являє собою незалежний BCD-таймер/лічильник.

RTC забезпечує годинник /календар, два програмованих Alarm-переривання та періодичне програмоване Wakeup-переривання, що виводить мікроконтролер з режиму сну, для керування режимами низького енергоспоживання.

Два 32-бітні регістри містять секунди, хвилини, години (12- або 24-годинний формат), день (день тижня), дату (день місяця), місяць та рік, у BCD-форматі.

Значення субсекунд (sub-seconds) доступне у двійковому форматі.

Компенсації кількості днів у 28-, 29- (високосний рік), 30- та 31-денних місяцях виконуються автоматично. Також можна виконати компенсацію літнього часу.

Додаткові 32-бітні регістри містять програмовані підсистеми тривоги, секунди, хвилини, години, день та дата.

Доступна функція цифрового калібрування для компенсації будь-якого відхилення точності кварцового генератора.

Після скидання backup-домену всі регістри RTC захищені від можливого паразитарного запису.

Поки напруга живлення залишається в робочому діапазоні, лічильник модуля RTC ніколи не зупиняється, незалежно від стану пристрою (Run-режим, режим низького енергоспоживання, режим скидання).

Загалом, дещо повторюючись, варто відмітити, що регістри модуля RTC розташовані в окремій області пам'яті, що має можливість зовнішнього живлення від

батареї. Регістри модуля RTC також оснащені додатковим захистом від запису, що забезпечує неможливість випадкового пошкодження інформації в них.

Крім часових регістрів в модулі виконані 20 регістрів резервування призначених для користувача даних з розрядністю 32 біти — так звана «backup memory». Ці регістри НЕ очищуються по сигналу «Скидання» при наявності зовнішнього джерела напруги, що дозволяє зберігати в них важливу інформацію.

Перелік основних регістрів модуля RTC та короткий опис їх функцій (принаймні, в контексті використовуваних в наведених в даному документі функцій):

- RTC_TR — регістр з поточним значенням часу (запис можливий тільки в режимі ініціалізації);
- RTC_DR — регістр з поточним значенням дати (аналогічно — запис можливий тільки в режимі ініціалізації);
- RTC_CR — основні налаштування режиму роботи годинника;
- RTC_ISR — регістр ініціалізації та статусу, з його допомогою можна ввести годинник в однойменний режим, а також відстежувати стан «прапорців» виникнення подій;
- RTC_PRER — внутрішній переддільник сигналу тактування;
- RTC_CALIBR — підлаштування годинника;
- RTC_WPR — розблокування захисту від запису;
- RTC_WUTR — 16-розрядний wakeup-таймер з автоматичним перезавантаженням, використовується для налаштування wakeup-режиму модуля RTC (для виходу з режиму сну), та генерації відповідного переривання;
- RTC_ALRMAR — налаштування функціонування alarm-режиму модуля — вибір полів поточних дати та часу, які будуть братись до уваги при порівнянні з заданим часом alarm-події;

Двійково-десятковий код — binary-coded decimal, BCD — форма запису раціональних чисел, коли кожен десятковий розряд числа записується у вигляді його чотирьохбітного двійкового коду.

За допомогою 4 біт можна закодувати 16 цифр. З них використовуються 10. Інші 6 комбінацій в BCD-коді являються забороненими.

Таблиця відповідності двійково-десяткового коду та десяткових цифр наведена на рисунку 13.

Двоично-десятичный код				Десятичный код
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Рисунок 13 — Відповідність двійково-десяткового коду десятковим цифрам

Таким чином, для прикладу, десяткове число 311_{10} буде записано в двійковому коді як $1\ 0011\ 0111_2$, а в BCD-коді — як $0011\ 0001\ 0001_{BCD}$.

2. Необхідні для початку роботи структури даних.

Для зручної роботи з модулем годинника реального часу, було вирішено створити структури даних, поля яких міститимуть значення дати та часу.

```
typedef struct s_RTC_struct_full {
	uint8_t year_tens;		// 20[1]9
	uint8_t year_units;		// 201[9]
	uint8_t week_day;		// 001 for Monday, 111 for Sunday
	uint8_t month_tens;		// [1]2
	uint8_t month_units;		// 1[2]
	uint8_t date_tens;		// [2]5
	uint8_t date_units;		// 2[5]
	uint8_t hour_tens;		// [0]0
	uint8_t hour_units;		// 0[0]
	uint8_t minute_tens;		// [0]3
	uint8_t minute_units;		// 0[3]
	uint8_t second_tens;		// [1]5
	uint8_t second_units;		// 1[5]
} RTC_struct_full;


typedef struct s_RTC_struct_brief {
	uint8_t years;		// 2019
```

```
    uint8_t week_day;       // 001 for Monday, 111 for Sunday
    uint8_t months;         // 12
    uint8_t date;           // 25
    uint8_t hours;          // 0
    uint8_t minutes;        // 03
    uint8_t seconds;        // 15
} RTC_struct_brief;
```

Структура `RTC_struct_full` призначена для зберігання значень дати та часу в «розширеному» форматі — десятки — окремо, одиниці — окремо. Таке рішення випливає з того, що саме в такому, подібному, форматі — в двійково-десятковому коді — значення дати та часу зберігаються в регістрах модуля RTC, що видно з Рисунку 3 та Рисунку 4.

Структура `RTC_struct_brief` призначена для зберігання значень дати та часу в «стислому» форматі — фактично, мається на увазі звичайний десятковий формат.

При записі значень дати та часу в регістри модуля RTC, використовується «повна» структура. При введенні значень дати та часу за допомогою пристроїв введення, для їх збереження використовується «стисла» структура — значення з якої згодом, за потреби, переносяться в «повну» структуру, шляхом розділення їх за допомогою арифметичних операцій, на одиниці та десятки.

3. Функції для роботи з структурами даних, котрі зберігають значення дати та часу.

Для роботи з структурами даних `RTC_struct_full` та `RTC_struct_brief`, котрі зберігають значення дати та часу, використовуються функції: `fill_struct_default`, `btn_fill_date_fields`, `fill_RTC_struct_full`.

3.1. Функція `fill_struct_default`.

```
void fill_struct_default(RTC_struct_brief volatile *br_data,
                         RTC_struct_full volatile *f_data)
{
    // default date: FRI 27.12.19 04:49:00
    f_data->year_tens = 0x1;
    f_data->year_units = 0x9;
    f_data->week_day = 0x5;
    f_data->month_tens = 0x1;
    f_data->month_units = 0x2;
    f_data->date_tens = 0x2;
    f_data->date_units = 0x7;
    f_data->hour_tens = 0x0;
```

```
	f_data->hour_units = 0x4;
	f_data->minute_tens = 0x4;
	f_data->minute_units = 0x9;
	f_data->second_tens = 0x0;
	f_data->second_units = 0x0;

	br_data->years = 0x19;
	br_data->week_day = 0x5;
	br_data->months = 0x12;
	br_data->date = 0x27;
	br_data->hours = 0x04;
	br_data->minutes = 0x49;
	br_data->seconds = 0x00;
   }
```

Як видно, дана функція реалізує заповнення відповідих полів структур `RTC_struct_brief` та `RTC_struct_full` значеннями, що являють початкові значення часу «за замовчуванням». Як можна бачити, поля «повної» структури заповнюються окремо одиницями та десятками значень часу.

3.2. Функція `btn_fill_date_fields`.

```
void btn_fill_date_fields(RTC_struct_brief volatile *br_data, uint32_t field_cnt,
int32_t add_sub, uint32_t is_alarm_cfg, uint32_t is_clock_cfg)
{
	if (is_alarm_cfg)
	{
		switch(field_cnt)
		{
			case 0:
				br_data->date = (br_data->date >= 1 && br_data->date <= 30) ?
(br_data->date + add_sub) : 1;
				break;
			case 1:
				br_data->hours = (br_data->hours <= 22) ? (br_data->hours +
add_sub) : 0;
				break;
			case 2:
```

```
                        br_data->minutes = (br_data->minutes <= 58) ? (br_data->minutes
+ add_sub) : 0;

                        break;

                  default:

                        br_data->date = br_data->date;

                        br_data->hours = br_data->hours;

                        br_data->minutes = br_data->minutes;

            }

      }

      else if (is_clock_cfg)

      {

            switch(field)     // switch(field_cnt) – варіант для окремого використання
                              // даної функції – в даному випадку такий варіант працює,
                              // так як field – це глобальна змінна, що передається в
                              // програмі в якості аргумента field_cnt

            {

                  case 0:

                        br_data->week_day = (br_data->week_day >= 1 && br_data-
>week_day <= 6) ? (br_data->week_day + add_sub) : 1;

                        break;

                  case 1:

                        br_data->hours = (br_data->hours <= 22) ? (br_data->hours +
add_sub) : 0;

                        break;

                  case 2:

                        br_data->minutes = (br_data->minutes <= 58) ? (br_data->minutes
+ add_sub) : 0;

                        break;

                  case 3:

                        br_data->seconds = (br_data->seconds <= 58) ? (br_data->seconds
+ add_sub) : 0;

                        break;

                  case 4:

                        br_data->date = (br_data->date >= 1 && br_data->date <= 30) ?
(br_data->date + add_sub) : 1;

                        break;

                  case 5:
```

```
                        br_data->months = (br_data->months >= 1 && br_data->months <=
11) ? (br_data->months + add_sub) : 1;
                        break;
                  case 6:
                        br_data->years = (br_data->years <= 98) ? (br_data->years +
add_sub) : 0;
                        break;

                  default:
                        br_data->week_day = br_data->week_day;
                        br_data->hours = br_data->hours;
                        br_data->minutes = br_data->minutes;
                        br_data->seconds = br_data->seconds;
                        br_data->date = br_data->date;
                        br_data->months = br_data->months;
                        br_data->years = br_data->years;
            }
      }
}
```

Ця функція необхідна для «проходу» по полях структури типу `RTC_struct_brief`, щоб заповнити їх значеннями, котрі користувач вводить за допомогою кнопки, з перевіркою введених значень на допустимі діапазони значень часу.

Аргумент функції `field_cnt` являє собою змінну-лічильник, що збільшується на 1 після налаштування значення кожного з полів дати та часу, `add_sub` — аргумент, котрий визначає те, відбуватиметься інкремент чи декремент поточного значення поля дати та часу — `add_sub` має приймати значення «1» або «-1» при передачі в дану функцію. Аргумент `is_alarm_cfg` встановлюється рівним «1», коли відбувається введення користувачем тільки тих значень дати та часу, котрі необхідні для налаштування часу спрацювання будильника — це, в даному випадку, значення дати, годин та хвилин. Аргумент `is_clock_cfg` встановлюється рівним «1», коли потрібно заповнити відповідну структуру значеннями усіх полів регістрів дати та часу — вводячи, за допомогою пристроїв введення, послідовно значення років, місяців, дати, дня тижня, годин, хвилин та секунд.

Всі значення вводяться в звичайному десятковому форматі, і згодом, за допомогою відповідної функції — `fill_RTC_struct_full` — розбиваються, за потреби, на

значення одиниць та десятків відповідних полів дати та часу, щоб в такому вигляді внести їх в регістри модуля RTC.

3.3. Функція `fill_RTC_struct_full`.

```
void fill_RTC_struct_full(RTC_struct_brief volatile *br_data, RTC_struct_full volatile
*f_data, uint32_t is_clock_cfg_mode)

{

	// get brief data format --> fill full data format

	// is_clock_cfg_mode == 0 --> alarm_cfg mode --> need only date, hour and minute
tens/units



	f_data->year_tens = (is_clock_cfg_mode) ? (br_data->years / 10) : (f_data-
>year_tens);

	f_data->year_units = (is_clock_cfg_mode) ? (br_data->years - (f_data->year_tens
* 10)) : (f_data->year_units);



	f_data->week_day = (is_clock_cfg_mode) ? br_data->week_day : (f_data->week_day);



	f_data->month_tens = (is_clock_cfg_mode) ? (br_data->months / 10) : (f_data-
>month_tens);

	f_data->month_units = (is_clock_cfg_mode) ? (br_data->months - (f_data-
>month_tens * 10)) : (f_data->month_units);



	f_data->date_tens = br_data->date / 10;

	f_data->date_units = (br_data->date - (f_data->date_tens * 10));



	f_data->hour_tens = br_data->hours / 10;

	f_data->hour_units = (br_data->hours - (f_data->hour_tens * 10));



	f_data->minute_tens = br_data->minutes / 10;

	f_data->minute_units = (br_data->minutes - (f_data->minute_tens * 10));



	f_data->second_tens = (is_clock_cfg_mode) ? (br_data->seconds / 10) : (f_data-
>second_tens);

	f_data->second_units = (is_clock_cfg_mode) ? (br_data->seconds - (f_data-
>second_tens * 10)) : (f_data->second_units);

}
```

Дана функція необхідна для того, щоб заповнити поля структури типу `fill_RTC_struct_full` значеннями дати та часу, котрі беруться зі структури типу `RTC_struct_brief`, за принципом, що залежить від значення вхідного аргументу `is_clock_cfg_mode.` Загалом, дана структура типу `fill_RTC_struct_full` містить значення дати та часу, які згодом зберігаються в регістрах дати та часу модуля RTC при налаштуванні значень поточного часу, або ж при налаштуванні значень дати та часу спрацювання сигналу будильника (Alarm A, в даному випадку). При цьому, за рахунок того, що при налаштуванні роботи режиму Alarm A, було налаштовано маскування певних можливих полів дати та часу, значення яких ігноруватиметься при порівнянні налаштованого значення та поточного значення лічильника для режиму Alarm A, в підсумку — для налаштування дати та часу спрацювання будильника, необхідно вказати тільки значення десятків дати, одиниць дати, десятків годин, одиниць годин, десятків хвилин, одиниць хвилин. Відповідно, немає потреби в виконанні зайвих перетворень, тому в такому випадку, значення «зайвих» полів просто перезаписується старим значенням.

4. Початкова ініціалізація-запуск роботи модуля RTC.

```
void RTC_init(void)

{

// Перш за все, перевіряємо прапорець INITS в регістрі RTC_ISR,
// щоб переконатись в тому, що модуль не є уже налаштованим,
// і що дійсно є сенс проводити процедуру ініціалізації його роботи

      if (RTC->ISR & RTC_ISR_INITS)

      {

            return;

      }



      // дозвіл тактування модуля PWR

      RCC->APB1ENR |= RCC_APB1ENR_PWREN;



      // дозвіл доступу і запису в область backup-регістрів RTC,
      // які є захищеними від запису

      PWR->CR |= PWR_CR_DBP;



      // виконається тільки при першій ініціалізації – скидання значень
      // в backup-регістрах при першому запуску модуля

      if (!(RCC->BDCR & RCC_BDCR_RTCEN))
```

```
	{
		// скидання значень в backup-регістрах
		RCC->BDCR |= RCC_BDCR_BDRST;
		RCC->BDCR &= ~RCC_BDCR_BDRST;
	}

	// запуск LSE – зовнішнє низькочастнотне джерело тактування
	RCC->BDCR |= RCC_BDCR_LSEON;

	// чекаємо, поки LSE запуститься, про що сповістить прапорцем «READY»
	while (!(RCC->BDCR & RCC_BDCR_LSERDY));

	// вибираємо LSE в якості джерела тактування для модуля RTC
	RCC->BDCR |= RCC_BDCR_RTCSEL_0;
	RCC->BDCR &= ~RCC_BDCR_RTCSEL_1;

	// вмикаємо модуль RTC
	RCC->BDCR |= RCC_BDCR_RTCEN;

	// функція для налаштування wakeup-режиму
	RTC_auto_wakeup_enable();
	// функція для налаштування Alarm-режиму
	RTC_alarm_init();
	// функція для початкового налаштування
	// регістрів дати та часу (вибір формату 12/24 і т.п.)
	RTC_data_init();
}
```

5. Конфігурація роботи модуля RTC, налаштування формату відображення часу.

```
void RTC_data_init(void)
{
	// знімаємо захист від запису в регістри модуля RTC шляхом послідовного
	// запису даних значень в регістр RTC_WPR
```

```
    RTC->WPR = 0xCA;

    RTC->WPR = 0x53;


    // входимо в режим ініціалізації – лічильник модуля RTC при цьому зупиняється,
    // і ми можемо оновити значення дати та часу вручну

    RTC->ISR |= RTC_ISR_INIT;


    // поллінг (опитування) прапорця INITF,
    // котрий сигналізує про активацію режиму ініціалізації

    while (!(RTC->ISR & RTC_ISR_INITF));


    // налаштування значення синхронного переддільника

    RTC->PRER |= 0xFF; // 255


    // налаштування значення асинхронного переддільника,
    // окремою інструкцією запису – з ціллю отримати сигнал
    // з частотою 1 Гц з вхідного сигналу тактування модуля RTC – загалом,
    // ці значення є табличними і наводяться в документації

    RTC->PRER |= (0x7F << 16);    // 127


    // очистимо регістри дати та часу

    RTC->TR = 0x00000000;

    RTC->DR = 0x00000000;


    // вибір 24-годинного формату – біт FMT == 0 в регістрі RTC_CR

    RTC->CR &= ~RTC_CR_FMT;


    // виходимо з режиму ініціалізації, при цьому регістри дати та часу оновлюються
    // відповідно до заданих вище значень

    RTC->ISR &= ~RTC_ISR_INIT;


    // активуємо захист регістрів модуля RTC від запису, шляхом запису з регістр WPR
    // значення, що відмінне від «0xCA» та «0x53»

    RTC->WPR = 0xFF;

}
```

Фактично, замість значення `0xFF`, котре записується в регістр `WPR`, для активації захисту регістрів модуля RTC від запису, можна використати будь-яке інакше

значення — воно має бути відмінним від значень, котрі використовуються для зняття даного блокування — так як запис будь-якого відмінного від цих значень, значення, призводить до активації цього захисту.

6. Налаштування-оновлення поточних значень дати та часу.

```
void RTC_data_update(RTC_struct_full volatile *f_data)

{

	// змінні, що зберігатимуть значення часу та дати

	uint32_t time_value, date_value;

	// очищуємо ці змінні
	// (фактично, можна було б і просто прирівняти їх до нуля
	// (ініціалізувати нулем), наприклад)

	time_value = time_value ^ time_value;

	date_value = date_value ^ date_value;


	// знімаємо захист від запису в регістри модуля RTC, шляхом послідовного

	// запису в регістр WPR наступних значень

	RTC->WPR = 0xCA;

	RTC->WPR = 0x53;


	// входимо в режим ініціалізації, що зупиняє лічильник модуля RTC

	RTC->ISR |= RTC_ISR_INIT;


	// чекаємо активації режиму ініціалізації

	while (!(RTC->ISR & RTC_ISR_INITF));


	// налаштування значення синхронного переддільника вхідного тактового сигналу,
	// згідно документації

	RTC->PRER |= 0xFF; // 255


	// налаштування значення асинхронного переддільника вхідного тактового сигналу,
	// згідно документації
	RTC->PRER |= (0x7F << 16);    // 127

	// очищуємо значення в регістрі часу

	RTC->TR = 0x00000000;
```

```
	// вносимо потрібні значення в змінну, що зберігає значення часу,
	// за допомогою зсуву на необхідну
	// позицію в цій змінній

	// значення часу беруться з структури типу RTC_struct_full

	time_value |= ((f_data->hour_tens << RTC_TR_HT_Pos) | (f_data->hour_units << 
RTC_TR_HU_Pos) | (f_data->minute_tens << RTC_TR_MNT_Pos) | (f_data->minute_units << 
RTC_TR_MNU_Pos));

	time_value |= ((f_data->second_tens << RTC_TR_ST_Pos) | (f_data->second_units << 
RTC_TR_SU_Pos));



	// записуємо значення зі змінної, що зберігає значення часу, в регістр часу

	RTC->TR = time_value;



	// очищуємо значення в регістрі дати

	RTC->DR = 0x00000000;



	// вносимо потрібні значення в змінну, що зберігає значення дати,
	// за допомогою зсуву на необхідну
	// позицію в цій змінній

	// значення часу беруться з структури типу RTC_struct_full

	date_value |= ((f_data->year_tens << RTC_DR_YT_Pos) | (f_data->year_units << 
RTC_DR_YU_Pos) | (f_data->week_day << RTC_DR_WDU_Pos) | (f_data->month_tens << 
RTC_DR_MT_Pos) | (f_data->month_units << RTC_DR_MU_Pos) | (f_data->date_tens << 
RTC_DR_DT_Pos) | (f_data->date_units << RTC_DR_DU_Pos));



	// записуємо значення зі змінної, що зберігає значення дати, в регістр дати

	RTC->DR = date_value;



	// вибір 24-годинного формату часу

	RTC->CR &= ~RTC_CR_FMT;



	// вихід з режиму ініціалізації

	RTC->ISR &= ~RTC_ISR_INIT;



	// активуємо захист запису в регістри модуля RTC

	RTC->WPR = 0xFF;

}
```

7. Запит та отримання значень поточних дати та часу.

```
void RTC_get_time(RTC_struct_brief volatile *br_data)

{

	// загалом, модуль влаштований таким чином, що регістри дати, часу
	// та subsecond-регістр, що містить частини секунд – RTC_DR, RTC_TR та RTC_SSR –
	// це так звані «тіньові регістри», що дублюють регістри основного лічильника
	// модуля
	// відповідно, кожні два такти частоти тактування модуля RTC відбувається
	// операція копіювання значень з регістрів лічильника в ці тіньові регістри

	// за допомогою даної операції відбувається перевірка того, чи вдало пройшло
	// копіювання значень дати та часу в тіньові регістри, і чи актуальні там
	// значення (чи синхронізованими є ці значення)

	while (!(RTC->ISR & RTC_ISR_RSF)); //  Calendar shadow registers synchronized



	// буферні змінні під значення дати та часу

	uint32_t TR_buf = 0, DR_buf = 0;



	// заносимо в буферну змінну значення з регістру часу

	TR_buf = (RTC->TR);



	// значення в регістрі часу та дати зберігаються в BCD-форматі,
	// де одиниці і десятки числа зберігаються окремо, відповідно,
	// спершу необхідно очистити зайві біти в регістрі, шляхом застосування
	// операції (RTC->DR & RTC_DR_DT) (для Data Tens – десятків значення дати)
	// (побітове і), і тоді,
	// за допомогою зсувів можна виокремити десятки та одиниці числа,
	// помножити число «десятки» на 10, додати до числа «одиниці»,
	// і, таким чином, перейти до звичного десяткового запису значення дати та часу



	// заповнюємо поля структури-прийомника типу RTC_struct_brief отриманими
	// в результаті арифметичних перетворень значненями, в звичайному
	// десятковому вигляді

	br_data->hours = ((((TR_buf & RTC_TR_HT) >> RTC_TR_HT_Pos) * 10) + ((TR_buf &
RTC_TR_HU) >> RTC_TR_HU_Pos));

	br_data->minutes = ((((TR_buf & RTC_TR_MNT) >> RTC_TR_MNT_Pos) * 10) + ((TR_buf
& RTC_TR_MNU) >> RTC_TR_MNU_Pos));

	br_data->seconds = ((((TR_buf & RTC_TR_ST) >> RTC_TR_ST_Pos) * 10) + ((TR_buf &
RTC_TR_SU) >> RTC_TR_SU_Pos));



	// заносимо в буферну змінну значення дати

	DR_buf = (RTC->DR);
```

```
	// аналогічно, конвертуємо значення дати в десятковий вигляд, і заносимо
	// їх в поля відповідної структури

	br_data->years = ((((DR_buf & RTC_DR_YT) >> RTC_DR_YT_Pos) * 10) + ((DR_buf &
RTC_DR_YU) >> RTC_DR_YU_Pos));

	br_data->months = ((((DR_buf & RTC_DR_MT) >> RTC_DR_MT_Pos) * 10) + ((DR_buf &
RTC_DR_MU) >> RTC_DR_MU_Pos));

	br_data->date = ((((DR_buf & RTC_DR_DT) >> RTC_DR_DT_Pos) * 10) + ((DR_buf &
RTC_DR_DU) >> RTC_DR_DU_Pos));

	br_data->week_day = ((DR_buf & RTC_DR_WDU) >> RTC_DR_WDU_Pos);


	// сигналізуємо про успішну операцію отримання значень дати та часу

	time_get_done = 1;

}
```

Варто також відмітити те, що при зчитуванні значень в регістрів дати та часу, необхідно послідовно зчитувати значення **обох** регістрів.

8. Початкова ініціалізація роботи режиму будильника (Alarm).

```
void RTC_alarm_init(void)

{

	// знімаємо захист регістрів модуля RTC від запису

	RTC->WPR = 0xCA;

	RTC->WPR = 0x53;


	// вимикаємо Alarm A

	RTC->CR &= ~RTC_CR_ALRAE;


	// чекаємо, поки встановиться прапорець дозволу запису в регістр RTC_ALRMAR

	while (!(RTC->ISR & RTC_ISR_ALRAWF));


	// маскуємо поля дати, годин, хвилин, секунд, котрі не будуть
	// враховуватись при порівнянні з заданими значеннями дати та часу
	// спрацювання сигналу Alarm A

	// задаємо потребу співпадіння дати

	RTC->ALRMAR &= ~RTC_ALRMAR_MSK4;   // 0: Alarm A set if the date/day match

	// задаємо потребу співпадіння годин

	RTC->ALRMAR &= ~RTC_ALRMAR_MSK3;   // 0: Alarm A set if the hours match

	// задаємо потребу співпадіння хвилин
```

```
	RTC->ALRMAR &= ~RTC_ALRMAR_MSK2;   // 0: Alarm A set if the minutes match

	// задаємо відсутність потреби співпадіння секунд

	RTC->ALRMAR |= RTC_ALRMAR_MSK1;    // 1: Seconds don’t care in Alarm A
comparison


	// поле DU[3:0] в регістрі RTC_ALRMAR буде вказувати на дату (WDSEL == 0),
	// а не на день тижня (WDSEL == 1)

	RTC->ALRMAR &= ~RTC_ALRMAR_WDSEL;  // DU[3:0] field represents the date units

	// вибір 24-годинного формату

	RTC->ALRMAR &= ~RTC_ALRMAR_PM;     // 0: AM or 24-hour format


	// дозвіл генерації переривання RTC Alarm interrupt на лінії EXTI Line 17

	// дозвіл тактування системного контролеру SYSCFG
	// (System configuration controller) через шину APB2

	if (!(RCC->APB2ENR & RCC_APB2ENR_SYSCFGEN))

	{

	      RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

	}

	// знімаємо маскування переривання з лінії 17

	EXTI->IMR |= EXTI_IMR_IM17;  // interrupt request mask - IM17 is not masked now

	// спрацювання переривання по передньому фронту на лінії 17

	EXTI->RTSR |= EXTI_RTSR_TR17;// rising edge trigger enabled for EXTI line 17


	// дозвіл генерування переривання по спрацюванню сигналу Alarm A в NVIC

	NVIC_EnableIRQ(RTC_Alarm_IRQn);    // enable the RTC_Alarm IRQ channel in the
NVIC

	// очищуємо pending flag переривання Alarm A

	NVIC_ClearPendingIRQ(RTC_Alarm_IRQn);

	// найвищий пріоритет

	NVIC_SetPriority(RTC_Alarm_IRQn, 0);     // highest priority


	// активуємо переривання по спрацюванню сигналу Alarm A

	RTC->CR |= RTC_CR_ALRAIE;


	// активуємо захист від запису регістрів модуля RTC
```

```
	RTC->WPR = 0xFF;


	// глобальний дозвіл переривань
	__enable_irq();  // global interrupts enable
}
```

9. Налаштування-оновлення значень дати та часу спрацювання сигналу будильника.

```
void RTC_alarm_update(RTC_struct_full volatile *f_data)
{
	// знімаємо захист від запису регістрів модуля RTC
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;


	// деактивуємо спрацювання сигналу Alarm A
	RTC->CR &= ~RTC_CR_ALRAE;


	// чекаємо, поки встановиться прапорець дозволу запису в регістр RTC_ALRMAR
	while (!(RTC->ISR & RTC_ISR_ALRAWF));


	// вносимо необхідні значення десятків дати в регістр дати та часу
	// спрацювання сигналу Alarm A
	RTC->ALRMAR |= (f_data->date_tens << RTC_ALRMAR_DT_Pos);  // Bits 29:28 DT[1:0]:
Date tens in BCD format
	// вносимо необхідні значення одиниць дати в регістр дати та часу
	// спрацювання сигналу Alarm A
	RTC->ALRMAR |= (f_data->date_units << RTC_ALRMAR_DU_Pos); // Bits 27:24 DU[3:0]:
Date units or day in BCD format.
	// вносимо необхідні значення десятків годин в регістр дати та часу
	// спрацювання сигналу Alarm A
	RTC->ALRMAR |= (f_data->hour_tens << RTC_ALRMAR_HT_Pos);  // Bits 21:20 HT[1:0]:
Hour tens in BCD forma
	// вносимо необхідні значення одиниць годин в регістр дати та часу
	// спрацювання сигналу Alarm A
	RTC->ALRMAR |= (f_data->hour_units << RTC_ALRMAR_HU_Pos); // Bits 19:16 HU[3:0]:
Hour units in BCD format.
	// вносимо необхідні значення десятків хвилин в регістр дати та часу
	// спрацювання сигналу Alarm A
```

```
	RTC->ALRMAR |= (f_data->minute_tens << RTC_ALRMAR_MNT_Pos);	// Bits 14:12 MNT[2:0]: Minute tens in BCD format.

	// вносимо необхідні значення одиниць хвилин в регістр дати та часу
	// спрацювання сигналу Alarm A

	RTC->ALRMAR |= (f_data->minute_units << RTC_ALRMAR_MNU_Pos);	// Bits 11:8 MNU[3:0]: Minute units in BCD format.


	// активуємо спрацювання сигналу Alarm A

	RTC->CR |= RTC_CR_ALRAE;


	// дозвіл генерації переривання RTC Alarm interrupt на лінії EXTI Line 17

	// дозвіл тактування системного контролеру SYSCFG
	// (System configuration controller) через шину APB2

	if (!(RCC->APB2ENR & RCC_APB2ENR_SYSCFGEN))

	{

		RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

	}

	// знімаємо маскування переривання з лінії 17

	EXTI->IMR |= EXTI_IMR_IM17;  // interrupt request mask - IM17 is not masked now

	// спрацювання переривання по передньому фронту на лінії 17

	EXTI->RTSR |= EXTI_RTSR_TR17;// rising edge trigger enabled for EXTI line 17


	// дозвіл генерування переривання по спрацюванню сигналу Alarm A в NVIC

	NVIC_EnableIRQ(RTC_Alarm_IRQn);	// enable the RTC_Alarm IRQ channel in the NVIC

	// очищуємо pending flag переривання Alarm A

	NVIC_ClearPendingIRQ(RTC_Alarm_IRQn);

	// найвищий пріоритет

	NVIC_SetPriority(RTC_Alarm_IRQn, 0);	// highest priority


	// активуємо переривання по спрацюванню сигналу Alarm A

	RTC->CR |= RTC_CR_ALRAIE;


	// активуємо захист від запису регістрів модуля RTC

	RTC->WPR = 0xFF;
```

```
	// глобальний дозвіл переривань
	__enable_irq();   // global interrupts enable
}
```

10. Деактивація запланованого сигналу будильника.

```
void RTC_alarm_disable(void)
{
	// прирівнюємо до нуля значення, що міститься в глобальній змінній
	// котра сигналізує про активність сигналу Alarm A
	alarm_enable = 0; // clear enable flag

	// знімаємо захист від запису регістрів модуля RTC
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

	// вимикаємо Alarm A
	RTC->CR &= ~RTC_CR_ALRAE;

	// чекаємо, поки встановиться прапорець дозволу запису в регістр RTC_ALRMAR
	while (!(RTC->ISR & RTC_ISR_ALRAWF));

	// за допомогою XOR очищуємо поля регістру RTC_ALRMAR,
	// котрі містять значення дати та часу спрацювання сигналу Alarm A
	RTC->ALRMAR &= (RTC_ALRMAR_DU ^ RTC_ALRMAR_DU);      // Bits 27:24 DU[3:0]: Date
units or day in BCD format.
	RTC->ALRMAR &= (RTC_ALRMAR_HT ^ RTC_ALRMAR_HT);      // Bits 21:20 HT[1:0]: Hour
tens in BCD forma
	RTC->ALRMAR &= (RTC_ALRMAR_HU ^ RTC_ALRMAR_HU);      // Bits 19:16 HU[3:0]: Hour
units in BCD format.
	RTC->ALRMAR &= (RTC_ALRMAR_MNT ^ RTC_ALRMAR_MNT);    // Bits 14:12 MNT[2:0]:
Minute tens in BCD format.
	RTC->ALRMAR &= (RTC_ALRMAR_MNU ^ RTC_ALRMAR_MNU);    // Bits 11:8 MNU[3:0]:
Minute units in BCD format.

	// активуємо захист від запису регістрів модуля RTC
	RTC->WPR = 0xFF;
}
```

11. Обробка переривань, що генеруються при спрацюванні сигналу будильника.

```
void RTC_Alarm_IRQHandler(void)

{

	// глобальна змінна-прапорець, що сигналізує про спрацювання сигналу Alarm A

	is_alarm = 1;	// display


	// знімаємо захист від запису регістрів модуля RTC

	RTC->WPR = 0xCA;

	RTC->WPR = 0x53;


	// вимикаємо Alarm A

	RTC->CR &= ~RTC_CR_ALRAE;

	// прирівнюємо до нуля значення, що міститься в глобальній змінній
	// котра сигналізує про активність сигналу Alarm A

	alarm_enable = 0;


	// чекаємо, поки встановиться прапорець дозволу запису в регістр RTC_ALRMAR

	while (!(RTC->ISR & RTC_ISR_ALRAWF));


	// за допомогою XOR очищуємо поля регістру RTC_ALRMAR,
	// котрі містять значення дати та часу спрацювання сигналу Alarm A

	RTC->ALRMAR &= (RTC_ALRMAR_DU ^ RTC_ALRMAR_DU);	// Bits 27:24 DU[3:0]: Date
units or day in BCD format.

	RTC->ALRMAR &= (RTC_ALRMAR_HT ^ RTC_ALRMAR_HT);	// Bits 21:20 HT[1:0]: Hour
tens in BCD forma

	RTC->ALRMAR &= (RTC_ALRMAR_HU ^ RTC_ALRMAR_HU);	// Bits 19:16 HU[3:0]: Hour
units in BCD format.

	RTC->ALRMAR &= (RTC_ALRMAR_MNT ^ RTC_ALRMAR_MNT);	// Bits 14:12 MNT[2:0]:
Minute tens in BCD format.

	RTC->ALRMAR &= (RTC_ALRMAR_MNU ^ RTC_ALRMAR_MNU);	// Bits 11:8 MNU[3:0]:
Minute units in BCD format.


	// активуємо захист від запису регістрів модуля RTC

	RTC->WPR = 0xFF;


	// очищуємо прапорець, котрий встановився в «1» при спрацюванні сигналу
	// Alarm A
```

```
	RTC->ISR &= ~RTC_ISR_ALRAF;  //  flag is cleared by software by writing 0

	// очищуємо pending flag

	EXTI->PR |= EXTI_PR_PR17;    // clear pending flag

}
```

12. Запит та отримання значень дати та часу спрацювання сигналу будильника.

```
void RTC_get_alarm(RTC_struct_brief volatile *br_data)

{

	// як і при отриманні поточних значень дати та часу, потрібно перевести
	// значення дати та часу спрацювання будильника, в десятковий вигляд

	// здійснюємо арифметичні операції, приводячи значення дати та часу до
	// десяткового вигляду, і вносимо отримані значення в поля структури
	// типу RTC_struct_brief

	br_data->date = ((((RTC->ALRMAR & RTC_ALRMAR_DT) >> RTC_ALRMAR_DT_Pos) * 10) +
((RTC->ALRMAR & RTC_ALRMAR_DU) >> RTC_ALRMAR_DU_Pos));

	br_data->hours = ((((RTC->ALRMAR & RTC_ALRMAR_HT) >> RTC_ALRMAR_HT_Pos) * 10) +
((RTC->ALRMAR & RTC_ALRMAR_HU) >> RTC_ALRMAR_HU_Pos));

	br_data->minutes = ((((RTC->ALRMAR & RTC_ALRMAR_MNT) >> RTC_ALRMAR_MNT_Pos) *
10) + ((RTC->ALRMAR & RTC_ALRMAR_MNU) >> RTC_ALRMAR_MNU_Pos));



	// сигналізуємо про успішну операцію отримання значень дати та часу
	// спрацювання будильника

	alarm_get_done = 1;

}
```

13. Налаштування wakeup-режиму роботи модуля RTC.

```
void RTC_auto_wakeup_enable(void)

{

	// знімаємо захист від запису регістрів модуля RTC

	RTC->WPR = 0xCA;

	RTC->WPR = 0x53;



	// вимикаємо Wakeup-таймер

	RTC->CR &= ~RTC_CR_WUTE;



	// виконуємо опитування, поки не отримаємо підтвердження дозволу на запис в
	// регістр лічильника з автоматичним
	// перезавантаженням та зміну значення бітів WUCKSEL[2:0]

	while (!(RTC->ISR & RTC_ISR_WUTWF));
```

```
	// задаємо значення для Wakeup-лічильника –
	// період спрацювання = 1 Гц – прапорець готовності буде встановлюватись
	// кожен такт сигналу тактування регістрів модуля RTC, котра рівна 1 Гц

	RTC->WUTR &= ~RTC_WUTR_WUT;  // the WUTF flag is set every (WUT[15:0] + 1) = (0
+ 1) = (1) ck_wut cycles



	// вибираємо джерело тактування для режиму Wakeup
	// вибираємо тактування від сигналу, котрий тактує основний лічильник модуля
	// RTC (сигнал тактування модуля RTC, котрий пройшов через всі переддільники)
	// (зазвичай, це сигнал з частотою 1 Гц)

	RTC->CR &= ~RTC_CR_WUCKSEL_1;// 10x: ck_spre (usually 1 Hz) clock is selected

	RTC->CR |= RTC_CR_WUCKSEL_2;



	// дозвіл генерації переривання RTC Wakeup – використовується лінія EXTI Line 22

	if (!(RCC->APB2ENR & RCC_APB2ENR_SYSCFGEN))

	{

	      RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

	}



	// знімаємо маскування переривання на лінії EXTI 22

	EXTI->IMR |= EXTI_IMR_IM22;  // interrupt request mask - IM22 is not masked now

	// спрацювання по передньому фронту

	EXTI->RTSR |= EXTI_RTSR_TR22;// rising edge trigger enabled for EXTI line 17



	// дозвіл генерування переривання по спрацюванню сигналу Alarm A в NVIC

	NVIC_EnableIRQ(RTC_WKUP_IRQn);// enable the RTC_WKUP IRQ channel in the NVIC

	// очищуємо pending flag

	NVIC_ClearPendingIRQ(RTC_WKUP_IRQn);

	// найвищий пріоритет

	NVIC_SetPriority(RTC_WKUP_IRQn, 0);// highest priority



	// активуємо генерування Wakeup-переривання

	RTC->CR |= RTC_CR_WUTIE;



	// активуємо Wakeup-лічильник
```

```
	RTC->CR |= RTC_CR_WUTE;


	// активуємо захист від запису регістрів модуля RTC
	RTC->WPR = 0xFF;


	// глобальний дозвіл переривань
	__enable_irq();   // global interrupts enable
}
```

14. Обробка переривань, що генеруються в wakeup-режимі роботи модуля RTC.

```
void RTC_WKUP_IRQHandler(void)
{
	// чекаємо, поки pending flag встановиться в «1»
	if(EXTI->PR & EXTI_PR_PR22)
	{
		// чекаємо, коли прапорець WUTF встановиться в «1», що свідчитиме про
		// те, що Wakeup-лічильник досяг заданого значення спрацювання
		// Wakeup-сигналу
		if(RTC->ISR & RTC_ISR_WUTF)
		{
			// програмно очищуємо прапорець WUTF
			RTC->ISR &= ~RTC_ISR_WUTF;


			// перевірка, чи активований в поточний момент часу
			// режим відображення поточного часу


			// Wakeup-режим в рамках проекту використовується для того, щоб
			// кожну 1 секунду оновлювати значення, котре виводиться
			// на LCD дисплей
			if (clock_show_mode)
			{
				// дана змінна встановлюється в «1» з частотою 1 Гц,
				// що необхідно для періодичного оновлення даних на дисплеї
				clk_1hz = 1;
				// сигналізуємо про готовність виводу даних на дисплей –
				// таким чином, періодично оновлюємо дані на дисплеї
				LCD_show_ready = 1;
```

```
                }


                // очищуємо pending flag переривання, що свідчить про те
                // що вся необхідна робота в обробнику переривань була виконана

                EXTI->PR |= EXTI_PR_PR22;     // clear pending flag

            }

        }

}
```

Посилання та документація:

1. AN3371 Application note — Using the hardware real-time clock (RTC) in STM32 F0, F2, F3, F4 and L1 series of MCUs
2. RM0368, Reference manual , STM32F401xB/C and STM32F401xD/E advanced Arm®-based 32-bit MCUs
3. STM32 RTC, Calendar: https://hubstub.ru/stm32/179-stm32-rtc-calendar.html